# A General Algebra of Business Rules for Heterogeneous Systems

Frederick V. Ramsey

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Software Technology Research Laboratory
De Montfort University

March 2007

# Table of Contents

# List of Figures

# List of Tables

## List of Listings

## Acknowledgements

When I was young, my parents often reminded me that I would be measured by the company that I keep. In the conduct of this research, I have had the good fortune to keep some very good company. It is with great pleasure that I now offer my sincerest thanks to the some of those that helped me along this journey:

- To my supervisor Professor Hussein Zedan for the guidance, direction, patience, encouragement, wisdom, and knowledge he offered. My wish to those who decide to pursue their dream of an advanced degree – may you be fortunate enough to find an advisor like Professor Zedan.
- To Dr. James Alpigini for his belief and encouragement in my work, and for his commitment of his time and energy in the support of my efforts.
- To Dr. Antonio Cau for his patience and insights in answering my many questions about ITL.
- To Professor Hongji Yang for his confidence and spirit, especially in the early days when I was just finding my way.
- To Dr. Francois Siewe for his unselfish commitment of several long days in front of a whiteboard with me, at a time when it made all the difference.
- To all my colleagues, past and present, at the lab. You have made my days in the lab some of the most enjoyable times of my professional life.
- And to my loving wife, Ellen, for her love, friendship, patience, and humor as we travel through this life together.

I dedicate this *opus* to the memory of my loving parents, Mr. and Mrs. E. M. Ramsey, Jr., gone from this world but never forgotten. Together, they taught me the most important thing that I will ever know – the power of education.

## Publications

Ramsey, F. V. & Alpigini, J. J. (2002). A simple mathematically based framework for rule extraction using Wide Spectrum Language. *Proceedings of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2002),* 44-52.

Ramsey, F. V. & Alpigini, J. J. (2002). Rough sets, guarded command language, and decision rules. *Proceedings of the Third International Conference Rough Sets and Current Trends in Computing (RSCTC 2002), Lecture Notes in Computer Science 2475*, 183-188.

Ramsey, F. V. & Alpigini, J. J. (2002). A simple mathematically based framework for rule extraction from an arbitrary programming language. *Proceedings of the 26th International Computer Software and Applications Conference (COMPSAC 2002),* 763-772.

Alpigini, J. J., Neill, C. J. & Ramsey, F. V. (2001). Classification of rule extraction techniques from knowledge-based systems. *Proceedings of the IASTED International Conference on Modeling and Simulation*, 60-64.

# Chapter 1

# Introduction

Rules give structure to knowledge. Programs use rules to dictate or constrain specific decisions or actions. Rules are incorporated into these systems based on either the experiences or expectations of the organization or a subset of knowledgeable individuals, so that all users of these systems are guided by the same knowledge and constrained to identical behaviors. Rules provide the semantic functionality of a system and represent the knowledge core around which that system is developed and maintained. Regardless of their specific form and implementation, these rule-based programs can be viewed as knowledge systems because the rules express specific domain knowledge in a usable form.

Within these programs or knowledge systems, rule revisions are typically made based on one of two factors: the organization's ever increasing understanding of its own successful practices, and the organization's response to a changing operational climate. These revisions reflect the real-time response of the organization to both internal and external changes, and also reflect the growing organizational knowledge and memory, and the associated 'state-of-the-organization.' As these rules have typically been tested, revised, and updated continuously, they represent a substantial and valuable intellectual asset.

Unfortunately, these rule revisions and updates are all too often made only within the code of these rule-based knowledge systems. As a result, no other accurate written records or documentation of these rules exists. When it becomes necessary to re-engineer these existing systems and/or create replacement systems, these valuable rules are frequently not reused because the legacy program code is the only valid source of these rules, and their extraction from the legacy code is thought to be too difficult. The problem is further exacerbated when a legacy re-engineering project potentially involves rule recovery from complex systems employing multiple programs in multiple languages. Failure to capture and reuse these rules means that the refined knowledge embodied in these rules could be, either temporarily or permanently, lost in the new system.

## 1.1 Motivation for this Research

This work was motivated by an interest in rules, their forms, and their importance, and by the recognition of the potential value of rule extraction from heterogeneous legacy systems. Based on the literature review at the initiation of this research, most rule extraction techniques reported in the literature have one or more major shortcomings that compromise their usefulness or applicability to rule extraction from heterogeneous systems. As discussed in detail in Chapter 2, these critical problems include substantial variations regarding exactly what constitutes a rule, the language specificity of many existing approaches and related tools, the functional requirement that the individual responsible for rule extraction be expert both in the knowledge domain and the program domain, inconsistencies associated with different individuals using different approaches for different languages, and the lack of mathematical formalism in most rule extraction approaches. Taken together, these five critical problems initially seemed rather daunting.

With further reflection, three core questions emerged.

1. What exactly is a rule? Specifically, can a general, succinct, formal, and robust rule definition be formulated that can be used to create, analyze, decompose, and/or understand rules?
2. Can a rule algebra be developed that allows the formal and consistent application of a formal general rule model to the extraction of rules and, as appropriate, to the creation of new rules?
3. Can a general framework be created that allows application of this rule model and rule algebra to the identification, analysis, and extraction of rules from legacy systems regardless of system, domain, platform, size, or language?

If these three questions can be answered, the reverse or re-engineering of legacy systems, and the forward engineering of new systems, can potentially be significantly improved. If these three questions can be answered, existing rule-based systems can be analyzed and new rule-based replacement systems can be developed using a consistent and complementary level of mathematical formality that is not typically applied to such tasks. The research presented in this thesis addresses these three core questions.

## 1.2 Original Contributions of this Research

This research makes an original and significant contribution in at least seven areas. Each area of original contribution is described briefly below.

1. A general formal framework for rule extraction, applicable to a wide range of legacy languages, is presented.

2. A formal general model of a rule is developed, general in that it can be adapted to the variety of languages and programming paradigms that might be encountered in different legacy code applications. Using Interval Temporal Logic (ITL), a rule is defined formally as a temporal conjunctive relationship between a state sequence describing the rule conditions and a future state sequence describing the rule outcome.

3. Using ITL and this temporal rule model, a rule algebra is developed to describe the set of operations that can be applied to compose, decompose, or transform rules. Using ITL, forty-three new lemmas are developed as part of this rule algebra and are presented in this thesis.

4. Within the context of this rule model and the associated rule algebra, various compositional paradigms are described including sequential composition; nesting; recursion; deterministic and non-deterministic guarded composition; and disjoint parallel composition. Using these compositional paradigms, rule-based representations of typical legacy code structures – the if-then-else structure, the while structure, and the indexed for-loop – are developed.

5. The strong correspondence between rules as defined in this research and statecharts is demonstrated. Using rule and statechart concepts, generic visual formalisms are developed for four common legacy-code programming structures. These statecharts are subsequently applied to represent rules extracted from different legacy programs. Whereas statecharts are typically used in the creation of new event-driven systems,

this work demonstrates that statecharts are an effective approach for analyzing and displaying hierarchical, non-event driven legacy systems.

6. The applicability of this rule model and rule algebra is demonstrated by applying them to the extraction, transformation, and analysis of rules from a diverse set of existing and legacy systems.

7. In addition to these reverse engineering applications, the forward engineering application of the rule model and rule algebra is demonstrated by developing rule-based descriptions of new software and hardware systems.

## 1.3 Organization of this Thesis

This thesis is organized as follows:

In Chapter 2, a review of the relevant literature related to rules and rule extraction is presented. Reviewed topics include rule definitions and rule models in both the forward and reverse engineering domains, code extraction, program slicing, and other reverse engineering methodologies relevant to rule extraction. Based on this literature survey, a nine-way general classification of rule extraction techniques is presented. The shortcomings of current rule extraction techniques are discussed.

In Chapter 3, a critical element necessary for a formal approach to rule extraction from legacy code is presented – a general formal framework applicable to a wide range of legacy languages. Under this rule extraction framework, general mathematical formality is introduced by describing a program as a set of language elements and structures, such that the program can be then partitioned into program structures that are or are not rules, and then analyzed accordingly.

In Chapter 4, a formal general model of a rule is developed. Starting with a state-based model of a rule, the temporal ordering of rule conditions and rule outcomes is considered. Other formalizations using temporal logic to represent and reason about the temporal relationships between states and/or state properties are reviewed. Using Interval Temporal Logic (ITL), a rule is defined formally as a temporal conjunctive relationship between a state sequence describing the rule conditions and a future state sequence describing the rule outcome.

In Chapters 5 and 6, a rule algebra is presented to describe the set of operations that can be applied to compose, decompose, or transform rules that describe specific state sequences. This rule algebra is developed incrementally by considering fundamental systems and the corresponding relationships between the state sequences that compose these systems. In developing this rule algebra, significant attention is given to composing rules and rule systems to describe larger and more complex state sequences. Various compositional paradigms are demonstrated with this rule algebra. Using these compositional paradigms, rule-based representations of typical legacy code structures are developed.

In Chapters 7, 8, and 9, the formal rule extraction framework of Chapter 3, the formal temporal rule model of Chapter 4, and the rule algebra of Chapters 5 and 6 are applied to the extraction of rules from a variety of existing systems, specifications, and legacy code, and to the forward engineering of new rule-based systems. Rules are extracted from a finite state machine, a detailed formal specification, a block of legacy Pascal code, and slices of a Wide Spectrum Language (WSL) program. In concert with this rule algebra, the use and value of statecharts for legacy code analysis is demonstrated. In addition to these rule extraction (i.e., reverse engineering) applications, the temporal rule model and the rule algebra are applied to the forward engineering of rule-based systems. These various reverse and forward applications are presented to demonstrate the wide-ranging applicability of the rule concepts developed in this research.

In Chapter 10, observations are presented regarding the development and application of the rule algebra, based on the work presented in Chapters 5 through 9.

In Chapter 11, some concluding remarks are presented and recommendations are made for possible future work relating to ideas introduced in this thesis.

# Chapter 2

# Business Rules – A Review

In this chapter, a review of current definitions and models of business rules in both the forward and reverse engineering domains is presented. Formal models of business rules are reviewed. Rule attributes common among these various definitions and models are identified, and a general definition of a business rule is proposed. Program slicing is briefly reviewed, including the application of program slicing to reverse engineering and other domains. The use of formal methods for code extraction and reverse engineering is reviewed. Rule extraction experiences are reviewed. Based on this literature survey, a nine-way, general classification of rule extraction techniques is developed. The critical shortcomings of current rule extraction techniques relative to their usefulness or applicability to the reverse engineering of heterogeneous systems and the forward engineering of new code or specifications are discussed.

## 2.1 Business Rules and Forward Engineering

Ulrich (1999) presented a two-part general definition of business rules adopted from the Object Management Group. Part 1 asserts that rules are declarations of policies or conditions that must be satisfied, and Part 2 declares that rules govern the manners in which businesses operate. The GUIDE Business Rules Project, as presented in Rouvellou et al. (2000), offered the following: "A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control the behavior of the business." This definition was extended to distinguish between constraint, invariant, derivation, and classification rules.

Perkins (2000) defines business rules as capturing or implementing precise business logic in processes, procedures, and systems. Business rules may include term definitions, data integrity constraints, mathematical and functional derivations, logical inferences, processing sequences, and relationships among data. A good business rule has three basic characteristics: (1) a rule is an explicit expression; (2) a rule is declarative, not procedural; and (3) a rule should be expressed in a single coherent model, used to express all kinds of business rules. Business rules can be implemented as metadata, process-driven approaches, and procedure-driven approaches.

Leite and Leonardi (1998) propose a business-rule taxonomy, where business rules are either functional or non-functional. Functional rules specify an organization's action, whereas non-functional rules are standards or relationships that the organization must observe. Non-functional rules are further divided into macro-system and quality rules. Macro-system rules describe policies and impose a constraint, whereas quality rules specify characteristics of an organization's standards or expectations regarding its processes or products.

For business process modeling, Presley and Rogers (1996) present a business rule model as an ontology. For the purposes of the model, this ontology is defined as a set of objects that make up a given domain, the associated properties, and the relationships among these objects that are represented in the domain terminology. This approach facilitates knowledge capture of both physical and conceptual objects and their associated relationships.

Odell (1995) investigated the nature of business rules in the context of object-oriented analysis and design using UML. Three types of constraint rules were identified: stimulus/response, operation constraint, and structure constraint. In addition, two types of derivation rules were identified: inference and computation. Stimulus/response rules specify WHEN and IF conditions that must be true for an operation to be triggered. Operation constraint rules specify conditions that must be true before and/or after an operation. Structure constraint rules specify policies or conditions about objects and their associations that cannot be violated. Inference rules specify that if certain facts are true, a specific conclusion can be inferred. Computation rules achieve their results with processing algorithms. With respect to their use, rules allow experts to specify policies or conditions in small autonomous units using explicit statements.

Ross (1997) defined a business rule as "a constraint or test exercised for the purpose of maintaining the integrity (i.e., correctness) of data." Using this definition, seven general rule classifications or families are identified: instance verifiers, type verifiers, position verifiers, functional verifiers, comparative evaluators, mathematical evaluators, and project controllers. Within each family, rules are classified into atomic types based on the specific type of computation the rule performs. In this data-centric approach, "rules compute."

Theodoulidis et al. (1992) investigated the temporal aspects of business rules. Three categories of rules were identified: constraint, derivation, and event-action. Constraint rules deal with both the static and transition integrity of structural components of the system. Derivation rules define how new static and transition components can be derived from existing system components (including other derived components), with exactly one derivation rule for each derived component. Event-action rules deal with the invocation of procedures, expressing conditions under which these procedures would be triggered.

## 2.2 Business Rules and Reverse Engineering

For the purposes of reverse engineering and legacy system analysis, Ulrich (1999) narrowed the general definition offered by the Object Management Group and concluded that a business rule is a "combination of conditional and imperative logic that changes the state of an object or data element."

For the reverse engineering domain, Sneed and Erdos (1996) defined business rules as a set of conditional operations attached to a given data result or output. Business rules are composed of four elements: results, arguments, assignments, and conditions. Arguments for business rules may come from many different sources including databases, user inputs from a terminal or window, or from other programs. Assignment and condition statements may be located throughout the program. Therefore, the authors conclude that the only easily locatable element of the business rule within an existing program is the result. Therefore, to identify or extract business rules, one must identify or know what data or output the rules produce. This definition based on output data was critical to their approach to business rule extraction.

In extracting business rules from existing systems, Shao and Pound (1999) concluded that business rules are declarative and not procedural, and they may or may not be stated explicitly within an organization except in existing program code. Business rules are classified into three groups – structural rules, behavioral rules, and constraint rules. Structural rules are statements about data objects within an organization's business. Behavioral rules are statements about the dynamic aspect or events in an organization's business. Constraint rules are about the conditions under which an organization operates.

## 2.3 Formal Models of Business Rules

Relatively few formal models of business rules exist, either in the forward or reverse engineering domains. This section presents a detailed review of those formal business rule models, with particular focus on formal representations of rule conditionals and rule-directed state transitions.

Alagar and Periyasamy (2001) present a formal specification language for formalizing business rules and business actions. This language, Business Transaction Object Z or BTOZ, is an extension of the Object-Z specification language. In general, a business rule is defined as a constraint on a business transaction, as specified or defined by the organization. A business system is formally defined as the tuple ($B$, $R$, $A$), where $B$ is a set of business objects, $R$ is a set of business rules, and $A$ is a set of agents. Every agent $A$ is responsible for enforcing rules in a single category and is aware of the business objects $B$ to which these rules apply. Every rule, $R$, is a basic predicate, abstracting a single business rule. Within this system, business actions are subject to business rules. A business action is formally defined as a generated signal ($A$, $o$, $r$), identifying that agent $A$ receives rule $r$ regarding an operation $o$. In general, each rule $r$ is written as a logical expression.

Huang et al. (1998) defined a business rule as a function, constraint, or transformation of inputs to outputs. Consistent with their research approach to use program slicing to extract rules from legacy COBOL systems, a business rule was formally defined as a program segment $F$ that transforms a set of input variables $I$ into a set of output variables $O$, such that $O = F(I)$. The subsequent forward representation of an extracted rule as a formula requires three elements: the domain variable of interest (i.e., the left-hand side of the formula); the expression for determining that domain variable (i.e., the right-hand side of the formula); and the conditions under which the formula holds.

Fu et al. (2001) studied the extraction and representation constraint rules – statements that define or constrain some aspect of a business. Operationally, constraints describe the specific conditions under which an organization operates and can appear in many forms. Focusing on this constraint reasoning, a predicate logic based language, Business Rule Language or BRL, was proposed. Within BRL, real world business objects or concepts are represented as structures. A structure is recursively defined as $S(S_1,...,S_n)$, where $S$ is the structure name and each element $S_i$ is a structure that is a component of $S$. If a given structure $S$ contains no components, it is a primitive structure; otherwise, it is a composite structure. A constraint specifies the allowable or valid states for a given structure $S$. BRL has relatively limited expressive power and includes only a small number of built-in predicates for representing the semantics of constraints captured from the reverse engineering of legacy systems. Four types of constraints are supported by BRL: Type I - constructs the domains for structures; Type II - restricts the number of instances of a given structure; Type III - specifies the relationships between two or more structures; Type IV - specifies the number of other structures that can be associated with a specific structure.

Ungureanu and Minsky (2000) defined a business rule for business-to-business e-commerce as a Law-Governed Interaction or LGI. The core concept of LGI is a policy, $P$, defined as the four-tuple $<M, G, CS, L>$, where $M$ is the set of messages regulated by this policy, $G$ is a group of agents that exchange messages from the set $M$, $CS$ is the set of control states describing the attributes of $G$ and the state of the individual agents within $G$ such that there is only one $CS$ per $G$, and $L$ is the enforced set of laws that regulate the exchange of messages between members of $G$. Events involving members of $G$ that are subject to a law $L$ of a policy $P$ are considered regulated events. For every active agent $x$ in $G$, there is a controller $C$ that assures the enforcement of $L$ for every event at $x$. The control state $CS_x$ of a given agent $x$ can be changed by primitive operations, subject to the requirement of $L$. Primitive operations used for the testing and update of control-state include true/false evaluation, addition, subtraction, removal, replacement, deliver, and forward.

To deal with the problem of the same antecedent conditions causing outcome conflicts due to multiple rules, Grosof et al. (1999) proposed a generalized version of Courteous Logic Programs (CLP). In this approach, rules are initially represented as declarative Ordinary Logic Programs (OLP) with well-founded semantics, as described by van Gelder et al. (1991). In an OLP, the head or outcome of a rule is the consequence of a series of logically connected atoms. These atoms form the body, premise, or antecedent conditions of the rule. Given that many contract terms involve conditional relationships, a rule in the e-commerce contract domain may involve an antecedent that contains multiple conjoined conditions. Rule conflict occurs when the antecedent conditions of multiple rules are satisfied, but the resulting consequences conflict. CLP extends the well-founded semantics of OLP to include prioritized conflict handling. Rule prioritization information is derived from available information such as relative specificity, recency, and authority. As a result, some rules are subject to override by other higher priority conflicting rules. Rules in CLPs are then transformed back into a semantically equivalent OLPs.

Plexousakis (1995) analyzed and simulated business processes using the high-level logic programming language GOLOG. GOLOG is based on extending the situation calculus, a first-order language for representing dynamic and evolving domains where all changes within a domain are the result of named actions, to include complex and perceptual actions. Under this approach, business processes are represented as actions that affect the domain state. In the situation calculus and in GOLOG, $A$ is a set of actions and $S$ is a set of situations. For an action $\alpha \in A$ and $s \in S$, the execution of action $\alpha$ on situation $s$ is described by $do(\alpha, s)$. Whereas all actions in the situation calculus are assumed to be primitive and deterministic, GOLOG allows complex actions through sequencing, iteration, and non-deterministic choice. GOLOG allows the specification of necessary pre- and/or post-conditions associated with a specific action. Using GOLOG, complex actions are decomposed into primitive actions, and the GOLOG language interpreter essentially acts as a theorem prover.

Koubarakis and Plexousakis (1999) presented a formal framework for business process modeling using concepts of concurrent logic programming and situational calculus. In the process submodel, actors performing actions change the situation, i.e., the current state of a system. Actions can be primitive or complex. Actions are considered primitive if decomposition reveals no additional information of interest. Primitive actions are formally defined as the tuple *<precondition, effect>*, where *precondition* and *effect* are represented by formulas written in a formal first-order model language. With these primitive actions, complex actions are formed using the syntax and semantics of ConGolog (De Giacomo et al., 2000), a concurrent version of GOLOG. Complex actions can be defined recursively and may include sequencing, waiting for a condition, non-deterministic choice of action, non-deterministic choice of action parameters, if-then-else conditionals, while-do iteration, non-deterministic iteration, concurrency, prioritized concurrency, non-deterministic concurrent iteration, interrupts, procedures, and do nothing. System state restrictions are imposed in the constraints submodel, where static and dynamic constraints are expressed using situation calculus and the symbols defined in the other relevant submodels. Although not explicitly defined as such, these actions and constraints function as the business rules that change the system state.

Herbst (1995) presents a meta-model of business rules for use in business systems analysis. This model extends the event-condition-action (ECA) rule model from the active database domain to an event-condition-action-action (ECAA) structure applicable to general business processes. Under this extension, every rule has exactly one event, no more than one condition, and only one or two actions – those resulting from the then portion of an if-then construct when the conditions are true, or those resulting from the else portion if-then-else when the conditions are not true. Events and conditions can be elementary or complex and can include recursive relationships. ECAA rules can be transformed into one or two ECA rules by negating the condition. Although this construct is not strictly formal, variations of this general construct are widely used in the logical formulation, modeling, and representation of business rules (Herbst et al., 1994).

## 2.4 General Properties of Extractable Business Rules

Although the research presented in the previous sections focused on specific issues relative to distinct needs, numerous commonalities exist with respect to what constitutes a business rule, in either the forward or reverse engineering domains. Based on the spectrum of definitions, models, criteria, and attributes presented in the available literature, the following general and informal specification of the properties of business rules is proposed:

1. Business rules are explicit. They are known, articulated, and subsequently included in the program code that constitutes the knowledge-based system. That these rules are known and stated explicitly presupposes the knowledge that they are important and therefore worth stating.

2. Business rules are precise. They are unambiguous relative to their knowledge and use domains.

3. Business rules are logically or mathematically operative on input data to create output data. This can be any combination of predicate logic and math operations taking the form of a constraint or transformation, and may consider the static, dynamic, and temporal state of the input data.

4. Business rules are imperative. If the predicate requirements are satisfied, the rule must be executed and it must be executed now. However, the specific temporal attributes of 'now' must be defined relative to the knowledge and use domains encompassed by the rule. For example, 'now' in an airline cockpit is significantly different that 'now' on a university campus. Therefore, a rule specification may presume instantaneous execution or may include specific values and conditions for this imperative element. Regardless of the imperative specifications, logically, a rule must always be executed. A complete rule that can be ignored or postponed indefinitely is not a rule.

5. Business rules are declarative and not procedural. A business rule identifies a possible output data state as either required or prohibited, but it does not specify the steps that must be taken to achieve or prohibit such a state transition.

This rule property description – explicit, precise, operative, imperative, and declarative (EPOID) – provides a consistent basis to compare and assess different rules and implementations in both the forward and reverse engineering domains. This new description provides a rational basis for exploring the role of knowledge and semantics in rule formation. Furthermore, this rule property description helps avoid confusion with other rule-driven knowledge applications and domains, where the objectives may include the discovery, application, or recovery of implicit, imprecise, or associative 'rules' from data.

## 2.5 Program Slicing

Several comprehensive reviews of program slicing techniques have been conducted (Tip, 1995; Binkley and Gallagher, 1996). Conceptually, program slicing is a decomposition technique where only those program statements contributing to a particular action or computation are identified and extracted. Tip (1995) defined a program slice as the subset of statements and control predicates of a given program that potentially influence the values computed at a specific point in that program. Weiser (1982) offered a formal definition of a slice *S* as a executable program extracted from program *P* by eliminating statements zero or more statements, such that *S* and *P* halt on the same state trajectory *T* associated with input *I*. Francel and Rugaber (1999) offered a formal definition of a program slice relative to statement *S* and variable *X* as only those code statements that might affect the value of *X* at statement *S*.

Program slicing can be differentiated in numerous different ways. Dynamic slicing versus static slicing is one common distinction. Dynamic slicing assumes fixed or specific data input for the program of interest; such that only the code reached, based on that specific data, is identified as the dynamic slice. In dynamic slicing, only the code statements traversed in the specific execution associated with that specific data are preserved as part of the slice. All other code, on the paths not taken based on the data provided, is eliminated from the slice. For example, depending on the specific data input, one branch of an IF statement would be executed and included in the slice, whereas the other would not be reachable and thus would be excluded. Static slicing makes no assumptions and imposes no limitations regarding the input data; therefore, all code that could be reached, given any data input, is identified as the static slice. In this case, both branches of an IF statement would be included in the slice. Statements can be gathered or eliminated by backward or forward traversal of the program code. Indeed, backward versus forward is another means of partitioning for different slicing techniques.

All slicing techniques require the *a priori* specification of a slicing criterion. For static slicing, this slicing criterion is the pair, that being the program statement location and variables. For dynamic slicing, specific input values are added and the slicing criterion becomes the triple, namely, input data, program statement location, and variables.

Numerous variants and hybrid approaches exist. Alternative algorithms abound for slicing under various circumstances of slicing objective, language, and programming logic. Specific implementations of program slicing in the reverse engineering domain include: condition-based slicing, forward slicing, and backward slicing (Ning et al., 1993); conditioned slicing (Fox et al., 2000; Danicic et al., 2005); generalized program slicing, recursive slicing, and hierarchical slicing (Huang et al., 1998); assignment reference slicing (Sneed and Erdos, 1996); transform slicing (Lanubile and Visaggio, 1997); forward dynamic object-oriented slicing (Song and Huynh, 1999); amorphous slicing (Binkley et al., 2000); semantic slicing (Ward 2001; Ward et al., 2005); and high-level architecture slicing (Zhao, 2000). In addition to its use in reverse engineering, program slicing has been successfully applied to other domains including software debugging, program understanding, parallelization, program differencing, program integration, software maintenance, and compiler tuning.

## 2.6 Formal Methods, Reverse Engineering, and Code Extraction

The term *formal methods* refers to methods that have a sound basis in mathematics. To date, there has been little research specifically performed in applying formal methods to the specific problem of business rule extraction. However, the use of formal methods in related areas is well-studied. Therefore, this section reviews the application of formal methods to reverse engineering and re-engineering projects in general, and to the problem of code extraction in particular.

Liu et al. (1997) reviewed the use of formal methods in the re-engineering of computing systems. A five-away classification of formal methods was developed based on model-based, logic-based, algebraic, process algebra, and net-based approaches. Within the context of this five-way classification, existing formal methods were reviewed with respect to their previous application in any reverse engineering domain. Consistent with their observation that formal methods are "both over-sold and under-used," only 4 of the 24 specific formal methods reviewed had been applied to reverse engineering. None of the reviewed formalisms was applicable to all three re-engineering stages: restructuring, reverse engineering, and forward engineering.

In research conducted as part of the REDO project, Bowen et al. (1993) described the use of formal methods in recovering specifications from COBOL applications. The overall focus of this research was to improve the maintainability, validation, transportability, and documentation of large software systems. The general process was to transform COBOL code using a succession of higher-level languages to produce a structured specification in Z++, an object-oriented extension of Z. Input COBOL code was cleaned and transformed into equivalent UNIFORM code. The intermediate language UNIFORM was developed to facilitate precise verification and code transformation. The UNIFORM code was subsequently transformed into a first order functional language. This first order functional language was then used to create the Z++ representation. With each intermediate step, implementation details were lost in favor of greater abstraction. Using the recovered specifications, a specification-based approach to maintenance was proposed based on exact the semantic associations between code and specifications.

Blazy and Facon (1997) applied formal methods to the partial evaluation, also known as program specialization, of Fortran 90 code. The objective of the overall approach was program understanding via the creation of specialized program segments based on specific input values. First, an inter-procedural pointer analysis of the code was performed. A formal specification of that analysis was developed with different formalisms, including inference rules with global definitions, as well as set and relational operators. These formal specifications were subsequently used to implement the reduced or specialized program.

Villavicencio and Oliveria (2001) combined both formal and semi-formal methods to reconstruct a formal specification from C language legacy code. The semi-formal method was code slicing, implemented first to reduce the code complexity and associated requirements of implementing formal semantics for all program variables at the same time. The functional semantics of the resulting code slices were then expressed in the HASKELL programming language. The formal basis for specification identification and extraction was the 'algebra of programming' applied in reverse order, starting with the identified output variables of interest.

Gannod and Cheng (1999) applied both informal and formal methods to the reverse engineering of large systems written in the C language into formal specifications. The overall reverse engineering process involved the construction of an informal high-level model of the software, an informal low-level model of the software including a call graph, and then, using these informal models, the selection of a specific module to which formal methods were applied. One of the major advantages of using both formal and semi-formal methods in a combined approach is that by using a semi-formal technique to guide the formal technique, the resulting formal specification will be organized based on the structure of the original program. The strongest post-condition predicate transformer (*sp*) and order-preserving transformations were applied to the selected module to develop an as-built formal specification. The strongest post-condition is the strongest condition R that is true after program S executes, when the starting specified condition Q is true. This was accomplished by the definition of C language syntax in terms of the formal semantics of the strongest post-condition predicate transformer. Eight different C programming language constructs were analyzed, including various assignment operators (e.g., = and +=), alteration constructs such as if and if-else, iteration operators such as the do-while, while, and for constructs, and function calls (Gannod and Cheng, 1996). Semantic formal equivalents using terms of the Dijkstra guarded command language were developed. The resulting as-built formal specification was then generalized using a formally defined abstraction match to remove undesired algorithmic and implementation details. These techniques have been incorporated into a suite of four tools specifically designed to assist with the understanding and reverse engineering of C language programs (Gannod and Cheng, 2001).

Zhou et al. (1999) present a language independent technique for formally assessing the critical behavior of a legacy system. By inserting assertion points at appropriate locations within the legacy code, the state of specific system attributes or variables can be monitored and understood. This approach facilitates system understanding by monitoring system states, as opposed to explicitly identifying or assessing code functionality. Within the context of rule extraction, this approach requires sufficient *a priori* knowledge regarding specific rule locations within the code to assign assertion points at appropriate locations to sufficiently monitor system behavior.

Lanubile and Visaggio (1997) presented a formal method for extracting reusable function code from poorly structured programs using the concept of the transform slice. Transform slicing potentially avoids the capture of extraneous code, as occurs with other slicing approaches. Transform slicing requires knowledge that a function is performed in the code and that the function is partially specified in terms of input and output data. Thus, a three-part slicing criterion is required for transform slicing: the function location, the input variables, and the output variables. Domain knowledge is used to identify the input and output variables. To assist with this problem of specifying the function location, a scavenging approach, using more generalized transform slicing, is used to generate a set of candidate functions from the program code. This candidate set reduces the magnitude of human intervention required to establish a starting point in the program code for extraction of a particular function. Tan and Kow (2001) implemented this transform slicing approach to identify the code elements that implement program functionality in SQL programs.

Fu et al. (2001) present a formal language for representation and presentation of business rules that have been extracted from legacy code. The problem of presenting extracted business rules as code fragments to a general audience is identified. Given that constraints can take many forms, the presentation of these constraint rules must be adapted to a specific user. The need for a new formal language is based on the observation that an extracted constraint can be obscure and difficult to comprehend when expressed using some low-level formalisms. The proposed language, Business Rule Language or BRL, is a predicate logic based language with a relative small number of built-in predicates focusing on constraint reasoning. Four types of constraints are supported and can generally be described as domain construction, instance restriction, relationship constraint, and instance association.

Yang et al. (2000) proposed the application of abstraction to reverse engineering problems and system specification recovery. For the purposes of this research, abstraction was defined as the act of hiding irrelevant details. A five-way taxonomy for abstraction was developed: weakening abstraction, hiding abstraction, temporal abstraction, structural abstraction, and data abstraction. Re-engineering wide spectrum language (RWSL) was developed to implement this abstraction approach. RWSL is a multi-layered wide spectrum language with a sound formal semantics. Case studies were presented describing the recovery of system specifications from programs written in C and ADA. Whereas this research focused on the extraction of system specifications, as opposed to extraction of individual code elements or rules, this approach is potentially an effective way of addressing the need for both a syntactic and semantic understanding of a program prior to rule identification and extraction.

Ward (2001) describes a formal transformation-based approach to source code analysis and manipulation, including code extraction. This approach uses Wide Spectrum Language (WSL), a general programming language with a theoretical foundation and with semantics that are defined formally (Ward, 1989). The general approach is the representation of a program, or a specific program element, as a function between the initial state of a given system prior to program execution and the final state of that system after program termination. By using a provable, mathematically sound transformation from a given programming language to WSL, a provably equivalent WSL program can be created. Then, by using provable program transformations within WSL, code can be moved, deleted, or merged, as appropriate, and equivalent higher level abstractions of the original program can be developed as needed. This general procedure of stepwise refinement using provable semantic-preserving changes for WSL transformations is described in Bull (1990). Using these equivalent WSL programs, the original program logic can be analyzed, specifications derived, or new code generated using a mathematically sound transformation from WSL to the new programming language. This approach has been successfully used to analyze and/or re-engineer programs where the original program code was in ADA, IBM Assembler, BASIC, CICS, COBOL, and Pascal (Bennett et al., 1992; Ward, 1999; Yang and Bennett, 1994; Zedan and Yang, 1998).

## 2.7 Business Rule Extraction

Sneed and Jandrasics (1988) reviewed the requirements of transforming software code back into specifications. Three distinct semantic levels of software, each with different abstraction levels, were described: the physical level, consisting of discrete units of code; the logical level, existing in some form of meta language describing the logical processing units; and the conceptual level, a set of abstract entities and the relationships among these entities. This conceptual level is what is typically referred to as the specification. In retransforming code back to a specification, the rule analyst must bridge the gap between the physical level where the program exists, and the conceptual level where the business model exists. Although software can be best altered or enhanced at the conceptual level, this view is frequently blocked by implementation details at the physical level. A general re-engineering plan is proposed where program elements at the physical level are mapped to design elements at the logical level. The resulting data and program design elements are then mapped to system specifications at the concept level. Within this conceptual or specification level, two alternative abstraction models are possible: macro, modeling the target system as processes and objects; and micro, modeling the target system as elementary function and data elements.

Aiken et al. (1993) reported on attempts to recover business rules, domain information, and data architectures from the Department of Defense's heterogeneous computer applications. These programs and databases included homegrown database management systems, COBOL databases, assembly language code, and MUMPS databases. The most difficult aspect of this re-engineering project was the discovery of business rules and data entities from the different types of legacy systems. To accomplish this, a "divide and conquer" approach was implemented. During the top-down phase, user screens, reports, and policy statements were reviewed to establish a high-level "as-is" business rule and data model framework. Using this high-level framework, the individual business and data model components were broken into individual components and analyzed. The bottom-up phase included the use of CASE tools, and the review of a traceability matrix, the data dictionary and data model, and supporting physical documents. Final rule identification and extraction appears to have been largely a manual process.

Ritsch and Sneed (1993) contrasted two alternatives for extracting system knowledge and business rules contained in an existing system: static analysis and dynamic analysis. Static analysis considers program data structures and program source code, including user interfaces. Dynamic analysis attempts to identify how a system, subsystem, or object responds to various inputs, including system function and performance as viewed by the user. Static analysis of the program source code and database schemes can yield information regarding database structure, database contents, entity relationships, program structure, control flow structure, decision logic, file access, and other program communications. However, static analysis cannot identify which program components are actually used, provide performance information, or relate program elements to a specific business function. Given these weaknesses of static analysis, and the typical inadequacy of system documents and the unavailability of the original system developers, a dynamic analysis methodology was proposed. This dynamic analysis approach identifies business rules as pairs of pre- and post-assertions, matching input with a specific program slice. The tool developed for this dynamic analysis consists of four elements: an instrumentor, a test monitor, an assertion generator, and a database auditor. The instrumentor inserts reporting probes at multiple locations with the program. The test monitor captures and stores the contents of the input-output panel or file for each transaction. The assertion generator matches the input and the program path with the output of each system transaction. With this, two assertion specifications are generated, one based on the input data, and one based on the output data. The database auditor logs the before-transaction state and the after-transaction state of each database file. These can then be compared and the fields altered by the transaction identified. Output from the assertion generator and the database auditor can then be combined into either formal or informal specifications.

Ning et al. (1993) described the concept of reusable component recovery, where functional components of legacy systems are identified, extracted, adapted, and reused in new system development. Whereas this approach has many advantages such as reuse, platform flexibility, and size reduction, the approach requires a thorough analysis and understanding of the legacy code, which is described as a "difficult, human dependent, and time-consuming task." To assist with this task, a tool-assisted program segmentation approach to component recovery and rule recovery was described. This two-step approach consists of a focusing step that facilitates the identification and combination of functional elements, and a factoring step that facilitates extraction of the focused functional elements into reusable packages. The focusing operation helps the analyst identify program code that is semantically related but may not be physically adjacent. Five focusing operations are used: selecting specific statements, call hierarchy analysis, condition-based slicing, forward slicing or ripple effect analysis, and backward slicing. Focusing creates localized functional code segments. In the factoring operation, these code segments are extracted and packaged into independent modules. For identifying and extracting reusable components from legacy COBOL systems, this program segmentation approach has been implemented in an Andersen Consulting proprietary tool, COBOL/SRE. This tool supports a variety of code analysis and understanding approaches, including system-level analysis, concept or functional pattern recognition, data model recovery, and program-level analysis. In addition to the program segmentation approach described above, other program-level analysis features include: parsing and syntax-directed text browsing; flow analysis, including call graphs and control flow graphs; complexity analysis; and anomaly detection.

Petry (1996) proposed a general methodology for rule extraction from programs using the concept of HyperCode, the transformation of a program source code into a hypertext-linked format to enable navigation through the program. The objective is to speed the learning and understanding of the program logic. By allowing easy navigation from one part of the code to another, rule extraction proceeds should proceed faster as compared with manual extraction. A general methodology for the use of HyperCode in rule extraction was presented. First, data entities of interest are identified. Second, the database-enforced relationships between these entities are identified. Third, the program code is parsed and the CRUD (create, retrieve, update or delete) actions taken by the program on these data items are identified. At this point, comments that create hypertext links are inserted into the program code, creating the basis for the HyperCode document. Finally, the various procedures, processes, and algorithms contained within the program are analyzed. This final step is the basis for the rule extraction process. Each program block is reviewed by the analyst and classified as either part of an overhead process, a decision process, an elementary process, or an algorithm. By eliminating the overhead processes, and understanding the relationships between the remaining decision processes, elementary processes, and algorithms, the business rules are identified and extracted. A prototype implementation using COBOL source code was presented.

Sneed and Erdos (1996) observed that, at that time, little work had been done on business rule extraction from real programs because the concept of business rule extraction had not been adequately expressed in an operational framework. It was observed that business rules can be extracted from source code only when four preconditions are met. First, variable names must be meaningful and informative. Second, the critical output data must be identifiable. Third, tools must be available to strip the program code to the essential elements. Fourth, data flow within the program must be identifiable. Reflecting their definition of business rules (presented in a preceding section) and the associated conclusion that the only easily locatable element of a business rule is the result, a reengineering tool was developed that uses the data result as a point of entry for rule extraction from COBOL programs. Using this tool, the first step of the rule extraction process is the identification of all assignment statements, including the location within the code where the target result is assigned. Next, the conditions that trigger these assignments are identified. These conditions are then linked with the associated assignments. The program is reduced to only those statements, the business rule, that create the target result. With this tool, the user identifies the result, and the other business rule elements are identified automatically.

Huang et al. (1998) identified five business rule extraction criteria:

1. The extracted business rules must be a faithful representation of the software.

2. Different groups will require different representations of the extracted business rules. Therefore, business rules must be represented in a hierarchical manner.

3. Business rules must be expressed in the domain vocabulary of the specific business application.

4. Because of the size and/or complexity of most activities, automatic rule extraction will be difficult, if not impossible. Therefore, the ideal rule extraction tool should be interactive and allow human assistance.

5. The extracted business rules should be in a form that is usable in other software maintenance activities, mapping between rule and code, and vice versa.

Consistent with their data transformation definition of business rules, a data-centered approach to rule extraction from COBOL programs was used. This rule extraction approach involved four steps: variable identification; slicing criterion identification; code extraction using generalized program slicing; and rule representation. The first step was to identify the important variables that are or can be used to express business rules. Only a small subset of the many code variables in typical business application is suitable for expressing business rules. Code variables were classified into various types, such as domain data, program data, local data, global data, input data, output data, constant data, or control data. This classification can be conducted by either parsing the code directly or by analyzing dependence graphs. Once this classification has been conducted, two heuristic rules were presented to identify input and output variables as domain variables of interest. Having identified these critical domain variables, the slicing criterion was established and the relevant code was then extracted using generalized program slicing. The objective of generalized program slicing is to extract only the code that either affects or is affected by the identified critical domain variables. Six additional heuristic rules for slicing criterion and slicing algorithm identification were presented. Given the complex nature of most rules, recursive slicing, or high-level abstraction and hierarchical slicing was recommended for most circumstances. The final step was rule presentation. Three alternatives were identified, depending on the targeted user of the extracted rule: code view, where rules are represented as code fragments; formula view, where rules are represented as variables and functions; or input-output dependency view, depicting the dataflow relationships among the variables. Selection of the specific representation of the extracted business rules is dependent on the target audience that will be using the business rules.

Sneed (1998) described a well-defined four-phase re-engineering process. First, the legacy software is measured. Next, the legacy code is reverse engineered to capture the design and evaluate to potential for reuse and re-engineering. Then, the legacy code is reorganized, restructured, or otherwise converted into separate reusable modules. Finally, these re-engineered modules are tested against the original legacy code to ensure functional equivalence. A software workbench, SOFT-REORG, developed to support this four-phase re-engineering process was described. SOFAUDIT evaluates legacy systems using seven complexity metrics and seven quality checks. SOFREDOC extracts design information from the code and associated data structures. SOFRECON allows program restructuring and program conversion. SOFRETEST allows testing of the reengineered programs against the original. With respect to program understanding and rule extraction, SOFREDOC provides ten basic views of the target program: data tree; procedure tree; decision tree; data flow diagram; macro table; constant table; business rules; call hierarchy; object reference diagram; and data reference diagram. To identify specific business rules, the data results of interest are identified by the user, and the SOFREDOC tool uses data flow and control flow slicing to identify the expression path for each selected data result. Versions of the tools have been developed for Assembler, PL/1, and COBOL.

Shao and Pound (1999) observed that business rules may be implemented in different ways in different parts of the system and different techniques may be required to recover them. Rule extraction techniques were classified into two broad groups, data understanding techniques and program understanding techniques. The objectives of data understanding techniques are to recover conceptual data models. Inputs for data understanding techniques are schema, data, program code, and transactions. Data understanding techniques are useful in recovering structural rules buried in the data and associated metadata, but they do not identify rules contained in the application programs. The objectives of program understanding techniques are to recover business rules, especially constraint and behavior rules, from these application programs. Input for program understanding techniques is straightforward – the program source code. The great majority of current program understanding techniques attempt to extract and describe the components of a given program syntactically. However, syntactic analysis does not reveal or consider the meaning of the program, and thus there is a growing interest in trying to extract and understand programs semantically. With regard to this semantically based approach, most techniques rely on a knowledge-based approach, and few tools currently exist. Most program understanding techniques do not analyze the recovered code or rules in relation to the database systems. Program-understanding techniques are most useful where business rules are embedded in programs only. To address this problem, a conceptual plan was presented for a data-centered, program understanding approach that attempts to integrate both data understanding and program understanding techniques. Using this approach, both databases and programs are analyzed together to extract constraints that may be located in application programs, data dictionaries, triggers, or stored procedures. The proposed approach consisted of three stages: preparation, extraction, and presentation. The preparation stage includes schema tools and parsing tools. The parsing tools are used to convert a program into a generic representation so that different source programs in different languages can be analyzed together. Schema tools are used to extract the conceptual data model. The extraction tools are used to analyze the generic programs and databases created with the parsing tools and schema tools, respectively. The presentation tools allow the extracted business rules to be presented differently to different users in a form that is most meaningful to them.

Sellink et al. (1999) investigated restructuring of programs written in mixed languages, in this case COBOL interspersed with CICS. The inclusion of CICS in programs results in a unique challenge in that an event-driven system structure is created independent of the host language, in this case, COBOL. Whereas this research was conducted to demonstrate the substantial improvement that could be achieved in maintainability, the experiences are equally as important and applicable to rule extraction through the improved understandability of program code. This program restructuring approach involved four steps. First, exception handling by the problematic CICS statements, including the HANDLE statement, was eliminated. Next, GOTO logic, which result in unstructured code, was removed, and control flow was structured into a series of subroutines, in this case PERFORM structures in COBOL. Next, the processing logic was restructured by removing explicit jump instructions and eliminating redundant code. Finally, the code was repartitioned so that the transaction processing logic was isolated from the business logic or rules. With this approach, all CICS commands are replaced with COBOL CALLs to a wrapper, allowing elements of the old program functionality and the associated the event-driven structure to be implemented in a modern language such as C++ or Java.

Ulrich (1999) described a process of code segmentation and code reduction to facilitate rule identification and extraction. In general, business rule extraction requires a high-level assessment of the target application so that the system can be segmented prior to actual rule extraction. This segmentation process may include resolving identified program weaknesses, restructuring convoluted logic, splitting large modules, and variable name rationalization or enrichment. Although these code improvement techniques may ultimately simplify the rule extraction process, the time and effort required to accomplish these high-level assessment tasks should be carefully considered before beginning. Only about 20 to 30 percent of source code within a given application relates to actual business rules; the remaining 70 to 80 percent of the code typically deals with non-business logic, such as physical operation, execution, and environment requirements. A general procedure for identifying and subsequently discarding this non-business logic and code was presented. This procedure requires the identification of specific code or program elements that will not contain business rules, including syntactically dead logic, semantically dead code, initialization logic, input/output logic, output area and report build logic, I/O status checking, error handling logic, data structure manipulation, special environment logic, and extraneous and superfluous logic. Once identified, these non-rule program and logic structures can be ignored, and the remaining portions of the code searched for program and logic structures that may contain business rules. These rule-containing structures may include those leading to the creation of a specific output variable, those linked to a specific conditional, or those specifically associated with an input transaction. Rules that are identified can then be logged for evaluation and possible reuse.

Numerous researchers have investigated the use of visualization techniques to elicit program structure from a variety of different program languages. Whereas an understanding of program structure does not explicitly identify business rules, enhanced program structure understanding can assist the rule analyst in the identification of critical program segments that may contain important business rules.

Call graphs represent the most basic and possibly the most widely used visual representation of program structure. Call graphs identify and present calls between entities in a program, thereby representing binary relationships between entities in a program. Murphy et al. (1996) conducted an empirical quantitative evaluation of five different call graph extractors for the C language. Substantial variation in output was observed between the five extractors, with most returning different call information from the same test program. This was largely due to different treatment of program elements such as macros, function pointers, and inconsistent interpretations of syntactic constructs.

Other researchers have extended the call graph paradigm to further enhance user functionality and facilitate program structure understanding. Feijs and de Jong (1998) described a proprietary 3-D visualization system in which various program module types are displayed as different LEGO-like bricks, and the interrelationships of these modules are depicted as different colored arrows. The resulting interconnected web of program modules and relationships are displayed such that different views, scales, and levels of information can be selected by the analyst. Mancoridis et al. (1999) developed a graphical clustering tool that created a system decomposition diagram by treating clustering as an optimization problem. Within a test system, modules and dependencies are mapped to Module Dependency Graph (MDG). Formally, a MGD is the set (*M, R*) where *M* is the set of named modules within a system and *R* is the set of dependencies between modules. The graphical clustering tool, Bunch, generates a visually simplified graph through the automatic and user-directed specification of subsystems or clusters of program modules. In contrast to completely automatic systems, this approach is especially useful in programs with a large number of modules, as the number of potential subsystem partitions grows exponentially with the number of program modules.

Storey and Muller (1995) investigated the use of a specialized nested graph technique, the fisheye technique, in the visualization of program structures in very large legacy systems. Nested graphs are composed of nodes, representing software artifacts, and of arcs, representing dependencies between these artifacts including call dependencies. Nodes and arcs can be either atomic or composite. Composite nodes represent software subsystems, and composite arcs represent a collection of dependencies. Through the nesting of nodes, the hierarchical structure of the system can be represented. Nested graphs allow multiple levels of abstraction to be visualized. Fisheye techniques allow the user to investigate a specific subsystem graph by selectively highlighting nodes within a specified area of interest, while simultaneously reducing the remaining portion of the graph. This traditional fisheye approach was expanded with the Simplified Hierarchical Multi-Perspective (SHriMP) technique, which creates views that can show multiple graphical perspectives of the program concurrently.

## 2.8 A General Classification of Rule Extraction Techniques

As different legacy knowledge systems present different challenges, most rule extraction experiences presented in the literature include the use of multiple extraction techniques. Based on this literature survey, the following nine-way, general classification of rule extraction techniques is presented. In practice, actual rule extraction typically involves several of these nine techniques used in a sequential manner.

Semantic Enrichment – This class of techniques assists the human analyst in the semantic understanding of the program and the associated business rules. Semantic enrichment can be applied to any program entity name – variables, constants, functions, procedures, etc. Numerous researchers (Shao and Pound, 1999; Sneed and Erdos, 1996; Ning et al., 1993; Aiken et al., 1993) have identified the importance of meaningful and understandable entity names as a critical element of rule extraction. This technique attempts to address, at the most basic level, the recurrent need to link the semantic elements of the system at the conceptual level with the syntactic elements of the code at the operational level. The value of this technique is directly related to the unavoidable fact that most rule extraction techniques still involve a high level of human intervention and interpretation in the rule extraction process.

Code Reduction – This class of techniques deletes those portions of program code that do not contain business rules. By eliminating this extraneous code, the manual or automatic process of rule extraction is made that much easier. The eliminated code typically involves program overhead activities, including input/output activities, error handling, and any special environment requirements. Code reduction has been directly used by a number of researchers to simplify the rule extraction overall task (Ulrich, 1999; Sneed and Erdos, 1996; Petry, 1996). Given that 70 to 80 percent of a typical program is not related to business rules (Ulrich, 1999), code reduction can be used to significantly reduce the magnitude of the rule extraction task. Within the context of the EPOID extractable business rule definition, any program element that is not logical or mathematically operative could be eliminated via code reduction.

Program Segmentation and Restructuring – This class of techniques focuses on the process of breaking large blocks of program code into smaller, autonomous segments or objects that can be more easily managed, reassembled, and understood. The general classification of program segmentation and restructuring may include code reorganization, reformatting, and remodularization. Numerous researchers have incorporated code segmentation into their overall rule extraction strategy (Ulrich, 1999; Sneed, 1998; Ning et al., 1993; Aiken et al., 1993) to bring semantically related portions of the code together physically. The substantial value of code segmentation and restructuring in program understanding had been demonstrated experimentally (Penteado et al., 1999; Sellink et al., 1999).

Data Structure/Model Analysis – This class of techniques focuses on the identification and/or recovery of database conceptual data models so that these models can then be used to support other specific rule recovery techniques. These techniques typically focus on the recovery of relationship, structure, and constraint information that may be available from the schema and associated metadata (Shao and Pound, 1999). These techniques may also include deriving or imposing data models on legacy systems that may have been developed without a formal data model (Aiken et al., 1993), or extracting data structures from program code (Petry, 1996). Although these data models may or may not contain any explicit business rules, information obtained through data structure/model analysis is typically a critical input to other rule extraction techniques. Entities recovered using these techniques directly address the EPIOD rule requirement that data elements of a rule be precise; that is, unambiguous relative to their knowledge and use domains.

Program Structure Analysis – This class includes a broad spectrum of techniques designed to identify the program hierarchy of functions, procedures, subroutines, paragraphs, objects, etc. Specific implementations by various researchers relative to rule extraction include call hierarchy analysis (Ning et al., 1993); procedure tree and decision tree analyses (Sneed, 1998); a three-step process of local analysis, use analysis (a recursive step), and global analysis (Gannod and Cheng, 1996); and a hypertext-assisted approach (Petry, 1996).

Data Flow Analysis – This class of techniques identifies the steps by which data inputs become program outputs, and may include the analysis of program decision logic and control flow. Researchers that have directly considered data flow relative to business rule extraction include Sneed (1998), Huang et al. (1998), and Gannod and Cheng (1999). The output of these data flow techniques can be used to directly address the operative component of the EPOID business rule definition; that is, rules operate on input data to create output. In doing so, these techniques are ultimately critical to the identification of those portions of the program code where specific, individual rules may be located within a given program.

Program Slicing – Closely allied to data flow analysis, this class of techniques focuses on identifying the specific path of data flow through a program relative to a specific single statement and variable. Widely used in other aspects of reverse engineering, researchers who have used slicing for rule extraction include Huang et al. (1998), Sneed and Erdos (1996), and Ning et al. (1993). Because program slicing can be used to identify input data and the logical/mathematical operations that are performed on that data, program slicing directly addresses the explicit, precise, and operative elements of the EPOID extractable rule definition.

Visualization – Whereas visualization itself is not a rule extraction technique, it is a critical element for program understanding. Visualization can be applied to data structure, program structure, and data flow, with each returning a critical and unique contribution to the total understanding. Researchers who have included visualization as part of a business rule or specification extraction process include Sneed (1998) and Gannod and Chen (1996).

Transformation/Conversion – This class of techniques is typically associated with formal methods and involves the transformation of program code into a higher-level abstracted language. This transformation is accomplished by converting a target code element to an equivalent abstraction in the selected formal language. As program implementation details are purposefully dropped during the transformation/conversion from the target functional language, the resulting abstraction is less cluttered syntactically, making it easier to identify the important semantic elements. Whereas transformation/conversion has not been used solely for the purpose of rule extraction, it has been used in specification/design recovery and code extraction. Researchers who have used transformation/conversion for specification/design recovery and code extraction include Bowen et al. (1993), Gannod and Cheng (1999), Yang et al. (2000), and Villavicencio and Oliveria (2001).

## 2.9 Problems with Existing Rule Extraction Approaches

Based on this literature survey, rule extraction techniques reported in the literature have one or more critical shortcomings that compromise their usefulness or applicability to rule extraction from heterogeneous systems. A discussion of these specific problems follows.

Firstly, there is substantial variation among researchers and practitioners regarding exactly what constitutes an rule. Although this is not a research failure *per se*, it does highlight the fundamental issue that various researchers and practitioners have different end-points, or expectations, regarding the rule analysis and/or extraction process. As a result, it may be difficult, or impossible, for one to use another's specific methodology or associated tool if that methodology or tool embodies different expectations regarding what constitutes an rule. This lack of a clear standard regarding what constitutes an rule makes the development, implementation, and assessment of a general rule analysis and/or extraction process, consistent across languages and platforms, very difficult. The target of the extraction process must be clearly defined, and agreed and accepted prior to initiating a rule extraction project on a heterogeneous, multiple language system.

Secondly, many existing approaches and the related tools are language specific. Frequently, they are focused on unique and language-specific syntax, or on language-specific structures, such as pointers in the C language. Although these approaches were developed to address the specific problems or circumstances presented by a given language, such language specificity may compromise the use of many approaches across different languages in a heterogeneous language environment.

Rule extraction from any existing program requires both the syntactic and semantic understanding of the code. Whereas syntactic analysis of a given language program can be eventually automated (subject to the problems raised above), semantic analysis of that program code requires a knowledgeable expert. Many of the reviewed rule extraction techniques attempt to classify, organize, and present language syntax in such a way as to aid in the human-based semantic extraction. Ultimately, the knowledgeable human must intervene, interpret the organized information, and then perform the actual rule identification and extraction. This yields the third major problem: the individual responsible for rule extraction must be expert both in the domain of the target rules, and in the domains of the various program languages in which these rules have been coded. Such multiple domain experts will be, by their very nature, extremely rare.

Traditionally, this problem has been addressed by the organization of an extraction team of multiple expert individuals that, as a unit, satisfies the multiple domain expertise requirement identified above. However, this management approach results in the fourth problem: the unavoidable inconsistencies of different individuals using different approaches for different languages or environments.

Fifthly, many rule extraction techniques are not mathematically formal or complete. The absence of mathematically formal, or semi-formal, elements in most current extraction techniques ultimately results in an underlying uncertainty regarding the completeness of the technique. An unintended omission of a critical rule or critical case would be certainly embarrassing, probably costly, and in the case of certain critical systems, possibly catastrophic. Most current techniques provide little basis for estimating the completeness of the extraction process and for assessing the possibility that a rule has been overlooked. Therefore, any final statement regarding the success of a given rule extraction exercise can be only a reasoned opinion, instead of a demonstrable and supportable fact.

Finally, few of the rule models from the reverse engineering domain have been applied in the forward engineering domain, and vice versa. In the absence of a sufficiently general rule model, rules that are extracted from legacy code may have to be transformed into a new rule model before those rules can be used the forward engineering of new specifications and code.

# Chapter 3

# A General Formal Framework for Rule Extraction

In this chapter, a critical element necessary for a formal approach to rule extraction from legacy code is presented – a general formal framework applicable to a wide range of legacy languages. Under this rule extraction framework, general mathematical formality is introduced by describing a program in an arbitrary program language as a set of language elements and structures. Using this framework, if rule structures can be adequately specified in terms of that program language, a program can be definitively partitioned into program structures that either are or are not rules. In circumstances where a rule cannot be adequately defined, an alternative, less definitive exclusionary approach is presented. This framework is assessed in relation to two programming languages.

## 3.1 Set-Based Formal Framework

Every programming language consists of a finite set of language elements. For the purpose of this analysis, elements are the atomic units of a language that have a single meaning, function, purpose, or otherwise represent a single value, entity, group, or class. In general, these language elements may include numerical values; variables; mathematical operators; logical operators; assignment operators; language-specific reserved or key words; language-specific punctuation, separators, delimiters, and terminators; and other language-specific commands or tokens necessary for the execution of the program code.

For this analysis, a state-based model of programming is adopted. A state is a function mapping a set of variables to a set of values. Programs are created to instantiate specific states and sequences of states. These states are defined, expanded, modified, selected, and/or sequenced by the programmer to reflect specific knowledge of a given domain. These state manipulations are achieved by the choice of specific program language elements. In a given programming language, two explicit examples of this state model are the type statement and the assignment statement. With the type statement, a specific state variable of a defined type is created. With an assignment statement, a value is bound to a specific variable. Whereas these two examples are direct implementations of the state model of programming, within the context of this state model, all program elements support, either directly or indirectly, the underlying objective of defining, expanding, modifying, selecting, and/or sequencing states.

For any given programming language, the syntactic composition required to create specific instances of the various language elements is explicitly stated or defined. For example, mathematical operators may be represented as a single symbol (+, -, *, or /), and a variable may be defined as a series of not more than 255 letters and numbers starting with a letter. Such instantiation syntax constitutes what most programmers know and practice as "the language." In all languages, these syntactic requirements limit the total number of possible unique instances of these language elements.

For a program written in a given language, a subset of the available language elements is used to create the specific language structures that form that unique program. These structures are constructed from language elements arranged by the programmer in a specific and unique sequence to accomplish an intended task. In the state model of programming, these tasks and the corresponding program structures always relate to the definition, expansion, modification, selection, and/or sequencing of specific states. A basic example of such a structure is the single line of code '$x$ := 1;'. Composed of the language elements of variable, assignment operator, number, and terminator, this structure dictates that, when executed, the then current state will be modified such that the value 1 is bound to the variable $x$. Within a program, multiple language structures can be connected and ordered, and a multi-state state sequence is defined by these ordered language structures. If logical branching is incorporated as a program structure, varying state sequences may result from the same program structure. Thus, a complex structure, composed of multiple program structures, can describe a wide, and possibly infinite, set of state sequences. However, regardless of the final complexity of the structures used and regardless of the potential for an infinite set of state sequences, in all finite programs, the total number of language structures contained in that program is limited and therefore knowable.

If extractable rule forms can be defined in terms of the set of language elements and structures, and all structures in a given program identified, then each identified structure can be assessed as to whether it is, or is not, a rule by whether it matches a previously specified rule structure. If all program structures are identified and can be compared in a two-value manner (yes/no) against the specified rule structures, then the rule analyst can assert, with mathematical certainty, that all rules of specific form(s) have been extracted from the program. If the identified program structures can be assessed only in a three-value manner (yes/no/maybe) against possible rule structures, then the location and magnitude of any uncertainty regarding what may or may not be a rule can be quantified.

These concepts can be expressed symbolically using set builder notation. For any programming language, let E be the set of all elements of that language, and S be the set of all language structures that can be formed from these elements, subject to the syntactic constraints of the language. The set of all rules, R, that can be formed in that language can be defined as:

$$R = \{ x \mid x \in S \wedge f(x, E, S) \} \tag{3.1-1}$$

where the function $f$ is an extractable rule definition function that specifies the properties that a rule must have in terms of the language elements and structures.

In the given language of interest, any program, P, can be defined as a finite subset of all structures S, or $P \subseteq S$. Finally, the set of all extractable rules, $R_E$, contained in program P can be defined as:

$$R_E = \{ z \mid z \in P \wedge z \in R \} \tag{3.1-2}$$

With these equations, the functional requirements of this approach are clear. First, a general rule definition must be developed and expressed as a function in terms of specific language elements and structures, as required by (3.1-1). Second, all structures contained in a given program must be efficiently identified and elicited. If both requirements can be achieved, then the rule extraction process reduces to the intersection of these two sets, as described in (3.1-2), and all rules within a given program can be identified with certainty. This inclusion approach provides a two-value solution to the rule extraction problem in that all structures within a given program either are, or are not, a rule.

If an acceptable extractable rule definition function cannot be achieved, then an alternative or exclusion approach can be formulated, using this same general framework. Using the previously defined sets P, a given program, and $R_E$, all extractable rules within that program, the structures in that program that are not rules can be described as the relative complement of these two sets, or:

$$P - R_E = \{ z \mid z \in P \wedge z \notin R \} \qquad (3.1\text{-}3)$$

Using this alternative approach, two requirements must be satisfied. First, and as before, all structures within the program must be efficiently identified and elicited. Second, those structures that are not rules, i.e., $z \notin R$ in (3.1-3), must be identified. In practice, programmers and rule analysts will probably know that certain language structures cannot be rules. A comment is one obvious example common to all languages. Other specific cases of non-rules will depend on the language, and the presumed attributes of rules. Thus, in the absence of a rigorous definition of what is a rule, this alternative, or exclusion, approach provides a three-value solution to the rule extraction problem: no, maybe, and yes. With this approach, some structures will be tagged with certainty and excluded from consideration as non-rules; the classification of the remaining structures remains uncertain, as some will, and some will not, contain rules.

Both approaches, given in (3.1-2) and (3.1-3), have their place in practice, each with their associated advantages and disadvantages. If an explicit, precise, and acceptable definition of an extractable rule can be developed, such as the extractable rule definition function $f$ in (3.1-1), then the inclusive approach of (3.1-2) can be implemented and all rules can be identified with two-value certainty. If such an explicit, precise, and acceptable rule definition is unavailable, then the alternative exclusionary approach (3.1-3) can be implemented. Although incomplete, this approach limits and identifies portions of code where rules may exist subject to the certainty and specificity of the criteria used to exclude non-rule structures.

## 3.2 Evaluation of the Framework – C Language

To assess the application of this framework, the inclusion and exclusion rule-extraction approaches presented in (3.1-2) and (3.1-3), respectively, were evaluated. For each of these analyses, a simple rule-based program written in the C language was developed and is presented in Listing 3.2-1. In this program, a simple user input, 1 or 0, is accepted from the keyboard, and then a reply value of yes or no is assigned and displayed based on a simple rule using the user input. By design, the code is very simple to provide the basis for clear and unambiguous examination.

```
1       #include <stdio.h>
2       #include <string.h>

3       int main(void)
4       {
5               char reply_yes[10] = "Yes";
6               char reply_no[10] = "No";
7               char reply[10];
8               char user_input ;

9               // This is a demo program
10              printf("Enter 1 or 0 : ");
11              user_input = getc(stdin);
12              if(user_input == '1')
13              {
14                      strcpy(reply, reply_yes);
15              }
16              else
17              {
18                      strcpy(reply, reply_no);
19              };

20              printf("Answer: %s \n", reply);

21              return(0);
22      }
```

Listing 3.2-1: A Simple Rule-Based Program in the C Language

This example code contains twelve structures. These structures are: two library reference structures (lines 1 and 2); a single program block structure (lines 3, 4, 21, and 22); four type definition structures (lines 5 through 8); one comment structure (line 9); one input capture structure (line 11); two output display structures (lines 10 and 20); and one logical if structure (lines 12 through 19). Note that the one logical if structure is composed of multiple, smaller structures.

For the implementation of (3.1-2), an extractable rule definition function is required. One element of the definition of an extractable business rule presented in Section 2.4 is that an extractable business rule must be logically or mathematically operative. Using this rule attribute and considering the requirement that a rule be a structure, the extractable rule definition function for this analysis will be whether a given structure is logically or mathematically operative. Any structure that contains a logical or mathematical operator will be declared a rule; any structure that does not contains a logical or mathematical operator will be declared a non-rule.

On applying this extractable rule definition function to the previously enumerated list of structures in the demonstration code, only one structure is found to fit the criteria of being mathematically or logically operative – the one logical if structure located at lines 12 through 19. This rule structure is highlighted in Listing 3.2-2. No other mathematical or logical operators exist. The other structures either contain no operators, or contain only assignment operators, e.g., the four type definition structures at lines 5 through 8. Subject to the continued acceptance of the extractable rule definition function used in this example, one can be mathematically certain, using the set requirements specified in (3.1-2), that all rules contained in the target program have been identified.

For the implementation of the alternative exclusionary approach of (3.1-3), no extractable rule definition function is required. Instead, only a basic understanding of both extractable rules and the C language is needed. For this analysis, it is assumed with certainty that comments, library references, output/display structures, and block/control statements cannot contain rules. This allows the elimination of lines 1, 2, 3, 4, 9, 10, 20, 21, and 22 as non-rules. It is further assumed with certainty that a rule must be operative (i.e., it must do something); this allows the elimination of lines 7 and 8 as they contain no operators of any kind. These eliminated structures are struck through in Listing 3.2-3. Thus, four structures are left that have not been eliminated, as presented in Listing 3.2-3: lines 5, 6, 11, and 12 through 19. Subject to a continued acceptance of the criteria used to eliminate the non-rule structures, one can be certain

```
1       #include <stdio.h>
2       #include <string.h>

3       int main(void)
4       {
5               char reply_yes[10] = "Yes";
6               char reply_no[10] = "No";
7               char reply[10];
8               char user_input ;

9               // This is a demo program
10              printf("Enter 1 or 0 : ");
11              user_input = getc(stdin);

12              if(user_input == '1')
13              {
14                      strcpy(reply, reply_yes);
15              }
16              else
17              {
18                      strcpy(reply, reply_no);
19              };

20              printf("Answer: %s \n", reply);

21              return(0);
22      }
```

Note: The extracted rules are shown in **bold**.

Listing 3.2-2: Rule Extraction from the C Language Program Using the Inclusion Approach of (3.1-2)

that the eliminated structures contain no rules. However, uncertainty remains with respect to whether the remaining code does, or does not, contain any rules, and if so, what those rules are.

The value of the exclusionary approach comes with iterative application. With one application using a relatively general definition of what is not a rule, fifty percent of the code was eliminated. From a practical perspective, this dramatically reduced the effort necessary by a rule analyst in the further analysis of the target code. For example, if on further inspection and reflection, it is determined that all type definition statements cannot be rules, then lines 5 and 6 can be eliminated. With this elimination, only two structures remain that can be rules – the user input structure of line 11 and the logical if structure of lines 12 through 19. Thus, with two iterations applying the exclusionary

```
~~#include <stdio.h>~~
~~#include <string.h>~~

~~int main(void)~~
~~{~~
        char reply_yes[10] = "Yes";
        char reply_no[10] = "No";
        ~~char reply[10];~~
        ~~char user_input ;~~

        ~~// This is a demo program~~
        ~~printf("Enter 1 or 0 : ");~~
        user_input = getc(stdin);

        if(user_input == '1')
        {
                strcpy(reply, reply_yes);
        }
        else
        {
                strcpy(reply, reply_no);
        };

        ~~printf("Answer: %s \n", reply);~~

        ~~return(0);~~
~~}~~
```

Note: The eliminated structures are shown in ~~strikethrough~~.

Listing 3.2-3: Rule Extraction from the C Language Program
Using the Exclusion Approach of (3.1-3)

approach of (3.1-3), the original program containing twelve structures has been reduced to two structures that may, or may not, contain rules. Therefore, in the absence of a formal definition of an extractable rule, as required for the application of the inclusion approach of (3.1-2), the exclusion approach of (3.1-3) allows an orderly approach to significantly reducing the size of the code that must be assessed for rules using other means.

## 3.3 Evaluation of the Framework – Wide Spectrum Language

To assess the issue of language specificity under this general framework, a second program analysis and rule extraction was conducted. This second assessment was implemented based on the translation and transformation of an original source program into an equivalent Wide Spectrum Language (WSL) program. Rule analysis and extraction was then performed on the equivalent WSL program.

With regard to rule extraction from heterogeneous systems, a WSL-based approach has numerous potential advantages. Firstly, using provable, mathematically sound transformations, programs in a variety of languages can be converted into WSL, thereby allowing its use with potentially any source language. Secondly, extraction methodologies for the WSL code could be developed and applied to the transformed programs with the certain knowledge that the underlying logical or mathematical objectives of those methodologies would be uniformly applied regardless of the original system language or paradigm. Thirdly, performing analyses in a single language, WSL, will allow the consistent execution of code analysis or rule extraction strategy regardless of the initial program language. Fourthly, different programs, written in different languages, different styles, and with different levels of extraneous code, e.g., error handling code, could be consistently abstracted using WSL. Finally, rules derived from different original source programs can be expressed easily in a common, consistent form.

As before, both the inclusion and exclusion rule-extraction approaches presented in (3.1-2) and (3.1-3), respectively, were evaluated. For these evaluations, a second rule-based program, written in the C language, was developed and is presented in Listing 3.3-1. This program accepts the user input of two numerical values from the keyboard, mathematically manipulates these two input values to determine a test value, then assigns and displays a reply value based on the comparison of this test value against a specified criterion. Although more sophisticated than the previous case presented in Listing 3.2-1, this code is very simple to provide the basis for clear and unambiguous examination.

This C code contains 22 structures. These structures are: three library reference structures (lines 1, 2, and 3); a global constant definition (line 4); a single program block structure (lines 5, 6, 31, and 32); five type definition structures (lines 7 through 11); one comment structure (line 12); two input capture structures (lines 14 and 16); three output display structures (lines 13, 15, and 30); five sequential mathematical assignments (lines 17 through 21); and one logical if structure (lines 22 through 29).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CRITERION 20

int main(void)
{
        char reply_yes[10] = "Yes";
        char reply_no[10] = "No";
        char reply[10];
        char buffer[80];
        double input1, input2, test ;

        // This is a demo program

        printf("Enter first input: ");
        input1 = atof(gets(buffer));

        printf("Enter second input: ");
        input2 = atof(gets(buffer));

        test = input1 + input2 ;
        test = test + 2 ;
        test = test * 2 ;
        test = test + input2 ;
        test = test + 2 ;

        if(test >= CRITERION)
        {
                strcpy(reply, reply_yes) ;
        }
        else
        {
                strcpy(reply, reply_no) ;
        };

        printf("The answer: %s \n", reply);

        return(0);
}
```

Listing 3.3-1: A Second Simple Rule-Based Program
in the C Language

This rule-based program was translated into WSL. The resulting WSL program is presented in Listing 3.3-2. This equivalent WSL program contains only seven structures, as compared to the 22 structures in the original C program. These structures are: one variable declaration structure (line 1, terminating on line 10); two output display structures (lines 2 and 4); two input capture structures (lines 3 and 5); one mathematical operation reflecting the transformation of the multi-line local procedure in the C code into a semantically equivalent single statement (line 6); and one logical if structure (lines 7 through 9) that incorporates related output display. This dramatic reduction in the total number of structures highlights one of the major potential advantages of this transformation approach to rule extraction and identification, namely, that using WSL allows the substantial simplification or abstraction of a target program code. Whereas not all circumstances will result in the magnitude of code reduction observed here, the elimination of code that is superfluous to the logical functioning of the program, either through translation into the streamlined syntax of WSL or through the transformation of source code into semantically equivalent WSL code, greatly aids in the comprehension of the program and the corresponding identification of program components of interest.

```
1     VAR < input1 := 0.0, input2 := 0.0, test := 0.0 >:
2     PRINFLUSH("Enter first input: ");
3     input1 := @String_To_Num(@Read_Line(Standard_Input_Port));
4     PRINFLUSH("Enter second input: ");
5     input2 := @String_To_Num(@Read_Line(Standard_Input_Port));
6     test := input1 * 2 + input2 * 3 + 6 ;
7     IF test >= 20
8           THEN PRINT("Answer: Yes")
9           ELSE PRINT("Answer: No") FI;
10    ENDVAR
```

Listing 3.3-2: The Equivalent Program in WSL

That an extractable business rule must be logically or mathematically operative, as presented in the definition of an extractable business rule in Section 2.4, was used again as the basis for the extractable rule definition function required for the implementation of (3.1-2). Upon applying this extractable rule definition function to the previously enumerated list of structures in the demonstration code, only two structures are found to fit the criterion of being mathematically or logically operative – the mathematical assignment of the test value (line 6) and the one logical if structure (lines 7 through 9). These rule structures are highlighted in Listing 3.3-3. No other mathematical or logical operators exist. The other structures either contain no operators, or contain only assignment operators. For example, since the function calls in lines 3 and 5 are for string input and manipulation only, and are not related to the mathematical or logical manipulation of these strings, these structures are determined not to contain any rules. Therefore, subject to the continued acceptance of the extractable rule definition function used in this example, one can be mathematically certain, using the set requirements specified in (3.1-2), that all rules contained in the example program have been identified.

```
1     VAR < input1 := 0.0, input2 := 0.0, test := 0.0 >:
2     PRINFLUSH("Enter first input: ");
3     input1 := @String_To_Num(@Read_Line(Standard_Input_Port));
4     PRINFLUSH("Enter second input: ");
5     input2 := @String_To_Num(@Read_Line(Standard_Input_Port));
6     test := input1 * 2 + input2 * 3 + 6 ;
7     IF test >= 20
8           THEN PRINT("Answer: Yes")
9           ELSE PRINT("Answer: No") FI;
10    ENDVAR
```

Note: The extracted rules are shown in **bold**.

Listing 3.3-3: Rule Extraction from the WSL Program Using the Inclusion Approach of (3.1-2)

For the implementation of the alternative exclusion approach of (3.1-3), no extractable rule definition function was required, only a basic understanding of both extractable rules and the WSL language is needed. For this analysis, it is assumed with certainty that variable declarations, output/display structures, and simple input capture structures cannot contain rules. This allows the elimination of lines 1, 2, 3, 4, 5, and 10 as non-rules. These eliminated structures are struck through in Listing 3.3-4. Thus, only two structures remain that have not been eliminated – line 6 and lines 7 through 9. Subject to a continued acceptance of the criteria used to eliminate the non-rule structures, one can be certain that the eliminated structures contain no rules. However, given the approach and the associated lack of an adequate rule definition, uncertainty must remain with respect to whether the remaining code does or does not contain any rules.

```
~~VAR < input1 := 0.0, input2 := 0.0, test := 0.0 >;~~
~~PRINFLUSH("Enter first input: ");~~
~~input1 := @String_To_Num(@Read_Line(Standard_Input_Port));~~
~~PRINFLUSH("Enter second input: ");~~
~~input2 := @String_To_Num(@Read_Line(Standard_Input_Port));~~
test := input1 * 2 + input2 * 3 + 6 ;
IF test >= 20
        THEN PRINT("Answer: Yes")
        ELSE PRINT("Answer: No") FI;
~~ENDVAR~~
```

Note: The eliminated structures are shown in ~~strikethrough~~.

Listing 3.3-4: Rule Extraction from the WSL Program Using the Exclusion Approach of (3.1-3)

## 3.4 Observations

Although purposefully limited in scope, these studies highlight the requirements, similarities, advantages, and limitations of the application of this general framework and the two related approaches to rule extraction. Both the inclusion and exclusion approaches require the identification of all structures contained in the target code. Complex structures that may be composed of multiple structures or structures within structures must be resolved and decomposed into relatively simple structures that can be analyzed against a rule definition or rule model, in this case, the extractable rule definition function of (3.1-3). Given the very limited size of these demonstration programs, identification of all structures at the appropriate level of detail was a simple, straightforward matter. However, if the original target program is lengthy, or if the original program language is either poorly documented or little known to the rule analyst, or both, then the identification of all program structures in the original program can be a formidable mechanical and logical task. Given lengthy source code, the possible number of structures will increase exponentially with the number of lines of code, making it difficult to efficiently identify all structures within a program. As demonstrated in the WSL examples, program conversion potentially allows the substantial simplification or abstraction of the target program code, thereby reducing the potential magnitude of the problem. Nonetheless, the 'state explosion' associated with lengthy code represents a potentially significant scalability issue regarding the application of this formal framework to real world rule extraction problems, regardless of the language on which the extraction activities are based.

The two approaches to rule extraction differ dramatically with regard to the necessity for and application of a suitable rule definition. The inclusion approach, based on (3.1-2), requires an explicit, precise rule definition. If such a rule model or rule definition can be developed and applied to the target program language, then all rules can be extracted from the target program code with absolute, mathematical certainty. Conversely, the exclusion approach, based on (3.1-3), requires no *a priori* definition of what constitutes a rule, only an ordered understanding of what is not a rule. Consistent with that compromise, the exclusion approach affords no mathematical certainty whether the extracted structures contain only rules or other non-rule structures. Therefore, all subsequent research presented in this thesis will use the inclusion approach.

Therefore, to achieve the stated goal of developing a suitable formal methodology for rule extraction to legacy code, two obstacles must be overcome. Firstly, a flexible but formal model of a business/knowledge rule must be developed that can be applied to a diversity of legacy code. Secondly, a formal approach regarding the potential scalability issues in real-world code must be devised. A formal model of a business/knowledge rule is presented in Chapter 4 and an algebra describing the application of that rule model is presented in Chapters 5 and 6. Potential scalability issues are addressed in Chapter 7 using the visual formalisms of statecharts.

# Chapter 4

# Temporal Logic and Rules

One of the critical impediments identified in Chapter 2 is the lack of a general rule definition that can be applied uniformly and consistently in the analysis of legacy code and in the execution of the corresponding rule extraction. In this chapter, a formal, general model of a rule is developed, general in that it can be adapted to the variety of languages and programming paradigms that might be encountered in different legacy code applications.

## 4.1 A State-Based Model of a Rule

A state is a function mapping a set of variables to a set of values. As most legacy code languages can be analyzed readily in terms of state variables and the operations that change the values bound to those variables, it is convenient to conceptualize most legacy code programs, and the rules contained therein, in terms of states and state transitions. Therefore, a simple rule can be described informally as a state transition from an initial state to a final state occurring only when a specified well-formed conditional is satisfied. Using this descriptive model of a rule, consider the following three-tuple:

$$< \Sigma, \delta, C > \qquad (4.1\text{-}1)$$

where:

$\Sigma =$ set of valid states, such that $s_{initial}, s_{final} \in \Sigma$,

$\delta =$ transition relationship, relating $s_{initial}$ to $s_{final}$, and

$C =$ properly formed condition that must be satisfied for the state transition relationship described by $\delta$ to occur.

Several general points merit note regarding this general descriptive model. Firstly, the state variables used in $\Sigma$ can represent any component, object, or property of interest. Secondly, no limitations are placed on the nature of the transition relationship $\delta$. This transition is expressed as a relation and not a function to allow for non-deterministic rules. Therefore, for a given rule, multiple alternative final states may result for a single initial state. Thirdly, the use of the state descriptors *initial* and *final* are relative to a single rule, where each transition described by a rule will have an initial and final state. Within this context, more sophisticated rules and rule-based programs can be formed by defining multiple rules and linking those rules together, such that the $s_{final}$ resulting from one rule may then be used as the $s_{initial}$ for a subsequent rule.

A critical issue in the refinement of the basic rule model presented in (4.1-1) is the nature of the condition, $C$, that must be satisfied for the transition from $s_{initial}$ to $s_{final}$, as described by the transition relationship $\delta$, to proceed. As described below, the form of this conditional is a critical factor in determining whether a given structure constitutes a rule. Consider the following simple assignment:

$$x := 1 \qquad (4.1\text{-}2)$$

Using the rule model presented in (4.1-1), this assignment is not a rule, as it does not include a condition. To include a condition, this simple assignment can be rewritten as the following if-then conditional:

$$\text{if } \mathit{true} \text{ then } x := 1 \qquad (4.1\text{-}3)$$

In both cases, $x$ will always evaluate to 1. Although the second form (4.1-3) includes a condition (i.e., 'if *true*'), the form of the conditional dictates that $x$ always evaluates to 1. To that end, no state knowledge is required to evaluate $x$. In either form, the variable $x$ always will be assigned a value of 1. Thus, both statements are unconditionally true. Borrowing from the concepts associated with the programming language PROLOG, statements that are always unconditionally true are facts (Bratko, 2001), and not rules.

Formally, this argument can be made using propositional logic. Let the atomic proposition $Q$ represent (4.1-2) and let the conditional presented in (4.1-3) be described using implication as $\mathit{true} \supset Q$. From propositional logic, $\mathit{true} \supset Q \equiv Q$. Therefore, (4.1-2) and (4.1-3) are logically equivalent. Based on this proven logical equivalence, because (4.1-2) is not a rule by definition, (4.1-3) is not a rule by extension.

Now consider the following modified if-then conditional:

if $y = \mathit{true}$ then $x := 1$ (4.1-4)

In this case, the evaluation of $x$ to 1 depends on the state of $y$. The state of $x$ is no longer certain. This conditional is formed such that the future value of $x$ is dependent on the value of y, and not on the invariant form of the condition as in the previous example. Extending the previous analysis using propositional logic and letting the atomic proposition $P$ represent $y = \mathit{true}$, (4.1-4) can be represented using implication as $P \supset Q$. Without additional knowledge regarding the current state of $P$ and the application of an inference rule, no further simplification of (4.1-4) can be made, supporting the conclusion that (4.1-4) if fundamentally different than (4.1-2) or (4.1-3).

Therefore, for the purposes of defining a rule, the properly formed condition criterion relates to the mathematical form of the conditional relative to expressing the conditions in terms of a state and the associated state variables. Consider the following rule-based, two-line program describing a simple two-variable state space and incorporating the conditional presented in (4.1-4):

$y := \mathit{true}$ (4.1-5a)
if $y = \mathit{true}$ then $x := 1$ (4.1-5b)

Consisting of an assignment and an if-then rule, this rule-based program will always evaluate $x$ to 1. However, this certain evaluation is based on the limited expression of knowledge within the program, i.e., that $y$ is specified in the program to be *true*, and not based on the mathematical or logical form of the rule conditional. Whereas one may consider this a trivial rule, it is potentially a properly formed rule in that the final assignment of $x$ is not constrained by the mathematical or logical form of the conditional controlling the assignment of $x$. If the knowledge about the state space being modeled were expanded such that $y$ might vary, then $x$ could vary also. In this case, this rule-based program is limited only by the knowledge of the state space relative to the interaction of the state variables, and not by the fundamental form used in the expression of the rule.

Therefore, the general concept presented in (4.1-1) is modified such that a business or knowledge rule is formally defined by the three tuple:

$$< \Sigma, \delta, C > \qquad (4.1\text{-}6)$$

where:

| | |
|---|---|
| $\Sigma =$ | set of valid states, such that $s_{initial}, s_{final} \in \Sigma$, |
| $\delta =$ | transition relationship, relating $s_{initial}$ to $s_{final}$, and |
| $C =$ | condition that must be satisfied for the state transition relationship described by $\delta$ to occur, and must be properly formed relative to the state such that $C(s) = true$ for $s \in \Sigma$. |

Whereas this research typically discusses the condition $C$ as a logical conditional of an imperative/procedural language, no limitation is imposed with regard to how this condition may be implemented. For example, $C$ could be dependent on a event, including the receipt of a message, such that the initiation, ongoing execution, or completion of such an event would evaluate $C$ to *true*. Similarly, $C$ could be expressed in terms of a group of concurrent actions, or the truth of the conditional is based on some set of temporal actions set of past, current, or future behaviors. The general rule definition presented in (4.1-6) has been constructed to permit the analysis of rules in a wide range of specifications and program codes and to support various forms of rule implementation within those specifications and program codes.

To further focus on the state outcome of this state-based model of a rule, the concept of a rule state is introduced. A rule state is the state (or state sequence, as will be discussed later in this chapter) that results from the implementation of a rule, that is, the state that results from the transition relationship $\delta$ of a properly formed rule, as described in (4.1-6). The rule state of the general rule described in (4.1-6) is $s_{final}$.

Refining the requirements associated with the rule condition, explicitly incorporating the rule state concept, and generalizing to eliminate the use of *initial* and *final*, the three-tuple formal definition presented in (4.1-6) can be expressed in an alternative form:

$$C(s) = \mathit{true} \wedge s' = \delta(s) \text{ for } s, s' \in \Sigma \tag{4.1-7}$$

where:

$s' =$ rule state, resulting from the transition specified by $\delta$,

$\delta =$ transition relationship, relating $s$ to $s'$, and

$C(s) =$ rule condition to be satisfied, expressed in terms of the state $s$.

## 4.2 A Very Basic Temporal View of Rules

Implicit in the description presented in (4.1-7) is a temporal ordering of states. As previously defined, the outcome of the rule is the rule state. As the transition to rule state $s'$ is conditioned on the environment being in state $s$, thereby satisfying the condition specified by $C$, and as no environment can be in two states at the same time, the rule state $s'$ must occur after state $s$. These general temporal properties of (4.1-7) can be described in the following simplified form:

$$\mathrm{C}(s) \wedge s'_{future} \text{ for } s, s' \in \Sigma \tag{4.2-1}$$

where:

$s' =$ rule state described by the transition relationship $\delta$ (relating $s$ to $s'$) and occurring in the future relative to $s$, and

$C(s) =$ rule condition to be satisfied, expressed in terms of the state $s$.

As a conjunctive structure, (4.2-1) is true only if both elements of the conjunction hold – if the condition expressed in terms of a state $s$ is satisfied and if the state is moved in the future into some state $s'$ as defined by the transition relationship $\delta$. Using the model presented in (4.2-1), rules can be described as a conjunctive structure that specifies both a state that satisfies the rule condition and a future rule state. Extending this description, a rule defines a temporal relationship between states. As these temporal aspects are critical to a formal model of a rule, the following section describes temporal logic and Interval Temporal Logic, as tools for expressing rules and reasoning with rules.

## 4.3 Temporal Logic and Interval Temporal Logic

Temporal logic is a powerful tool for the formal reasoning about time and the behavior of dynamic systems without requiring the introduction or use of explicit time variables. Using temporal logic, time concepts relative to a sequence of states can be expressed using different temporal operators, including always ($\Box$), sometimes ($\Diamond$), and next ($\circ$). A comprehensive review of the development and implementation of temporal logic is presented in Manna and Pnueli (1992, 1995).

This research uses Interval Temporal Logic (ITL), a flexible notation for propositional and first-order reasoning about periods of time (i.e., intervals) in hardware and software. ITL can be used to reason about both sequential and parallel composition, and includes powerful and extensible specification and proof techniques for reasoning about critical properties such as safety and liveness (Moszkowski, 1996). As Cau and Zedan (1997) have demonstrated that most imperative programming constructs can be represented as formulas in ITL, it is well suited for the analysis of legacy code as well as the analysis and specification of other non-legacy constructs. Detailed descriptions of ITL can be found in Moszkowski (1986, 1994, 2000, 2003) and the ITL homepage (STRL, 2006).

Fundamental to ITL is the concept of the interval – a (in)finite sequence of states that describes the behavior of a program or specification over time. Using the states $s$ in $\Sigma$, intervals of time, i.e., sequences of states, can be constructed from $\Sigma^+$, the set of all non-empty sequences of states. Such an interval of states is represented by $\sigma$ and the length of that interval is one less than the number of states in that interval. Under this definition, a single state is a valid interval, and the length of a single state interval is zero. As intervals can themselves be composed of intervals, ITL is highly adaptable to both abstraction and refinement, as intervals can be either aggregated or partitioned, depending on the specific circumstances.

Intervals in ITL are described by expressions and formulas. The syntax of ITL is presented in Table 4.3-1, where $z$ is an integer value, $a$ is a static variable (i.e., a variable that does not change within an interval), $A$ is a state variable (i.e., a variable that can change within an interval), $v$ is a static or state variable, $g$ is a function symbol, and $p$ is a predicate symbol. Formulas may or may not include temporal operators. A state formula, in this research denoted by $w$, is a formula that contains no temporal operators. The verity of a state formula for a given interval, that is, a sequence of states, is assessed based by the first state in that interval.

Table 4.3-1 Syntax of ITL

*Expressions*

$$exp ::= \ z \mid a \mid A \mid g(exp_1, \ldots, exp_n) \mid \imath a\colon f$$

*Formulae*

$$f ::= \ p(exp_1, \ldots, exp_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \cdot f \mid \mathsf{skip} \mid f_1 \,;\, f_2 \mid f^*$$

Source: ITL home page at http://www.cse.dmu.ac.uk/~cau/itlhomepage/

The formal semantics of ITL is listed in Table 4.3-2. The informal semantics of some of the ITL constructs key to the analysis of rules and the research presented herein include:

- skip – unit interval
- $f_1$ ; $f_2$ holds over an interval if that interval can be decomposed (or "chopped") into a two intervals, a prefix and suffix interval, such that $f_1$ holds over the prefix interval and $f_2$ holds over the suffix. If the interval is infinite, then $f_1$ must hold for that interval. The ; operator is read as "chop."
- $f^*$ holds if the interval is decomposable (i.e., chopable) into a finite number of intervals such that $f$ holds for each of them. If the interval is infinite, it must be decomposable into an infinite number of finite intervals for which $f$ holds. The $^*$ operator is read as "chop-star."

The following are some simple ITL formulas and their informal meanings.

- $I = 1$ holds for a interval if the value of $I$ in the initial state of that interval is 1, regardless of the value of $I$ in any subsequent states that may compose that interval. This formula can hold on a single state interval.
- $I = 2 \wedge$ skip holds for a two-state interval if the value of $I$ in the initial state of that interval is 2.

Table 4.3-2 Semantics of ITL

$\mathcal{E}_\sigma[\![v]\!] = \sigma_0(v)$

$\mathcal{E}_\sigma[\![g(exp_1, \ldots, exp_n)]\!] = \hat{g}(\mathcal{E}_\sigma[\![exp_1]\!], \ldots, \mathcal{E}_\sigma[\![exp_n]\!])$

$\mathcal{E}_\sigma[\![\imath a\colon f]\!] = \begin{cases} \chi(u) & \text{if } u \neq \{\} \\ \chi(\mathrm{Val}_a) & \text{otherwise} \end{cases}$

where $u = \{\sigma'(a) \mid \sigma \sim_a \sigma' \wedge \mathcal{M}_{\sigma'}[\![f]\!] = \mathrm{tt}\}$

$\mathcal{M}_\sigma[\![p(exp_1, \ldots, exp_n)]\!] = \mathrm{tt}$ iff $\hat{p}(\mathcal{E}_\sigma[\![exp_1]\!], \ldots, \mathcal{E}_\sigma[\![exp_n]\!])$

$\mathcal{M}_\sigma[\![\neg f]\!] = \mathrm{tt}$ iff $\mathcal{M}_\sigma[\![f]\!] = \mathrm{ff}$

$\mathcal{M}_\sigma[\![f_1 \wedge f_2]\!] = \mathrm{tt}$ iff $\mathcal{M}_\sigma[\![f_1]\!] = \mathrm{tt}$ and $\mathcal{M}_\sigma[\![f_2]\!] = \mathrm{tt}$

$\mathcal{M}_\sigma[\![\forall v \cdot f]\!] = \mathrm{tt}$ iff for all $\sigma'$ s.t. $\sigma \sim_v \sigma', \mathcal{M}_{\sigma'}[\![f]\!] = \mathrm{tt}$

$\mathcal{M}_\sigma[\![\mathsf{skip}]\!] = \mathrm{tt}$ iff $|\sigma| = 1$

$\mathcal{M}_\sigma[\![f_1 \,;\, f_2]\!] = \mathrm{tt}$ iff

```
(exists a $k$, s.t. $\mathcal{M}_{\sigma_0 \ldots \sigma_k}[\![f_1]\!] = \mathrm{tt}$ and
    (($\sigma$ is infinite and $\mathcal{M}_{\sigma_k \ldots}[\![f_2]\!] = \mathrm{tt}$) or
    ($\sigma$ is finite and $k \leq |\sigma|$ and $\mathcal{M}_{\sigma_k \ldots \sigma_{|\sigma|}}[\![f_2]\!] = \mathrm{tt}$)))
or ($\sigma$ is infinite and $\mathcal{M}_\sigma[\![f_1]\!]$)
```

$\mathcal{M}_\sigma[\![f^*]\!] = \mathrm{tt}$ iff

```
if $\sigma$ is infinite then
    (exist $l_0, \ldots, l_n$ s.t. $l_0 = 0$ and
        $\mathcal{M}_{\sigma_{l_n} \ldots}[\![f]\!] = \mathrm{tt}$ and
        for all $0 \leq i < n$, $l_i < l_{i+1}$ and $\mathcal{M}_{\sigma_{l_i} \ldots \sigma_{l_{i+1}}}[\![f]\!] = \mathrm{tt}$)
    or
    (exist an infinite number of $l_i$ s.t. $l_0 = 0$ and
        for all $0 \leq i, l_i < l_{i+1}$ and $\mathcal{M}_{\sigma_{l_i} \ldots \sigma_{l_{i+1}}}[\![f]\!] = \mathrm{tt}$)
else
    (exist $l_0, \ldots, l_n$ s.t. $l_0 = 0$ and $l_n = |\sigma|$ and
        for all $0 \leq i < n, l_i < l_{i+1}$ and $\mathcal{M}_{\sigma_{l_i} \ldots \sigma_{l_{i+1}}}[\![f]\!] = \mathrm{tt}$)
```

Source: ITL home page at http //www cse dmu ac uk/~cau/itlhomepage/

- $\circ I = 3$ holds for interval if the value of $I$ in the second state of that interval is 3. Given that the ITL next operator $\circ$ is defined as "skip ;", this formula is equivalent to the formula skip ; $I = 3$ .
- $I = 4$ ; $I = 5$ holds for interval if the value of $I$ in the initial state is 4 and in some later state, but not necessarily the second or next interval, the value of $I$ is 5.

Some frequently used non-temporal derived constructs, temporal derived constructs, concrete derived constructs, and derived constructs related to expressions are presented in Tables 4.3-3 through 4.3-6, respectively.

Table 4.3-3 Frequently used non-temporal derived constructs

| | | | |
|---|---|---|---|
| true | $\widehat{=}$ | $0 = 0$ | true value |
| false | $\widehat{=}$ | $\neg$true | false value |
| $f_1 \vee f_2$ | $\widehat{=}$ | $\neg(\neg f_1 \wedge \neg f_2)$ | or |
| $f_1 \supset f_2$ | $\widehat{=}$ | $\neg f_1 \vee f_2$ | implies |
| $f_1 \equiv f_2$ | $\widehat{=}$ | $(f_1 \supset f_2) \wedge (f_2 \supset f_1)$ | equivalent |
| $\exists v \bullet f$ | $\widehat{=}$ | $\neg \forall v \bullet \neg f$ | exists |

Note: From the ITL home page at http://www.cse.dmu.ac.uk/~cau/itlhomepage/

Table 4.3-4 Frequently used temporal derived constructs

| | | | |
|---|---|---|---|
| $\bigcirc f$ | $\widehat{=}$ | skip ; $f$ | next |
| more | $\widehat{=}$ | $\bigcirc$true | non-empty interval |
| empty | $\widehat{=}$ | $\neg$more | empty interval |
| inf | $\widehat{=}$ | true ; false | infinite interval |
| isinf $(f)$ | $\widehat{=}$ | inf $\wedge f$ | is infinite |
| finite | $\widehat{=}$ | $\neg$inf | finite interval |
| isfin $(f)$ | $\widehat{=}$ | finite $\wedge f$ | is finite |
| fmore | $\widehat{=}$ | more $\wedge$ finite | non-empty finite interval |
| $\Diamond f$ | $\widehat{=}$ | finite ; $f$ | sometimes |
| $\Box f$ | $\widehat{=}$ | $\neg \Diamond \neg f$ | always |
| $\textcircled{w} f$ | $\widehat{=}$ | $\neg \bigcirc \neg f$ | weak next |
| $\Diamond_i f$ | $\widehat{=}$ | $f$ ; true | some initial subinterval |
| $\Box_i f$ | $\widehat{=}$ | $\neg(\Diamond_i \neg f)$ | all initial subintervals |
| $\Diamond_a f$ | $\widehat{=}$ | finite ; $f$ ; true | some subinterval |
| $\Box_a f$ | $\widehat{=}$ | $\neg(\Diamond_a \neg f)$ | all subintervals |

Source: ITL home page at http://www.cse.dmu.ac.uk/~cau/itlhomepage/

Table 4.3-5 Frequently used concrete derived constructs

| | | | |
|---|---|---|---|
| if $f_0$ then $f_1$ else $f_2$ | $\hat{=}$ | $(f_0 \wedge f_1) \vee (\neg f_0 \wedge f_2)$ | if then else |
| if $f_0$ then $f_1$ | $\hat{=}$ | if $f_0$ then $f_1$ else empty | if then |
| fin $f$ | $\hat{=}$ | $\Box$(empty $\supset f$) | final state |
| sfin $f$ | $\hat{=}$ | $\neg$(fin $(\neg f)$) | strong final state |
| halt $f$ | $\hat{=}$ | $\Box$(empty $\equiv f$) | terminate interval when |
| shalt $f$ | $\hat{=}$ | $\neg$(halt $(\neg f)$) | strong terminate interval when |
| keep $f$ | $\hat{=}$ | $\boxed{a}$(skip $\supset f$) | all unit subintervals |
| keepnow $f$ | $\hat{=}$ | $\Diamond_i$(skip $\wedge f$) | initial unit subinterval |
| $f^{\omega}$ | $\hat{=}$ | isinf (isfin $(f)^*$) | infinite chopstar |
| fstar $(f)$ | $\hat{=}$ | isfin (isfin $(f)^*$) $\vee$ isfin (isfin $(f)^*$) ; isinf $(f)$ | finite chopstar |
| while $f_0$ do $f_1$ | $\hat{=}$ | $(f_0 \wedge f_1)^* \wedge$ fin $\neg f_0$ | while loop |
| repeat $f_0$ until $f_1$ | $\hat{=}$ | $f_0$ ; (while $\neg f_1$ do $f_0$) | repeat loop |

Source: ITL home page at http //www.cse.dmu.ac.uk/~cau/itlhomepage/

Table 4.3-6 Frequently used derived constructs related to expressions

| | | | |
|---|---|---|---|
| $\bigcirc exp$ | $\hat{=}$ | $\imath a$: $\bigcirc(exp = a)$ | next value |
| fin $exp$ | $\hat{=}$ | $\imath a$: fin $(exp = a)$ | end value |
| $A := exp$ | $\hat{=}$ | $\bigcirc A = exp$ | assignment |
| $exp_1 \approx exp_2$ | $\hat{=}$ | $\Box(exp_1 = exp_2)$ | equal in interval |
| $exp_1 \leftarrow exp_2$ | $\hat{=}$ | finite $\wedge$ (fin $exp_1$) $= exp_2$ | temporal assignment |
| $exp_1$ gets $exp_2$ | $\hat{=}$ | keep $(exp_1 \leftarrow exp_2)$ | gets |
| stable $exp$ | $\hat{=}$ | $exp$ gets $exp$ | stability |
| padded $exp$ | $\hat{=}$ | (stable $(exp)$ ; skip) $\vee$ empty | padded expression |
| $exp_1 <\!\sim exp_2$ | $\hat{=}$ | $(exp_1 \leftarrow exp_2) \wedge$ padded $exp_1$ | padded temporal assignment |
| goodindex $exp$ | $\hat{=}$ | keep $(exp \leftarrow exp \vee exp \leftarrow exp + 1)$ | goodindex |
| intlen $(exp)$ | $\hat{=}$ | $\exists I \cdot (I = 0) \wedge (I$ gets $I + 1) \wedge (I \leftarrow exp)$ | interval length |

Source ITL home page at http //www.cse dmu ac.uk/~cau/itlhomepage/

Propositional axioms and rules for ITL are presented in Table 4.3-7. Cau and Moszkowski (1996) describe the development and implementation of a theorem prover and proof checker tool for ITL using the SRI's Prototype Verification System (PVS). This proof tool has been used to develop and verify an extensive library of ITL lemmas (STRL, 2006). A summary of selected ITL lemmas from this library that are used in this research is presented in Table 4.3-8.

Table 4.3-7 Propositional axioms and rules for ITL

| | | |
|---|---|---|
| ChopAssoc | $\vdash$ | $(f_0; f_1); f_2 \equiv f_0; (f_1; f_2)$ |
| OrChopImp | $\vdash$ | $(f_0 \vee f_1); f_2 \supset (f_0; f_2) \vee (f_1; f_2)$ |
| ChopOrImp | $\vdash$ | $f_0; (f_1 \vee f_2) \supset (f_0; f_1) \vee (f_0; f_2)$ |
| EmptyChop | $\vdash$ | $\mathsf{empty}; f_1 \equiv f_1$ |
| ChopEmpty | $\vdash$ | $f_1; \mathsf{empty} \equiv f_1$ |
| BiBoxChopImpChop | $\vdash$ | $\boxdot(f_0 \supset f_1) \wedge \Box(f_2 \supset f_3) \supset (f_0; f_2) \supset (f_1; f_3)$ |
| StateImpBi | $\vdash$ | $p \supset \boxdot p$ |
| NextImpNotNextNot | $\vdash$ | $\bigcirc f_0 \supset \neg\bigcirc\neg f_0$ |
| KeepnowImpNotKeepnowNot | $\vdash$ | $\mathsf{keepnow}(f_0) \supset \neg\, \mathsf{keepnow}(\neg f_0)$ |
| BoxInduct | $\vdash$ | $f_0 \wedge \Box(f_0 \supset \textcircled{w} f_0) \supset \Box f_0$ |
| InfChop | $\vdash$ | $(f_0 \wedge \mathsf{inf}); f_1 \equiv (f_0 \wedge \mathsf{inf})$ |
| ChopStarEqv | $\vdash$ | $f_0^* \equiv (\mathsf{empty} \vee ((f_0 \wedge \mathsf{more}); f_0^*))$ |
| ChopstarInduct | $\vdash$ | $(\mathsf{inf} \wedge f_0 \wedge \Box(f_0 \supset (f_1 \wedge \mathsf{fmore}); f_0)) \supset f_1^*$ |
| MP | $\vdash$ | $f_0 \supset f_1, \; \vdash f_0 \;\Rightarrow\; \vdash f_1$ |
| BoxGen | $\vdash$ | $f_0 \;\Rightarrow\; \vdash \Box f_0$ |
| BiGen | $\vdash$ | $f_0 \;\Rightarrow\; \vdash \boxdot f_0$ |

Source: ITL home page at http://www.cse.dmu.ac.uk/~cau/itlhomepage/

This overview of ITL is provided as a basis and background for the development of the formal rule model presented later in this chapter and the rule algebra developed throughout the remainder of this thesis. As necessary, the various elements of ITL summarized in this section are used in the development of the rule model and rule algebra presented in this thesis. Lemmas introduced later in this chapter and in subsequent chapters to define this rule model and rule algebra have been developed as part of this research using ITL.

Table 4.3-8 Summary of selected ITL lemmas used in this research

AndChopImp :
$\vdash ((f_0 \wedge f_1) ; f_2) \supset ((f_0 ; f_2) \wedge (f_1 ; f_2))$

ChopAndImp :
$\vdash (f_0 ; (f_1 \wedge f_2)) \supset ((f_0 ; f_1) \wedge (f_0 ; f_2))$

ChopOrEqv :
$\vdash (f_0 ; (f_1 \vee f_2)) \equiv ((f_0 ; f_1) \vee (f_0 ; f_2))$

NextAndNextEqvNextRule:
$\vdash ((f_0 \wedge f_1) \equiv f_2)$ implies $\vdash ((\circ f_0 \wedge \circ f_1) \equiv \circ f_2)$

NextChop :
$\vdash (\circ f_0 ; f_1) \equiv \circ(f_0 ; f_1)$

OrChopEqv :
$\vdash ((f_0 \vee f_1) ; f_2) \equiv ((f_0 ; f_2) \vee (f_1 ; f_2))$

StateAndChop:
$\vdash ((w \wedge f_0) ; f_1) \equiv (w \wedge (f_0 ; f_1))$

StateAndNextChop:
$\vdash ((w \wedge \circ f_0) ; f_1) \equiv (w \wedge \circ(f_0 ; f_1))$

StateChop:
$\vdash (w ; f_0) \supset w$

## 4.4 Previous Temporal Representations of State Properties

Various formations using temporal logic have been used to represent and reason about the relationship between current and future states and/or state properties. Although these formations are not always described as rules, they do demonstrate how different states can be linked temporally to form coherent logical structures.

Lamport (1977) introduced the 'leads to' operator to express a liveness property, where a liveness property requires that something must eventually happen. The 'leads to' operator was defined using temporal operators in Lamport (1980) as:

$$P \supset \Diamond Q \qquad (4.4\text{-}1)$$

where P and Q are assertions. Under this form, if P is true, then Q will be true eventually, either at the same time or at some later time. This concept was modified in Owicki and Lamport (1982) where the 'leads to' operator was defined as:

$$\Box(P \supset \Diamond Q) \qquad (4.4\text{-}2)$$

Under this formation of the 'leads to' operator, it is always true that if P ever becomes true, then Q will be true at the same time or at some later time, where P and Q are either immediate or temporal assertions.

Manna and Pnueli (1990) proposed a hierarchy of related formulations involving implication and temporal operators, where P and Q are state formulas or assertions:

| | | |
|---|---|---|
| Entailment: | $\Box(P \supset Q)$ | (4.4-3a) |
| Conditional guarantee: | $P \supset \Diamond Q$ | (4.4-3b) |
| Simple obligation: | $\Diamond P \supset \Diamond Q$ | (4.4-3c) |
| Obligation of exceptional occurrences: | $\Diamond P \supset \Diamond(Q \wedge \Diamond P)$ | (4.4-3d) |
| Response | $\Box(P \supset \Diamond Q)$ | (4.4-3e) |
| Conditional persistence: | $\Box(P \supset \Diamond\Box Q)$ | (4.4-3f) |
| Persistence-equivalent: | $P \supset \Diamond\Box Q$ | (4.4-3g) |
| Reactivity: | $\Box\Diamond P \supset \Box\Diamond Q$ | (4.4-3h) |

The interrelationship of some of these formulations is evident. Simple obligation is an extension of conditional guarantee; conditional persistence is an extension of response; and response incorporates entailment and conditional guarantee. Obligation of exceptional occurrences is a specific instantiation of simple obligation, as the authors observe that it guarantees that Q happens only after some occurrence of P.

Siewe et al. (2003) used ITL to express an 'always-followed-by' operator for reasoning about security policies. This 'always-followed-by' operator is defined as:

$$\Box(f \supset \Diamond(f\,;\, w)) \qquad (4.4\text{-}4)$$

where $f$ is a temporal formula and $w$ is a state formula.

## 4.5 A Temporal, State-Based Model of a Rule

As demonstrated in Section 4.4, temporal logic can be used to represent and reason about the relationship between current and future states and/or state properties. In these previous uses of temporal logic to express rule-like structures, implication has been consistently used to express the logical relationship between the formulas describing the current and future states. However, and as explained below, implication has an undesirable property with regard to the formation of rules – the vacuously true case. For implication, the vacuously true case exists when the antecedent is false and the consequent is true. The basis for the vacuously true case is evident when an implication, $f_0 \supset \circ f_1$, is expressed in its equivalent disjunctive form, $\neg f_0 \vee \circ f_1$. In this example, if $\circ f_1$ is true, the implication (and its equivalent disjunctive form) will hold regardless of the verity of the antecedent $f_0$. Whereas the logical necessity of the vacuously true case for implication is not questioned here, it does seriously weaken the use of implication as the basis for the formation of rules. This is less a logical problem and more an interpretive question of whether the definition of a rule using implication is the best alternative for expressing the formal, state-relationship basis for a rule, as developed above, and the informal expectations of what constitutes a rule, as previously discussed in Chapter 2.

As described in the previous sections, a rule is a relationship between a state and a future state. If the program is in a state or otherwise moved to a state such that the consequent of an implication-form rule is satisfied, then that implication-form rule is true by definition, even if the antecedent is false and the program is not in a state satisfying the rule condition expressed by the antecedent. Stated another way, using implication to form rules allows one to unequivocally declare that an implication-form rule describing the relation between two states is true even though the rule condition (i.e., the implication antecedent) is not met; only the consequent need be true for an implication-form rule to be true. The vacuously true case conflicts with the intuitive expectations of a rule and informal requirements previously presented in Chapter 2 that a rule be both explicit and precise with regard to what conditions must be met for the rule to hold. Therefore, an alternative logical formation – other than implication – is preferable for the formation and representation of rules.

Returning to the very basic temporal view of rules as presented in (4.2-1), a rule is conceptually represented as a conjunction of a rule condition, expressed in terms of a state, and a future state – the rule state – that results from the enforcement of the rule. Generalizing this to consider sequences from $\Sigma^+$, the set of all nonempty sequences of states, (4.2-1) can be recast in a form amenable to the use of ITL:

$$C(\sigma) \wedge \sigma'_{future} \text{ for } \sigma, \sigma' \in \Sigma^+ \tag{4.5-1}$$

where:

$\sigma'$ = rule state (or sequence of states) occurring in the future relative to $\sigma$ and described by the transition relationship $\delta$, relating $\sigma$ to $\sigma'$ where $\sigma' = \delta(\sigma)$.

$C(\sigma)$ = rule condition to be satisfied, expressed in terms of the state (or sequence of states) $\sigma$.

The general state sequence and temporal concepts presented in (4.5-1) can be formalized using ITL and a rule can be described as:

$$f_i \wedge \circ f_j \tag{4.5-2}$$

where:

$f_i$ = temporal (or state) formula in ITL describing a sequence of states (i.e., the rule condition) that must be met for the rule to hold.

$f_j$ = temporal (or state) formula in ITL describing a sequence of states (i.e., the rule state) that must occur for the rule to hold.

Regarding the correspondence between (4.5-1) and (4.5-2), $f_i$ describes the rule condition $\sigma$ that must be met; $f_j$ describes the rule state $\sigma'$ that must occur for the rule to hold; and the use of the ITL next operator $\circ$ specifies that the sequence of states satisfying $f_i$ must occur in the future relative to the sequence of states satisfying $f_j$ (subject to the specific semantics of the ITL next $\circ$ operator as presented in Table 4.3-2). As this rule form uses conjunction, no vacuously true case exists. In this form, the rule is true only if both the rule state, described by $f_i$, is achieved and all rule conditions, as expressed in $f_j$, are satisfied.

One final element must be added to complete the formalization of the concepts presented in (4.5-2). Remembering that a state is a function that maps a set of variables to a set of values and that $\delta$ is a transition relationship relating some state sequence $\sigma$ to some future state sequence $\sigma'$, $\sigma'$ differs from $\sigma$ based on changes to specific variables as specified by the transition relationship $\delta$. The variables that change values can be formalized under ITL using the frame extension described by Cau and Zedan (1997). Letting $W$ be a set of state variables, then *frame*($W$) denotes that only the variables in $W$ can possibly change in the transformation from $\sigma$ to $\sigma'$ as defined by $\delta$. The formal semantics of *frame*, expressed in ITL, are presented in Cau and Zedan (1997). This frame extension can be applied to (4.5-2), and the general form of a rule can be defined as:

$$W : f_i \wedge \circ f_i \tag{4.5-3}$$

where:

| | |
|---|---|
| $f_i =$ | temporal (or state) formula in ITL describing a sequence of states that must be met for the rule to hold. |
| $f_j =$ | temporal (or state) formula in ITL describing a sequence of states that must occur for the rule to hold. |
| $W =$ | set of state variables such that *frame*($W$) denotes that only the variables in $W$ can possibly change in the state transformation that occurs such that $f_i$ and $\circ f_j$ hold. |

Using the general form presented in (4.5-3), $f_i$ specifies the rule condition and $f_i$ describes the rule state resulting from the rule. When it is self-evident or otherwise not necessary that it be explicitly stated, $W$ can be inferred and need not be shown.

As ITL temporal formulas include state formulas (special temporal formulas whose verity is assessed based on only the first state of a sequence of states), (4.5-3) can be restricted to only state formulas and expressed as:

$$W : w_i \wedge \circ w_j \tag{4.5-4}$$

where:

| | |
|---|---|
| $w_i =$ | state formula in ITL describing the first state in a sequence of states that must be met for the rule to hold. |
| $w_j =$ | state formula in ITL describing the first state in a sequence of states that must occur for the rule to hold. |
| $W =$ | set of state variables such that *frame*($W$) denotes that only the variables in $W$ can possibly change values in the state transformation that occurs such that $w_i$ and $\circ w_j$ hold. |

Using the general form presented in (4.5-4), $w_i$ specifies the rule condition and $w_j$ describes the rule state resulting from the rule. Whereas the general form of (4.5-3) will be typically used for the general representation and analysis of rules, the state-restricted form of (4.5-4) will be occasionally used to express certain provable transformations that, although of limited scope, are especially applicable to certain procedural legacy code.

In Cau and Zedan (1997), a specification statement in ITL is described as having the syntax of $W : f$. As $f$ in this general specification statement is an ITL formula, $f$ can be instantiated with the ITL formula $f_i \wedge \circ f_j$ and $W : f_i \wedge \circ f_j$ is achieved. Therefore, the general rule form of (4.5-3) can be viewed as an extension of the specification statement that includes a conjunction of a sequence of states and a future sequence of states, in this case described using the ITL next operator $\circ$.

Cau and Zedan (1997) describe the semantics of the specification statement $W : f$ as *frame*($W$) $\wedge f$. Extending these semantics, the semantics of (4.5-3) is given by *frame*($W$) $\wedge f_i \wedge \circ f_j$. Applying propositional logic, specifically the elimination of conjunction, $f_i \wedge \circ f_j$ can be concluded from *frame*($W$) $\wedge f_i \wedge \circ f_j$. This conclusion is consistent with the previous assertion that $W$ need not be explicitly stated when it is self-evident or can be inferred from the specific rule instance.

This section closes with a final emphasis on the underlying concept that a rule is a temporal relationship between states, originally introduced in (4.1-1) and temporalized in (4.2-1). Let $\sigma_i$ and $\sigma_j$ be two intervals of states such that $\sigma_i, \sigma_j \in \Sigma^+$ and let $f_i$ and $f_j$ be valid temporal formulas expressed in ITL such that $\sigma_i \models f_i$ and $\sigma_j \models f_j$. By definition, if $f_i \wedge \circ f_j$ is true then there is a relationship $\rho_{rule}$ between $\sigma_i$ and $\sigma_j$. Whereas this relationship could be represented, with sufficient formal development, as $\sigma_i \, \rho_{rule} \, \sigma_j$ or $\rho_{rule}(\sigma_i, \sigma_j)$, this relationship will henceforth be described in terms of the general-form rule $f_i \wedge \circ f_j$ with the understanding that this general form rule describes the temporal relationship between $\sigma_i$ and $\sigma_j$.

## 4.6 Rules versus Rule Execution

As developed in this chapter, a rule is a relationship between a sequence of states and a future sequence of states, and is formally described conjunctively using ITL as $f_i \wedge \circ f_j$. Rules can be developed, that is, the relationship described, either observationally or prescriptively. If a program or specification is observed to exhibit a sequence of states that satisfies $f_i$ and in the next state that program or specification exhibits a sequence of states that satisfies $f_j$, then this behavior can be described by the rule $f_i \wedge \circ f_j$. (This observational construction is supported in propositional logic in that $p, q \vdash p \wedge q$). Similarly, if a program or specification is observed to exhibit a sequence of states that satisfies $f_j$, and in the previous state exhibits a sequence of states that satisfies $f_i$, then this behavior can also be described by the rule $f_i \wedge \circ f_j$. (Although ITL contains no past time operators, this reverse strategy relies on the observation that there is a sequence of states that satisfies $f_j$, that prior to that sequence there is a sequence that satisfies $f_i$, and that $\circ f_j$ is true relative to $f_i$.) Alternatively, a rule developer may prescribe or specify that, at some time, the program or specification will exhibit a sequence of states that satisfies $f_i$ and in the next state will exhibit a sequence of states that satisfies $f_j$. The rule developer may describe this relationship by the rule $f_i \wedge \circ f_j$. In all cases, and whether of observational or prescriptive origin, the relationship between sequences of states satisfying $f_i$ and $\circ f_j$ is described by the general-form rule $f_i \wedge \circ f_j$.

Whereas a rule may represent a relationship between states, it is only when a rule is executed that the future sequence of states embodied in that rule can be achieved. Therefore, a rule is executed at a specific time or under specific circumstances with the expectation of a specific outcome. The implication form defining the execution of a rule can be described by the following lemma:

LEMMA: ImpFormExecute

$\vdash f_0 \wedge \circ f_1$ implies $\vdash f_0 \supset (f_0 \wedge \circ f_1)$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 \wedge \circ f_1$ | premise |
| 2 | $\quad f_0$ | conditional proof assumption |
| 3 | $\quad f_0 \wedge \circ f_1$ | 1, reiteration |
| 4 | $f_0 \supset (f_0 \wedge \circ f_1)$ | 2-3, $\supset$ introduction |

Using ImpFormExecute, the execution form of the rule $f_0 \wedge \circ f_1$ can be described using implication as:

$$f_0 \supset (f_0 \wedge \circ f_1) \tag{4.6-1}$$

The formation may be clearer if this implication is read as "$f_0$ is sufficient for $f_0 \wedge \circ f_1$." Extending this interpretation, if the state, when the rule is executed, satisfies the rule condition described by $f_0$, this satisfaction is sufficient for the imposition and enforcement of the relationship described by the rule $f_0 \wedge \circ f_1$. And with the imposition of the rule $f_0 \wedge \circ f_1$, the next state satisfies $f_1$. Alternatively, using the traditional description of implication as *if...then*, rule execution can be described as follows: if the rule condition specified by $f_0$ is met then impose and enforce the rule $f_0 \wedge \circ f_1$ specifying that the next state will satisfy $f_1$.

With regard to the state sequences that may result from the rule execution form presented in (4.6-1), two alternative state sequences can be described with the equivalent disjunctive form of (4.6-1):

$$\neg f_0 \vee (f_0 \wedge \circ f_1) \tag{4.6-2}$$

As presented in (4.6-2), the state sequence resulting from the execution of the rule can be described either by $\neg f_0$ or $f_0 \wedge \circ f_1$. Stated another way, the execution of the rule $f_0 \wedge \circ f_1$ will result in either one of two state sequences – one satisfying $f_0 \wedge \circ f_1$ or one satisfying $\neg f_0$ – depending on the state at the time of rule execution.

The requirements for and outcome of the execution of the rule $f_0 \wedge \circ f_1$ using implication can be described by the following lemma:

LEMMA: RuleExecute

$\vdash f_0$ and $\vdash f_0 \supset (f_0 \wedge \circ f_1)$ implies $\vdash \circ f_1$

Proof:

| | | |
|---|---|---|
| 1 | $f_0$ | premise |
| 2 | $f_0 \supset (f_0 \wedge \circ f_1)$ | premise |
| 3 | $(f_0 \supset f_0) \wedge (f_0 \supset \circ f_1)$ | 2, distribution of $\supset$ over $\wedge$ |
| 4 | $true \wedge (f_0 \supset \circ f_1)$ | 3, propositional reasoning |
| 5 | $f_0 \supset \circ f_1$ | 4, unit of $\wedge$ |
| 6 | $\circ f_1$ | 1, 5, MP |

Under this lemma, if the program or specification is in a state satisfying $f_0$, and the rule $f_0 \wedge \circ f_1$ is executed, where the logic of that execution is described by the implication $f_0 \supset (f_0 \wedge \circ f_1)$, then the next state will satisfy $f_1$.

RuleExecute highlights the critical differentiation between and the logical separation of a rule and the execution of that rule. Rules define or describe the relationship between the rule condition and the rule state, expressed formally as the conjunctive relationship between a sequence of states satisfying $f_0$ and a future sequence of states satisfying $\circ f_1$, or $f_0 \wedge \circ f_1$. Rule execution describes the programmatic implementation of how this rule is called, executed, and/or enforced. Whereas the rule $f_0 \wedge \circ f_1$ may describe a relation between states and future states, this rule will only describe a specific state change to a sequence of states satisfying $\circ f_1$ only when the rule is executed. This distinction is critical for the logical and analytical separation between the knowledge that rules incorporate and the programmatic implementation of those rules.

Although a slightly shorter proof for RuleExecute is possible, careful analysis of the approach used yields another lemma regarding the representation of the programmatic implementation of rules with implication.

LEMMA: RuleExecuteEqvImp

$\vdash f_0 \supset (f_0 \wedge \circ f_1) \equiv f_0 \supset \circ f_1$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 \supset (f_0 \wedge \circ f_1) \equiv f_0 \supset (f_0 \wedge \circ f_1)$ | tautology |
| 2 | $f_0 \supset (f_0 \wedge \circ f_1) \equiv (f_0 \supset f_0) \wedge (f_0 \supset \circ f_1)$ | 1, distribution of $\supset$ over $\wedge$ |
| 3 | $f_0 \supset (f_0 \wedge \circ f_1) \equiv \mathit{true} \wedge (f_0 \supset \circ f_1)$ | 2, propositional reasoning |
| 4 | $f_0 \supset (f_0 \wedge \circ f_1) \equiv f_0 \supset \circ f_1$ | 3, unit of $\wedge$ |

RuleExecuteEqvImp demonstrates that the rule execution form $f_0 \supset (f_0 \wedge \circ f_1)$ is logically equivalent to the simple implication form $f_0 \supset \circ f_1$. Using the conjunctive model of a rule as presented in this thesis, RuleExecuteEqvImp supports a conclusion that the common view of a single rule-like structure as implication is actually a logical description of the execution of a rule and not a logical description of the rule itself, where a rule is a temporal relationship between two state sequences.

## 4.7 Observations

In this chapter, a rule has been defined formally as a conjunctive relationship between a state sequence and a future state sequence. This relationship is described in ITL as the general-form rule $f_i \wedge \circ f_j$. This rule form can be used to either describe or specify, either observationally or prescriptively, a temporal relation between a state sequence satisfying the rule condition $f_i$, and a state sequence satisfying the rule state $f_j$.

Unlike the traditional use of implication to represent rules, this conjunctive form avoids the troubling vacuously true case associated with implication. Using implication to form rules allows one to unequivocally declare that an implication-form rule describing the relation between two states is true even though the rule condition (i.e., the implication antecedent) is not met. This is troubling because the vacuously true case conflicts with the intuitive expectations of a rule and informal requirements previously presented in Chapter 2 that a rule be both explicit and precise with regard to what conditions must be met for the rule to hold. The conjunctive general-form rule $f_i \wedge \circ f_j$ avoids the problem. However, with courteous regard to the traditional (and arguably incorrect) view of rules as implication, proof was given that the execution of the general-form rule, described using implication as $f_0 \supset (f_0 \wedge \circ f_1)$, is logically equivalent to the simple implication form $f_0 \supset \circ f_1$.

A critical objective in the development of this rule model was the general adaptability of the rule model to a variety of programming paradigms, so that it can be applied in concert with the general rule extraction framework developed in Chapter 3. However, as the goal of many rule extraction exercises is the development of a new specification or program that will implement the extracted rules, this rule model should be equally adaptable to forward engineering. In the next chapters, a rule algebra is developed that describes how the general-form rule $f_i \wedge \circ f_j$ can be used to describe complicated state sequence in either the reverse or forward engineering domains.

# Chapter 5

# Rule Algebra – Fundamentals

The modern word *algebra* originates from the Arabic word *al-jebr* meaning "reunion of broken parts" (Oxford, 1971) or "reduction of parts to a whole" (Merriam-Webster, 1998); *al-jebr* is derived from the Arabic word *jabara* meaning "reunite, ... consolidate, restore" (Oxford, 1971) or "to bind together" (Merriam-Webster, 1998). Because the next two chapters are focused on how rules can be created systematically from component parts including other rules and then linked to form larger structures, these origins of the word *algebra* are particularly enlightening and appropriate.

Numerous definitions for algebras or algebraic systems exist in the modern mathematics and computer science canon (Birkhoff and MacLane, 1977; Buchi, 1989; Burris and Sankappanavar, 1981; Gill, 1976; Hungerford, 1974; Levy, 1980; Stanat and McAllister, 1977). For this thesis, a very general definition is used – that an algebra is a structure composed of sets of objects and operations on those objects (Denecke and Wismath, 2002). For this rule algebra, these objects are states and state sequences specified by a rule or collection of rules. Using the general formal model presented in Chapter 4, a rule algebra is presented that describes the set of operations that can be applied to compose, decompose, or transform those rules to describe other sequences of states. In this chapter, the fundamentals of this rule algebra are presented. Whereas some relatively simple rules are analyzed in the development of the fundamentals of this rule algebra, these simple rules are included to demonstrate how this rule algebra can be used to describe other simple relations typically presented in the mathematical canon. While simple, these fundamental rules and the associated proofs are far from trivial as they provide the reader a sound basis for understanding both the rule model and the more advanced elements of the rule algebra that follow. In the next chapter (Chapter 6), advanced concepts associated with this rule algebra are developed using the fundamentals presented in this chapter.

## 5.1 Rules, Total Rules, and Rule Systems

Consider the following general-form rule:

$$f_0 \wedge \circ f_1 \qquad (5.1\text{-}1)$$

This rule is satisfied if the rule condition $f_0$ is satisfied (i.e., true) and the next sequence of states satisfies $f_1$. Assuming the system currently exhibits a state sequence satisfying $f_0$, execution of this rule can be performed as described by ImpFormExecute and RuleExecute (previously presented in Chapter 4), and $\circ f_1$ can be concluded. However, no information is provided regarding the future state sequence associated with or related to the non-satisfaction of $f_0$. Three cases exist if $f_0$ is false, as described below.

In the first case, as described above, no explicit representation is made with regard to the next state sequence in the event of the non-satisfaction of the rule condition $f_0$. Stated another way, no complementary rule including $\neg f_0$ as the rule condition is specified. Therefore, in the event of $\neg f_0$, the next state sequence and any associated changes in system state are governed by other aspects of the system, and are not described by this rule. These controlling elements may include but not limited to the presence or absence of an overall frame axiom specifying that state variables do not change unless explicitly changed. In the absence of any information about such aspects such as an overall frame axiom, or unless redefined by a subsequent formula, the next state after $\neg f_0$ is undefined.

In the remaining two cases, a rule addressing the non-satisfaction of the rule condition is explicitly stated and a resulting rule state sequence specified. In these cases, both the satisfaction and non-satisfaction of the rule condition are considered, and these complementary rule pairs are referred to as total rules.

In the second case, if the rule condition $f_0$ is false, a complementary rule can be defined that specifies the relationship between the state sequences satisfying $\neg f_0$ and a next sequence of states satisfying $f_2$:

$$\neg f_0 \wedge \circ f_2 \qquad (5.1\text{-}2)$$

Applying ImpFormExecute and RuleExecute, when the rule specified in (5.1-2) is executed from a state sequence satisfying $\neg f_0$, the next state sequence will satisfy $f_2$.

The coordinated execution of the total rule defined by complementary rule pair $f_0 \wedge \circ f_1$ and $\neg f_0 \wedge \circ f_2$ is described by the following lemma:

LEMMA: TotalFormExecute

$\vdash f_0 \wedge \circ f_1$ and $\vdash \neg f_0 \wedge \circ f_2$ implies $\vdash (f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2)$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 \wedge \circ f_1$ | premise |
| 2 | $\neg f_0 \wedge \circ f_2$ | premise |
| 3 | $f_0 \supset (f_0 \wedge \circ f_1)$ | 1, ImpFormExecute |
| 4 | $\neg f_0 \supset (\neg f_0 \wedge \circ f_2)$ | 2, ImpFormExecute |
| 5 | $f_0 \supset \circ f_1$ | 3, RuleExecuteEqvImp |
| 6 | $\neg f_0 \supset \circ f_2$ | 4, RuleExecuteEqvImp |
| 7 | $(f_0 \supset \circ f_1) \wedge (\neg f_0 \supset \circ f_2)$ | 5, 6, $\wedge$ introduction |
| 8 | $(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2)$ | 7, propositional reasoning |

Using TotalFormExecute and given the complementary rule pair $f_0 \wedge \circ f_1$ and $\neg f_0 \wedge \circ f_2$, the total rule execution form $(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2)$ may be concluded. In this form, the total rule execution form is a disjunction of the two complementary general-form rules. This total rule execution form corresponds with the logical form of the ITL expression of the concrete derived construct if-then-else, as presented in Table 4.3-5, with the notable exception that this total rule form $(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2)$ includes the ITL next operator in the specification of the rule state. (The if-then-else construct is discussed in detail in Chapter 6.)

The third case is a specialized form of the total rule presented in the previous case. For this case, consider the rule $W: (f_0 \wedge \circ f_1)$. For those state sequences that do not satisfy $f_0$, a complementary rule can be defined that specifies that the system state remains unchanged:

$$W: (\neg f_0 \wedge \circ f_{unchanged}) \tag{5.1-3}$$

In this complementary rule, the temporal formula $f_{unchanged}$ specifies that the system state remains unchanged. The formal semantics of $f_{unchanged}$ are defined as follows using an interpretation $\mathcal{M}_\sigma$ that gives meaning to expressions and formulas over an interval $\sigma$:

$$\mathcal{M}_\sigma[\![f_{unchanged}]\!] = \mathit{true} \text{ iff for all } v \in W,\ \mathcal{M}_\sigma[\![\mathsf{stable}(v)]\!] \tag{5.1-4}$$

Because the semantics of $f_{unchanged}$ specifies that all frame variable values remain stable (using the ITL stable construct), the explicit statement of the frame $W$ in rules of this form will be omitted unless otherwise needed.

With this, the total rule defined by the complementary rule pair $f_0 \wedge \circ f_1$ and $\neg f_0 \wedge \circ f_{unchanged}$ is used to specify that a system exhibiting a state sequence satisfying $f_0$ be moved in the next state to a state sequence satisfying $f_1$. Otherwise, if the system does not satisfy $f_0$, all state variables remain unchanged in the next state. Applying TotalFormExecute to the total rule described by $f_0 \wedge \circ f_1$ and $\neg f_0 \wedge \circ f_{unchanged}$, the total rule execution form $(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_{unchanged})$ is concluded.

The specific use of form $\circ f_{unchanged}$ to formalize the perpetuation of the system in the unchanged state is important. The temporal formula $\circ f_{unchanged}$ is defined in ITL as skip ;$f_{unchanged}$, with skip adding one unit interval to the state sequence by definition. The semantics of $f_{unchanged}$ specify that no variables in the frame may change value. Therefore, the imposition of the temporal formula $\circ f_{unchanged}$ creates a sequence of two identical states $\ldots s_n s_{n+1} \ldots$ where $s_n = s_{n+1}$. Lamport (1994) describes such a transition as a stuttering step, and Milner (1980) symbolizes such silent and unobservable transitions between states as $\tau$. Whereas the possible removal of such silent steps in some algebras is noted (e.g., Baeten and Weijland, 1990), the purposeful and uniform use of $\circ f_{unchanged}$ allows all rules, including those that intentionally do not result in a state change, to be represented using the general rule form $f_i \wedge \circ f_j$. The convenience and advantages of this logical consistency will become evident as the rule algebra presented herein is developed.

Frequently, rules are discussed and/or analyzed within the context of a rule system. For the purposes of this research, a rule system is defined as a collection of two or more related rules. Rules included in these rule systems may be presented individually, expressed disjunctively (as described in the following paragraphs), or composed in other ways (as described later in this chapter). As will be demonstrated, some multiple rule systems may be transformable into a single general-form rule, but there is no requirement that this always hold. No formal restriction is placed on what can be described as related with respect to defining a rule system.

The disjunctive association of rules from a given rule system is often a convenient and powerful way to logically associate related rules into a single structure. This disjunctive association may be allowed based on reasoning about the specifics of given rule system, or may be allowed based on the application of propositional logic. In certain rule systems, two rules may be related disjunctively because a third way is not given. With respect to rules, this reasoning is generally analogous to, but not directly derivative of, the law of the excluded middle. Alternatively, using propositional logic, any rule, regardless of its verity, may be added disjunctively to a true rule (i.e., the law of addition or $\vee$ introduction). This section examines how rules can be disjunctively associated, and under what circumstances such associations are accretive with regard to a rule algebra in that useful transformations may be enabled by such associations. Four disjunctive associations are examined: between rules that share a common rule condition; between rules that share a common rule state; between two rules with complementary rule conditions; and between two disjoint rules.

Consider the system depicted in Figure 5.1-1 containing three states ($s_0$, $s_1$, and $s_2$) and two transitions linking the three states.

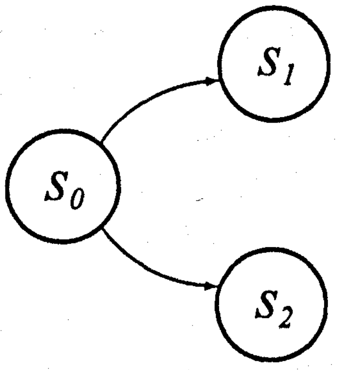


Figure 5.1-1: Three-State Rule System with Rules Sharing a Common Rule Condition

Three state formulas, $w_0$, $w_1$, and $w_2$, are used to describe this system, where $s_0 \models w_0$, $s_1 \models w_1$, and $s_2 \models w_2$. Multiple-state state sequences starting with one of the specified states will also satisfy the respective state formula. The two state transitions included in this system are described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

| | | |
|---|---|---|
| $s_0$ | $w_0 \wedge \circ w_1$ | (5.1-5a) |
| $s_0$ | $w_0 \wedge \circ w_2$ | (5.1-5b) |
| $s_1$ | – | |
| $s_2$ | – | |

As identified in (5.1-5a) and (5.1-5b), the rule conditions for both rules in this system are satisfied by state sequences that begin with $s_0$. Given that these two rules describe the two and only two relations associated with $s_0$, these rules with a common rule condition can be combined disjunctively as:

$$(w_0 \wedge \circ w_1) \vee (w_0 \wedge \circ w_2) \qquad (5.1\text{-}6)$$

An equivalence transformation to transform these two disjunctively associated rules sharing a common rule condition to a single general-form rule is presented in the following lemma:

LEMMA: CommonRuleCondEqv

$\vdash (f_0 \wedge \circ f_1) \vee (f_0 \wedge \circ f_2) \equiv f_0 \wedge \circ(f_1 \vee f_2)$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge \circ f_1) \vee (f_0 \wedge \circ f_2) \equiv (f_0 \wedge \circ f_1) \vee (f_0 \wedge \circ f_2)$ | Tautology |
| 2 | $(f_0 \wedge \circ f_1) \vee (f_0 \wedge \circ f_2) \equiv f_0 \wedge (\circ f_1 \vee \circ f_2)$ | 1, distribution of $\wedge$ over $\vee$ |
| 3 | $(f_0 \wedge \circ f_1) \vee (f_0 \wedge \circ f_2) \equiv f_0 \wedge \circ(f_1 \vee f_2)$ | 2, ITL (ChopOrEqv) |

Applying CommonRuleCondEqv to (5.1-6) yields:

$$w_0 \wedge \circ(w_1 \vee w_2) \qquad (5.1\text{-}7)$$

This demonstrates how disjunctively associating two rules sharing a common rule condition and then applying CommonRuleCondEqv to that disjunctive structure allows the two related rules to be expressed as one equivalent general-form rule.

Consider the system depicted in Figure 5.1-2 containing three states ($s_0$, $s_1$, and $s_2$) and two transitions linking the three states.

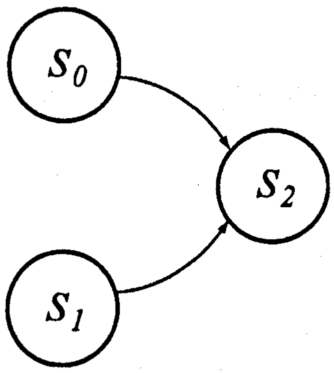


Figure 5.1-2: Three-State Rule System with Rules Sharing a Common Rule State

Three state formulas, $w_0$, $w_1$, and $w_2$, are used to describe this system, where $s_0 \models w_0$, $s_1 \models w_1$, and $s_2 \models w_2$. Multiple-state state sequences starting with one of the specified states will also satisfy the respective state formula. The two state transitions included in this system are described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

| | | |
|---|---|---|
| $s_0$ | $w_0 \wedge \circ w_2$ | (5.1-8a) |
| $s_1$ | $w_1 \wedge \circ w_2$ | (5.1-8b) |
| $s_2$ | – | |

As identified in (5.1-8a) and (5.1-8b), the rule states for both rules in this system are satisfied by state sequences that begin with $s_2$. Given that these two rules describe the two and only two relations associated with $s_2$, these rules with a common rule state can be combined disjunctively as:

$$(w_0 \wedge \circ w_2) \vee (w_1 \wedge \circ w_2) \tag{5.1-9}$$

An equivalence transformation to transform these two disjunctively associated rules sharing a common rule state to a single general-form rule is presented in the following lemma.

LEMMA: CommonRuleStateEqv

$\vdash (f_0 \wedge \circ f_2) \vee (f_1 \wedge \circ f_2) \equiv (f_0 \vee f_1) \wedge \circ f_2$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge \circ f_2) \vee (f_1 \wedge \circ f_2) \equiv (f_0 \wedge \circ f_2) \vee (f_1 \wedge \circ f_2)$ | tautology |
| 2 | $(f_0 \wedge \circ f_1) \vee (f_0 \wedge \circ f_2) \equiv (f_0 \vee f_1) \wedge \circ f_2$ | 1, distribution of $\wedge$ over $\vee$ |

Applying CommonRuleStateEqv to (5.1-9) yields:

$$(w_0 \vee w_1) \wedge \circ w_2 \tag{5.1-10}$$

This demonstrates how disjunctively associating two rules sharing a common rule state and then applying CommonRuleStateEqv to that disjunctive structure allows the two related rules to be expressed as one equivalent general-form rule.

Consider the following system of two rules that contain complementary rule conditions:

$$f_0 \wedge \circ f_1 \tag{5.1-11a}$$
$$\neg f_0 \wedge \circ f_2 \tag{5.1-11b}$$

These two rules can be combined disjunctively to form:

$$(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2) \tag{5.1-12}$$

(5.1-12) is the previously discussed total rule form. Applying propositional logic (i.e., the distribution of $\vee$ over $\wedge$ and $\wedge$ elimination) yields:

$$f_0 \vee \neg f_0 \tag{5.1-13}$$

As demonstrated with (5.1-13), all state sequences will satisfy the rule conditions included in the total rule form of (5.1-12). Whereas the verity of (5.1-12) can be assured only by either $\circ f_1$ or $\circ f_2$, (5.1-13) highlights, but does not prove, the basis for the disjunctive association of two rules containing complementary rule conditions.

Consider the following system of two rules that exhibit no obvious relationship:

$$f_0 \wedge \circ f_1 \tag{5.1-14a}$$
$$f_2 \wedge \circ f_3 \tag{5.1-14b}$$

Assuming one of these rules is known to hold for the given system, the other can be added disjunctively to form:

$$(f_0 \wedge \circ f_1) \vee (f_2 \wedge \circ f_3) \tag{5.1-15}$$

Expanding (5.1-15) with propositional logic yields:

$$(f_0 \vee f_2) \wedge (\circ f_1 \vee f_2) \wedge (f_0 \vee \circ f_3) \wedge (\circ f_1 \vee \circ f_3) \tag{5.1-16}$$

Given the assumption that one of the two rules at (5.1-14a) and (5.1-14b) holds, the verity of (5.1-15) and (5.1-16) is assured. However, in the absence of any additional information regarding any other relationships between the contributing formulas and/or the corresponding state sequences, no other revealing transformations can be made. Although allowable within the propositional calculus (given an assumption that one of the two rules is known to hold), disjunctively associating two rules that share no state sequences or do not include complementary state sequences as rule conditions offers no transformational advantage.

## 5.2 Rule Domain, Rule Codomain, and Rule Universe

Whereas rules as defined in this thesis are not functions, certain concepts that are used to describe functions are useful in understanding rules. In this section, the concepts of domain and codomain are adapted to rules, and the derivative concept of the rule universe is introduced.

## 5.2.1 Rule Domain

The domain of a given rule is defined as the set of state sequences that satisfy the rule condition associated with that given rule. Consider the following rule expressed in terms of state formulas:

$$w_0 \wedge \circ w_1 \tag{5.2.1-1}$$

Under this rule, the rule domain is the set of all states (or initial states in state sequences) that will satisfy the specified rule condition $w_0$. Consider the following rule expressed in terms of temporal formulas:

$$f_0 \wedge \circ f_1 \tag{5.2.1-2}$$

Under this rule, the rule domain is the set of all state sequences that will satisfy the specified rule condition $f_0$.

Formally, the rule domain $domain_{rule}$ for a general rule *rule*, defined as $f_0 \wedge \circ f_1$, is defined in terms of a state sequence $\sigma$ as:

$$domain_{rule} \triangleq \{\sigma \in \Sigma^+ \mid \sigma \models f_0\} \tag{5.2.1-3}$$

Because the temporal formula $f$ is inclusive of the state formula $w$, (5.2.1-3) describes the rule domain for both (5.2.1-1) and (5.2.1-2). Future definitions and analyses are expressed in terms of temporal formulas only, unless there is an explicit need for the distinct presentation of the state formula case.

With respect to state formulas, it is tempting to think in terms of only a single state satisfying a state formula, for example, $s_0 \models w_0$. However, in ITL, any multiple state sequence that starts with the specified single state also satisfies the associated state formula. Continuing the previous example regarding the state formula $w_0$, $s_0s_1 \models w_0$, $s_0s_1s_2 \models w_0$, $s_0s_n... \models w_0$, etc. Taken to the limit, an infinite number of state sequences could satisfy the relevant state formula. Therefore, for the purposes of this thesis, any time a state is specified as satisfying a given state formula, it is understood that all multiple-state state sequences starting with that specified state will also satisfy that state formula. Consistent with this convention, the term "minimum rule domain" is used to describe the individual states that satisfy a given state formula, reflecting the understanding that any multiple-state state sequences starting with any one of the individual states specified in the domain will also satisfy the respective state formula. This concept will also be used in describing the rule codomain and rule universe of rules composed with state formula.

Because the rule domain is a set of states or state sequences, the rule domain for a rule system is described as the union of the rules domains of the rules that comprise the rule system. Formally, for a rule system *rs* consisting of *n* rules $rule_1$ through $rule_n$, the domain of the rule system *rs* is described as:

$$domain_{rs} = \bigcup_{1}^{n} domain_{rule_n} \qquad (5.2.1\text{-}4)$$

### 5.2.2 Rule Codomain

The codomain or range of a given rule is the set of all rule states that are related to the states in the rule domain. For a general rule *rule,* defined as $f_0 \wedge \circ f_1$ where $\sigma \models f_0$, $\sigma' = \delta_{rule}(\sigma)$, and $\sigma' \models f_1$, the rule codomain is the set of all rule states that satisfy $\circ f_1$ when the rule condition $f_0$ is satisfied. The rule codomain $codomain_{rule}$ for a general rule *rule* is defined as:

$$codomain_{rule} \triangleq \{\sigma \in domain_{rule} \mid \delta_{rule}(\sigma)\} \qquad (5.2.2\text{-}1)$$

For a rule system *rs* consisting of *n* rules $rule_1$ through $rule_n$, the codomain of the rule system *rs* is described as:

$$codomain_{rs} = \bigcup_{1}^{n} codomain_{rule_n} \qquad (5.2.2\text{-}2)$$

### 5.2.3 Rule Universe

The rule universe represents all state sequences associated with a rule as part of the rule condition or the rule state. Formally, the rule universe is defined as:

$$universe_{rule} \triangleq domain_{rule} \cup codomain_{rule} \tag{5.2.3-1}$$

As defined above, the rule universe includes all state sequences in the rule domain and the rule codomain. The union described in (5.2.3-1) is feasible because both $domain_{rule}$ and $codomain_{rule}$ are defined as sets of state sequences. That $domain_{rule}$ and $codomain_{rule}$ are both sets of state sequences suggests that the codomain of one rule can be the domain of another rule, thereby allowing rules to be related sequentially.

For a rule system *rs*, the rule universe for that rule system is defined similarly:

$$universe_{rs} \triangleq domain_{rs} \cup codomain_{rs} \tag{5.2.3-2}$$

## 5.3 Rule Satisfiability

To be useful, rules must be satisfiable. A rule that is not satisfiable is both trivial and useless; it cannot describe relationships between states and represents no knowledge. Extending the definition of satisfiability from propositional logic, the rule $f_0 \wedge \circ f_1$ is satisfiable if there exists some set of state sequences such that both the rule condition and the rule state are satisfiable, i.e., $f_0 = true$ and $\circ f_1 = true$, and satisfiable in such a way that conjunction defining the relation between $f_0$ and $\circ f_1$ holds. Rule satisfiability is closely allied to the concepts of rule domain and rule codomain, as discussed below.

Formally, given a general rule *rule* defined as $f_0 \wedge \circ f_1$, the rule condition $f_0$ is satisfiable if:

$$\exists \sigma \in \Sigma^{+} \mid \sigma \models f_0 \tag{5.3-1}$$

The similarities of this formal definition of rule condition satisfiability with the definition of the rule domain should be noted. Given the necessity of rule condition satisfiability, a rule may not have the empty set for the rule domain. Similarly, the rule state $f_1$ is satisfiable if:

$$\exists \sigma' \in \Sigma^+ \mid \sigma' \models f_1 \tag{5.3-2}$$

As a conjunctive relationship, *rule* is satisfiable if a rule state exists for every state sequence that satisfies the rule condition. Formally, given the previously defined *rule* where $f_0 \wedge \circ f_1$, $\sigma \models f_0$, $\sigma' \models f_1$, and $\sigma' = \delta_{rule}(\sigma)$, *rule* is satisfiable if :

$$\forall \sigma \in domain_{rule} \mid \sigma' \tag{5.3-3a}$$

or

$$\forall \sigma \in domain_{rule} \mid \delta_{rule}(\sigma) \tag{5.3-3b}$$

Stated another way, if a rule state is not associated with every state sequence that satisfies the rule condition, the rule does not describe a valid relation. The similarities of this formal definition of rule satisfiability with the formal definition of the rule codomain should be noted.

These concepts are readily expandable to total rules and rule systems. For example, given a total rule $(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2)$, the complementary rule conditions $f_0$ and $\neg f_0$ assure that all states will satisfy a rule condition. If $f_1$ and $f_2$ are properly formed, as described in (5.3-3), then the satisfiability of the total rule is assured.

## 5.4 Injective, Surjective, and Bijective Rules

As defined in this thesis, rules are not functions, and certain concepts used to describe functions may not be applicable to rules. The applicability of the terms injective, surjective, and bijective to the description and analysis of rules is discussed in this section.

### 5.4.1 Injective Rules

A rule is injective, or one-to-one, if all unique state sequences in the rule domain result in unique state sequences in the rule codomain. Formally, given a general rule *rule* defined as $f_0 \wedge \circ f_1$, *rule* is injective if :

$$\forall\ \sigma_0, \sigma_0' \in domain_{rule} \wedge \forall\ \sigma_1, \sigma_1' \in codomain_{rule} \\ \mid\ \sigma_0 \neq \sigma_0' \supset \sigma_1 \neq \sigma_1' \qquad (5.4.1\text{-}1)$$

Because $\sigma_0$ and $\sigma_0'$ are elements of $domain_{rule}$, then by definition, $\sigma_0 \models f_0$ and $\sigma_0' \models f_0$. Similarly, because $\sigma_1, \sigma_1' \in codomain_{rule}$, $\sigma_1 \models f_1$ and $\sigma_1' \models f_1$.

### 5.4.2 Surjective Rules

A rule is surjective, or onto, if all state sequences in the rule codomain can be reached from the rule domain. However, rule codomain has been previously defined in terms of the rule domain, and by definition all state sequences in the rule codomain must be associated with at least one state sequence in the rule domain. Therefore, all rules are surjective rules. Therefore, a classification of rules as surjective is redundant, and the term surjective is not used to describe rules.

### 5.4.3 Bijective Rules

A rule is bijective if that rule is both injective and surjective. Because all rules are surjective rules, any rule that is injective is also bijective. Therefore, classification of such rules as bijective is redundant, and the term bijective is not used to describe rules.

## 5.5 Inverse Rules and Invertible Rules

Consider two rules, *rule* and *rule'*. By definition, *rule'* may be described as the inverse of *rule* if:

*a.* *rule* and *rule'* are injective rules
*b.* $domain_{rule} = codomain_{rule'}$
*c.* $domain_{rule'} = codomain_{rule}$
*d.* $\delta_{rule} = (\delta_{rule'})^{-1}$

If these criteria are met, then the sequential execution of *rule* and *rule'* will leave the system in the state that existed prior to the execution of *rule.*

An invertible rule is a rule that can have an inverse rule, although the inverse rule need not be specified. Stated another way, for a rule to be invertible, all of the above criteria must be met regarding the rule and the associated inverse rule. Failure to meet these criteria can be used to demonstrate that a rule is not invertible.

## 5.6 Sequentially Relating Two Rules

### 5.6.1 Sequentially Associating Rules Using the General Rule Form

Consider the following general rule:

$$f_i \wedge \circ f_j \tag{5.6.1-1}$$

This rule is a temporal formula that is itself composed of two temporal formulas – $f_i$ describing the rule condition and $f_j$ describing the rule state. Because no limitation has been placed on the temporal formulas used to express $f_i$ or $f_j$, and because general-form rules are themselves temporal formulas, rules can be used within rules to specify the rule state and rule conditions in another rule. Instantiating $f_i$ with $f_0 \wedge \circ f_1$ and instantiating $f_j$ with $f_1 \wedge \circ f_2$, the general rule presented in (5.6.1-1) can be instantiated as:

$$(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \tag{5.6.1-2}$$

In (5.6.1-2), the rule condition is described by the rule $f_0 \wedge \circ f_1$ and the rule state is described by the rule $f_1 \wedge \circ f_2$. Because $f_1$ is used both in the specification of the rule condition and the rule state, (5.6.1-2) describes a relationship between two related rules. Composed in this form, this rule specifies the sequential relationship between the common state sequences considered in two different rules.

This sequential composition using the general rule form is demonstrated using the following example. Consider the system depicted in Figure 5.6.1-1 containing three state sequences ($\sigma_0$, $\sigma_1$, and $\sigma_2$) and two transitions linking the three state sequences.

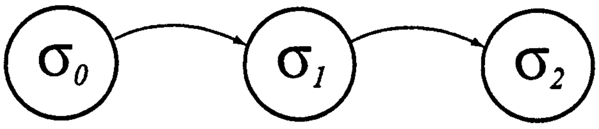


Figure 5.6.1-1: Three-Sequence State Transition Diagram

Three temporal formulas, $f_0$, $f_1$, and $f_2$, are used to describe this system, where $\sigma_0 \models f_0$, $\sigma_1 \models f_1$, and $\sigma_2 \models f_2$. The two transitions included in this system are described in rule form and organized based on the state sequence satisfying the corresponding rule condition:

|  |  |  |
|---|---|---|
| $\sigma_0$ | $f_0 \wedge \circ f_1$ | (5.6.1-3a) |
| $\sigma_1$ | $f_1 \wedge \circ f_2$ | (5.6.1-3b) |
| $\sigma_2$ | – |  |

Because these transitions share a state sequence (i.e., the final state sequence of one transition is the initial state sequence of the another transition), the general rule form $f_i \wedge \circ f_j$ can be applied to relate the state sequences described by each rule. With is rule-form sequential composition of these two rules, a new rule is formed that describes a state sequence that may result from this system:

$$(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \qquad (5.6.1\text{-}4)$$

The following lemmas describe some possible manipulations and reductions of two rules sequentially composed using the general rule form as the basis for composition. NextAndDistEqv, used in the following proofs, is presented in Appendix A.

LEMMA: TwoSeqRulesEqv1

$\vdash (f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \equiv f_0 \wedge \circ f_1 \wedge \circ\circ f_2$

Proof:

|  |  |  |
|---|---|---|
| 1 | $(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \equiv (f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2)$ | tautology |
| 2 | $(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \equiv f_0 \wedge \circ f_1 \wedge \circ f_1 \wedge \circ\circ f_2$ | 1, NextAndDistEqv |
| 3 | $(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \equiv f_0 \wedge \circ f_1 \wedge \circ\circ f_2$ | 2, idempotence of $\wedge$ |

LEMMA: TwoSeqRulesEqv2

$\vdash (f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \equiv f_0 \wedge \circ(f_1 \wedge \circ f_2)$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \equiv f_0 \wedge \circ f_1 \wedge \circ\circ f_2$ | TwoSeqRulesEqv1 |
| 2 | $(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \equiv f_0 \wedge \circ(f_1 \wedge \circ f_2)$ | 1, NextAndDistEqv |

LEMMA: TwoSeqRulesImp

$\vdash (f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2)$ implies $\vdash f_0 \wedge \circ\circ f_2$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2)$ | Premise |
| 2 | $f_0 \wedge \circ f_1 \wedge \circ\circ f_2$ | 1, TwoSeqRulesEqv1 |
| 3 | $f_0 \wedge \circ\circ f_2$ | 2, $\wedge$elimination |

The outcomes of both TwoSeqRulesEqv2 and TwoSeqRulesImp are expressed in the general rule form $f_i \wedge \circ f_j$. In both cases, the rule condition is $f_0$. In TwoSeqRulesEqv2, the rule state is $(f_1 \wedge \circ f_2)$. In TwoSeqRulesImp, the rule state is $\circ f_2$.

Applying TwoSeqRulesEqv1 to (5.6.1-4), the equivalent form $f_0 \wedge \circ f_1 \wedge \circ\circ f_2$ is obtained and the sequential nature of the original structure is clear. Assuming that at the time of rule execution the system satisfies $f_0$, TwoSeqRulesEqv2 is applied in the following execution of the rule presented in (5.6.1-4):

| | | |
|---|---|---|
| 1 | $f_0$ | premise |
| 2 | $(f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2)$ | premise |
| 3 | $f_0 \wedge \circ(f_1 \wedge \circ f_2)$ | 2, TwoSeqRulesEqv2 |
| 4 | $f_0 \supset (f_0 \wedge \circ(f_1 \wedge \circ f_2))$ | 3, ImpFormExecute |
| 5 | $\circ(f_1 \wedge \circ f_2)$ | 1, 4, RuleExecute |
| 6 | $\circ f_1 \wedge \circ\circ f_2$ | 5, NextAndDistEqv |
| 7 | $\circ\circ f_2$ | $\wedge$elimination |

As demonstrated above, sequentially composing the rule system presented in (5.6.1-3a) and (5.6.1-3b) using the general rule form into rule (5.6.1-4), and executing that rule from a state sequence satisfying $f_0$ results in a state sequence that satisfies $\circ\circ f_2$. A similar result can be achieved by using TwoSeqRulesImp. This general rule execution strategy is applicable to many of the rule transformations that will be presented in the chapter and will not be demonstrated again.

### 5.6.2 Sequential Composition with Chop

The ITL operator chop can be used to express the sequential composition of two temporal formulas (Moszkowski, 1986). Consider the system depicted in Figure 5.6.2-1 containing three states ($s_0$, $s_1$, and $s_2$) and two transitions linking the three states.

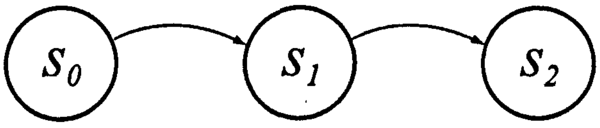


Figure 5.6.2-1: Three-State State Transition Diagram

Three state formulas, $w_0$, $w_1$, and $w_2$, are used to describe this system, where $s_0 \models w_0$, $s_1 \models w_1$, and $s_2 \models w_2$. The two state transitions included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

| | | |
|---|---|---|
| $s_0$ | $w_0 \wedge \circ w_1$ | (5.6.2-1a) |
| $s_1$ | $w_1 \wedge \circ w_2$ | (5.6.2-1b) |
| $s_2$ | – | |

Because these two rules share a state sequence satisfying $w_1$, the ITL operator chop can be used to sequentially compose the two rules. Therefore, the entire system is described as:

$$(w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_2) \tag{5.6.2-2}$$

The following lemma describes how two rules, constructed of state formulas and sequentially composed using the ITL operator chop, can be expressed as single general-form rule.

LEMMA: StateTwoChopRulesImp

$\vdash (w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_2)$ implies $\vdash w_0 \wedge \circ(w_1 ; \circ w_2)$

Proof:

| | | |
|---|---|---|
| 1 | $(w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_2)$ | premise |
| 2 | $w_0 \wedge (\circ w_1 ; (w_1 \wedge \circ w_2))$ | 1, ITL (StateAndChop) |
| 3 | $\circ w_1 ; (w_1 \wedge \circ w_2)$ | 2, $\wedge$elimination |
| 4 | $\circ w_1 ; w_1 \wedge \circ w_1 ; \circ w_2$ | 3, ITL (ChopAndImp) |
| 5 | $\circ w_1 ; \circ w_2$ | 4, $\wedge$elimination |
| 6 | $\circ(w_1 ; \circ w_2)$ | 5, ITL (NextChop) |
| 7 | $w_0$ | 2, $\wedge$elimination |
| 8 | $w_0 \wedge \circ(w_1 ; \circ w_2)$ | 5, 7, $\wedge$introduction |

Applying StateTwoChopRulesImp to (5.6.2-2) yields:

$$w_0 \wedge \circ(w_1 ; \circ w_2) \tag{5.6.2-3}$$

Using StateTwoChopRulesImp, two rules that have been sequentially composed using chop can be expressed as a single general-form rule, where $w_0$ specifies the rule condition and $w_1 ; \circ w_2$ specifies the rule state.

Applying NextChop to (5.6.2-3) and substituting the definition of the ITL next operator $\circ$ yields:

$$w_0 \wedge \mathsf{skip} ; w_1 ; \mathsf{skip} ; w_2 \tag{5.6.2-4}$$

In this form, the sequential nature of the composition of (5.6.2-1a) and (5.6.2-1b) using chop is clear.

As previously discussed, a state formula is satisfied by a single state or the first state in a multi-state sequence. Therefore, a second state formula can be chopped to the second (or later) state of a given state sequence described by another state formula, thereby satisfying the semantic requirements of the ITL chop operator that the two chopped intervals share a common state. Therefore, the sequential composition model can be expanded so that two total rules that do not share a common state formula, and therefore common initial states, can be sequentially composed using chop.

Consider the following sequential composition using the ITL chop operator of two general-form rules that do not share a temporal formula:

$$(w_0 \wedge \circ w_1) \, ; (w_2 \wedge \circ w_3) \tag{5.6.2-5}$$

Applying StateAndNextChop to (5.6.2-5) yields the following equivalent form:

$$w_0 \wedge \circ(w_1 \, ; (w_2 \wedge \circ w_3)) \tag{5.6.2-6}$$

With this equivalence transformation, the two chopped rules of (5.6.2-5) have been transformed into a general-form rule that includes a chopped and nested rule in the rule state. An alternative transformation of two chopped rules is described in the following lemma.

LEMMA: StateTwoChopRulesImp2

$\vdash (w_0 \wedge \circ f_1) \, ; (w_2 \wedge \circ f_3)$ implies $\vdash w_0 \wedge (\circ f_1 \, ; w_2) \wedge (\circ f_1 \, ; \circ f_3)$

Proof:

| | | |
|---|---|---|
| 1 | $(w_0 \wedge \circ f_1) \, ; (w_2 \wedge \circ f_3)$ | premise |
| 2 | $w_0 \wedge (\circ f_1 \, ; (w_2 \wedge \circ f_3))$ | 1, ITL (StateAndChop) |
| 3 | $\circ f_1 \, ; (w_2 \wedge \circ f_3)$ | 2, $\wedge$elimination |
| 4 | $(\circ f_1 \, ; w_2) \wedge (\circ f_1 \, ; \circ f_3)$ | 3, ITL (ChopAndImp) |
| 5 | $w_0$ | 2, $\wedge$elimination |
| 6 | $w_0 \wedge (\circ f_1 \, ; w_2) \wedge (\circ f_1 \, ; \circ f_3)$ | 4, 5, $\wedge$introduction |

Applying StateTwoChopRulesImp2 to (5.6.2-5) yields:

$$w_0 \wedge (\circ w_1 \, ; w_2) \wedge (\circ w_1 \, ; \circ w_3) \tag{5.6.2-7}$$

In this form, the sequential aspects of (5.6.2-6) are evident, because (5.6.2-7) is a conjunction of three state sequences. And as a conjunctive structure, (5.6.2-7) can manipulated with propositional logic to eliminate conjuncts as necessary to achieve the desired final form. Stated another way, either $w_0 \wedge (\circ f_1 \,;\, w_2)$ or $w_0 \wedge (\circ f_1 \,;\, \circ f_3)$ can be concluded from StateTwoChopRulesImp2.

Another alternative transformation of two chopped rules is described in the following lemma.

LEMMA: StateTwoChopRulesImp3

$\vdash (w_0 \wedge \circ f_1) \,;\, (w_2 \wedge \circ f_3)$ implies $\vdash w_0 \,;\, w_2 \wedge \circ f_1 \,;\, \circ f_3$

Proof:

| | | |
|---|---|---|
| 1 | $(w_0 \wedge \circ f_1) \,;\, (w_2 \wedge \circ f_3)$ | premise |
| 2 | $((w_0 \wedge \circ f_1) \,;\, w_2) \wedge ((w_0 \wedge \circ f_1) \,;\, \circ f_3)$ | 1, ChopAndImp |
| 3 | $(w_0 \wedge \circ f_1) \,;\, w_2$ | 2, $\wedge$ elimination |
| 4 | $w_0 \,;\, w_2 \wedge \circ f_1 \,;\, w_2$ | 3, AndChopImp |
| 5 | $(w_0 \wedge \circ f_1) \,;\, \circ f_3$ | 2, $\wedge$ elimination |
| 6 | $w_0 \,;\, \circ f_3 \wedge \circ f_1 \,;\, \circ f_3$ | 5, AndChopImp |
| 7 | $w_0 \,;\, w_2 \wedge \circ f_1 \,;\, w_2 \wedge w_0 \,;\, \circ f_3 \wedge \circ f_1 \,;\, \circ f_3$ | 4, 6, $\wedge$ introduction |
| 8 | $w_0 \,;\, w_2 \wedge \circ f_1 \,;\, \circ f_3$ | 7, $\wedge$ elimination |

Applying StateTwoChopRulesImp3 to (5.6.2-5) yields:

$$w_0 \,;\, w_2 \wedge \circ w_1 \,;\, \circ w_3 \tag{5.6.2-8}$$

The sequential composition of two total rules is described in the following three lemmas.

LEMMA: TwoTotalRulesChopEqv1

$$\begin{aligned}\vdash\ & ((w_{a_0} \wedge \circ f_{a_1}) \vee (\neg w_{a_0} \wedge \circ f_{a_2})) \,;\, ((w_{b_0} \wedge \circ f_{b_1}) \vee (\neg w_{b_0} \wedge \circ f_{b_2})) \\ \equiv\ & (w_{a_0} \wedge \circ f_{a_1}) \,;\, (w_{b_0} \wedge \circ f_{b_1}) \vee (w_{a_0} \wedge \circ f_{a_1}) \,;\, (\neg w_{b_0} \wedge \circ f_{b_2}) \\ & \vee (\neg w_{a_0} \wedge \circ f_{a_2}) \,;\, (w_{b_0} \wedge \circ f_{b_1}) \vee (\neg w_{a_0} \wedge \circ f_{a_2}) \,;\, (\neg w_{b_0} \wedge \circ f_{b_2})\end{aligned}$$

or

$$\begin{aligned}\vdash\ & (rule_{a_{true}} \vee rule_{a_{false}}) \,;\, (rule_{b_{true}} \vee rule_{b_{false}}) \\ \equiv\ & (rule_{a_{true}} \,;\, rule_{b_{true}}) \vee (rule_{a_{true}} \,;\, rule_{b_{false}}) \vee (rule_{a_{false}} \,;\, rule_{b_{true}}) \vee (rule_{a_{false}} \,;\, rule_{b_{false}})\end{aligned}$$

where: $rule_{a_{true}} \triangleq (w_{a_0} \wedge \circ f_{a_1})$

$$rule_{a_{false}} \triangleq (\neg w_{a_0} \wedge \bigcirc f_{a_2})$$
$$rule_{b_{true}} \triangleq (w_{b_0} \wedge \bigcirc f_{b_1})$$
$$rule_{b_{false}} \triangleq (\neg w_{b_0} \wedge \bigcirc f_{b_2})$$

Proof:

| | | |
|---|---|---|
| 1 | $((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}))\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | tautology |
| | $\equiv ((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}))\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | |
| 2 | $\equiv (w_{a_0} \wedge \bigcirc f_{a_1})\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | 1, ITL |
| | $\vee\ (\neg w_{a_0} \wedge \bigcirc f_{a_2})\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | (OrChopEqv) |
| 3 | $\equiv (w_{a_0} \wedge \bigcirc f_{a_1})\ ;\ (w_{b_0} \wedge \bigcirc f_{b_1}) \vee (w_{a_0} \wedge \bigcirc f_{a_1})\ ;\ (\neg w_{b_0} \wedge \bigcirc f_{b_2})$ | 2, ITL |
| | $\vee\ (\neg w_{a_0} \wedge \bigcirc f_{a_2})\ ;\ (w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2})\ ;\ (\neg w_{b_0} \wedge \bigcirc f_{b_2})$ | (ChopOrEqv) |

LEMMA: TwoTotalRulesChopEqv2

$$\vdash ((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}))\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$$
$$\equiv\ w_{a_0} \wedge \bigcirc(f_{a_1}\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2})))$$
$$\vee\ \neg w_{a_0} \wedge \bigcirc(f_{a_2}\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2})))$$

or

$$\vdash (rule_{a_{true}} \vee rule_{a_{false}})\ ;\ (rule_{b_{true}} \vee rule_{b_{false}})$$
$$\equiv\ (w_{a_0} \wedge \bigcirc(f_{a_1}\ ;\ (rule_{b_{true}} \vee rule_{a_{false}})))\ \vee\ (\neg w_{a_0} \wedge \bigcirc(f_{a_2}\ ;\ (rule_{b_{true}} \vee rule_{b_{false}})))$$

where:

$$rule_{a_{true}} \triangleq (w_{a_0} \wedge \bigcirc f_{a_1})$$
$$rule_{a_{false}} \triangleq (\neg w_{a_0} \wedge \bigcirc f_{a_2})$$
$$rule_{b_{true}} \triangleq (w_{b_0} \wedge \bigcirc f_{b_1})$$
$$rule_{b_{false}} \triangleq (\neg w_{b_0} \wedge \bigcirc f_{b_2})$$

Proof:

| | | |
|---|---|---|
| 1 | $((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}))\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | tautology |
| | $\equiv ((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}))\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | |
| 2 | $\equiv (w_{a_0} \wedge \bigcirc f_{a_1})\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | 1, ITL |
| | $\vee\ (\neg w_{a_0} \wedge \bigcirc f_{a_2})\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | (OrChopEqv) |
| 3 | $\equiv w_{a_0} \wedge \bigcirc(f_{a_1}\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2})))$ | 2, ITL (State- |
| | $\vee\ \neg w_{a_0} \wedge \bigcirc(f_{a_2}\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2})))$ | AndNextChop) |

LEMMA: TwoTotalRulesChopEqv3

$$\vdash ((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}))\ ;\ ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$$

$$\begin{aligned}\equiv\ & (w_{a_0} \wedge \bigcirc(f_{a_1} ; (w_{b_0} \wedge \bigcirc f_{b_1})))\\ & \vee (w_{a_0} \wedge \bigcirc(f_{a_1} ; (\neg w_{b_0} \wedge \bigcirc f_{b_2})))\\ & \vee (\neg w_{a_0} \wedge \bigcirc(f_{a_2} ; (w_{b_0} \wedge \bigcirc f_{b_1})))\\ & \vee (\neg w_{a_0} \wedge \bigcirc(f_{a_2} ; (\neg w_{b_0} \wedge \bigcirc f_{b_2})))\end{aligned}$$

or

$$\vdash (rule_{a_{true}} \vee rule_{a_{false}}) ; (rule_{b_{true}} \vee rule_{a_{false}})$$

$$\begin{aligned}\equiv\ & (w_{a_0} \wedge \bigcirc(f_{a_1} ; rule_{b_{true}}))\\ & \vee (w_{a_0} \wedge \bigcirc(f_{a_1} ; rule_{b_{false}}))\\ & \vee (\neg w_{a_0} \wedge \bigcirc(f_{a_2} ; rule_{b_{true}}))\\ & \vee (\neg w_{a_0} \wedge \bigcirc(f_{a_2} ; rule_{b_{false}}))\end{aligned}$$

where:

$$\begin{aligned} rule_{a_{true}} &\triangleq (w_{a_0} \wedge \bigcirc f_{a_1})\\ rule_{a_{false}} &\triangleq (\neg w_{a_0} \wedge \bigcirc f_{a_2})\\ rule_{b_{true}} &\triangleq (w_{b_0} \wedge \bigcirc f_{b_1})\\ rule_{b_{false}} &\triangleq (\neg w_{b_0} \wedge \bigcirc f_{b_2})\end{aligned}$$

Proof:

| | | |
|---|---|---|
| 1 | $((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2})) ; ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ $\equiv ((w_{a_0} \wedge \bigcirc f_{a_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2})) ; ((w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{b_0} \wedge \bigcirc f_{b_2}))$ | tautology |
| 2 | $\equiv (w_{a_0} \wedge \bigcirc f_{a_1}) ; (w_{b_0} \wedge \bigcirc f_{b_1}) \vee (w_{a_0} \wedge \bigcirc f_{a_1}) ; (\neg w_{b_0} \wedge \bigcirc f_{b_2})$ $\vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}) ; (w_{b_0} \wedge \bigcirc f_{b_1}) \vee (\neg w_{a_0} \wedge \bigcirc f_{a_2}) ; (\neg w_{b_0} \wedge \bigcirc f_{b_2})$ | 1, TwoTotal-RulesChopEqv1 |
| 3 | $\equiv (w_{a_0} \wedge \bigcirc(f_{a_1} ; (w_{b_0} \wedge \bigcirc f_{b_1}))) \vee (w_{a_0} \wedge \bigcirc(f_{a_1} ; (\neg w_{b_0} \wedge \bigcirc f_{b_2})))$ $\vee (\neg w_{a_0} \wedge \bigcirc(f_{a_2} ; (w_{b_0} \wedge \bigcirc f_{b_1}))) \vee (\neg w_{a_0} \wedge \bigcirc(f_{a_2} ; (\neg w_{b_0} \wedge \bigcirc f_{b_2})))$ | 2, ITL (State-AndNextChop) |

These lemmas are also expressed in terms of specific rule definitions to simplify presentation and highlight the underlying rule structure(s). With TwoTotalRules-ChopEqv1, two chopped total rules are decomposed to an equivalent disjunction of four chopped individual rules describing the possible state sequence associated with the original sequential composition. The individual rules that compose these four disjuncts follow the typical two-by-two truth matrix pattern – true-true, true-false, false-true, or false-false – reflecting the satisfaction or non-satisfaction of the rule conditions associated with the two total rules used in the original sequential composition. With TwoTotalRulesChopEqv2, the original chopped sequential composition of two total rules is transformed into a disjunction of two general-form rules. With TwoTotalRulesChopEqv3, the original chopped sequential composition of two total rules is transformed into a disjunction of four general-form rules. Together, substantial flexibility is provided with these three lemmas in transforming the chopped composition of two total rules.

## 5.7 Reflexive and Irreflexive Rules

The concepts of reflexivity and irreflexivity as used in relations are extended to describe the attributes of reflexive and irreflexive rules.

### 5.7.1 Reflexive Rules

A rule is reflexive if every state sequence in the rule domain is related to itself. Formally, a rule *rule* is reflexive if:

$$\forall\, \sigma \in domain_{rule}\ \exists\, \sigma \in codomain_{rule}\ \mid \sigma = \delta_{rule}(\sigma) \qquad (5.7.1\text{-}1)$$

Implicit in this definition is that, for a given rule *rule*, all elements of the rule domain are contained in the rule codomain, or $domain_{rule} \subseteq codomain_{rule}$.

The simplest possible reflexive rule system can be built around a one-state system. Consider the one-state reflexive system presented in Figure 5.7.1-1, containing one state transition.

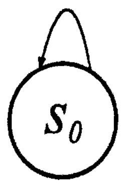


Figure 5.7.1-1: One-State Reflexive System

In this system, $s_0 \models w_0$. The one transition included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

$$s_0 \qquad w_0 \wedge \circ w_0 \qquad (5.7.1\text{-}2)$$

Graphically, a reflexive rule is represented by a loop from a state to itself, given that the rule state must include the rule condition state.

As a demonstrative exercise, the rule at (5.7.1-1) can be sequentially composed with itself using the general rule form $f_i \wedge \circ f_j$ as a basis for sequential composition. Instantiating both $f_i$ and $f_j$ with $w_0 \wedge \circ w_0$ yields:

$$(w_0 \wedge \circ w_0) \wedge \circ(w_0 \wedge \circ w_0) \tag{5.7.1-3}$$

Applying TwoSeqRulesEqv to (5.7.1-3) yields the following equivalent rule expressed in general rule form:

$$w_0 \wedge \circ(w_0 \wedge \circ w_0) \tag{5.7.1-4}$$

Applying NextAndDistEqv to (5.7.1-4) yields the following equivalent conjunctive description of the state sequence associated with sequentially composing (5.7.1-2) with itself:

$$w_0 \wedge \circ w_0 \wedge \circ\circ w_0 \tag{5.7.1-5}$$

Consider the two-state reflexive system presented in Figure 5.7.1-2, containing two state transitions:

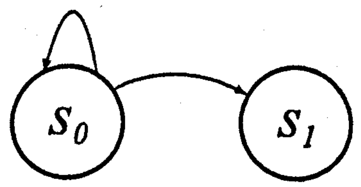


Figure 5.7.1-2: Two-State Reflexive System

In this system, $s_0 \models w_0$ and $s_1 \models w_1$. It is noted that this is a special case of that described in (5.1-6). The two transitions included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

| | | |
|---|---|---|
| $s_0$ | $w_0 \wedge \circ w_0$ | (5.7.1-6a) |
| $s_0$ | $w_0 \wedge \circ w_1$ | (5.7.1-6b) |
| $s_1$ | – | |

Because all states in the rule domain exist in the rule codomain and a relation exists between all members of the rule domain, described as $w_0 \wedge \circ w_0$, and at least one member of the rule codomain, the formal requirements for a reflexive system are met.

As identified in (5.7.1-6a) and (5.7.1-6b), both transitions in this system are satisfied by state sequences that begin with $s_0$. These transitions for which $s_0$ satisfies the rule condition can be combined disjunctively as:

$$(w_0 \wedge \circ w_0) \vee (w_0 \wedge \circ w_1) \qquad (5.7.1\text{-}7)$$

Applying CommonRuleCondEqv to (5.7.1-5) yields:

$$w_0 \wedge \circ(w_0 \vee w_1) \qquad (5.7.1\text{-}8)$$

The application of CommonRuleCondEqv allows the two-rule rule system of (5.7.1-6a) and (5.7.1-6b), disjunctively associated in (5.7.1-7), to be expressed as a single general-form rule. As highlighted with this transformation, this two-state reflexive system is a simple example of a non-deterministic system expressible as a single rule. The satisfaction of the rule condition $w_0$ is associated with two alternative future state sequences – either a state sequence satisfying $w_0$ or a state sequence satisfying $w_1$.

## 5.7.2 Irreflexive Rules

In an irreflexive rule, no state in the rule domain can be directly related to itself. Formally, a rule *rule* is irreflexive if:

$$\forall\, \sigma \in domain_{rule} \wedge \forall\, \sigma' \in codomain_{rule} \mid \sigma' = \delta_{rule}(\sigma) \supset \sigma \neq \sigma' \qquad (5.7.2\text{-}1)$$

This definition supports the informal view that a state sequence satisfying the rule condition cannot also satisfy the rule state. Graphically, an irreflexive rule cannot include a loop from a state to itself.

Consider the two-state irreflexive system presented in Figure 5.7.2-1, containing one state transition:

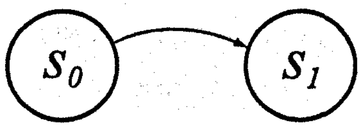


Figure 5.7.2-1: Two-State Irreflexive System

In this system, $s_0 \models w_0$, and $s_1 \models w_1$. The one transition included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

$$\begin{array}{ll} s_0 & w_0 \wedge \circ w_1 \\ s_1 & - \end{array} \qquad (5.7.2\text{-}2)$$

Because this two-state irreflexive system contains only one state transition, the entire system is described as:

$$w_0 \wedge \circ w_1 \qquad (5.7.2\text{-}3)$$

This two-state irreflexive system is the simplest possible two-state system, because it contains only one state transition and therefore is described by a single general-form rule without manipulation.

## 5.8 Symmetric, Antisymmetric, and Asymmetric Rules

The concepts of symmetry, antisymmetry, and asymmetry as used in relations are extended to describe the attributes of symmetric, antisymmetric, and asymmetric rules.

### 5.8.1 Symmetric Rules

A rule is symmetric if it is its own inverse. For a rule to be symmetric, whenever that rule includes a transition from $\sigma$ to $\sigma'$, that rule must also include a transition from $\sigma'$ to $\sigma$. Formally, a rule *rule* is symmetric if:

$$\forall \sigma, \sigma' \in \mathit{universe}_{rule} \mid \sigma' = \delta_{rule}(\sigma) \supset \sigma = \delta_{rule}(\sigma') \qquad (5.8.1\text{-}1)$$

Because there is no requirement in (5.8.1-1) that $\sigma$ and $\sigma'$ be unique, the simplest possible symmetric rule involves a one-state system. In a one-state symmetric system consisting of $s_0$, $s_0 \models w_0$. The single state transition in this system, relating $s_0$ to itself, is described by the rule:

$$w_0 \wedge \circ w_0 \qquad (5.8.1\text{-}2)$$

This one-state symmetric system is also a one-state reflexive system, previously described in Section 5.7.1.

Consider the two-state symmetric system presented in Figure 5.8.1-1, containing two state transitions:

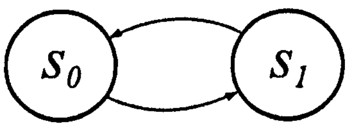


Figure 5.8.1-1: Two-State Symmetric System

In this system, $s_0 \neq s_1$, $s_0 \models w_0$, and $s_1 \models w_1$. The two transitions included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

| | | |
|---|---|---|
| $s_0$ | $w_0 \wedge \circ w_1$ | (5.8.1-3a) |
| $s_1$ | $w_1 \wedge \circ w_0$ | (5.8.1-3b) |

Using these two individual state transitions, the entire system is described disjunctively as:

$$(w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0) \qquad (5.8.1\text{-}4)$$

Both the individual rules of (5.8.1-3a) and (5.8.1-3a), and the disjunctive system description of (5.8.1-4) are used in the following paragraphs to describe the behavior of this two-state symmetric system.

Because the two rules, (5.8.1-3a) and (5.8.1-3b), describing the state transitions for this system share a state (i.e., because the rule state specified by one rule is the rule condition of the other rule), the general rule form $f_i \wedge \circ f_j$ can be applied to sequentially compose these two rules to form a new rule that describes a state sequence associated with this system. If the system is assumed to be in $s_0$ and using (5.8.1-3a) to express the rule condition and (5.8.1-3b) to express the rule state, the following rule describes this sequential association of the two rules:

$$(w_0 \wedge \circ w_1) \wedge \circ(w_1 \wedge \circ w_0) \tag{5.8.1-5}$$

TwoSeqRulesEqv2 is applied to (5.8.1-5) to obtain the following equivalent rule:

$$w_0 \wedge \circ(w_1 \wedge \circ w_0) \tag{5.8.1-6}$$

Either TwoSeqRulesEqv1 can be applied to (5.8.1-5) or NextAndDistEqv can be applied to (5.8.1-6) to obtain the following equivalent conjunctive description of the state sequence associated with this rule:

$$w_0 \wedge \circ w_1 \wedge \circ\circ w_0 \tag{5.8.1-7}$$

Alternatively, if the system is assumed to be in $s_1$ and using (5.8.1-3b) to express the rule condition and (5.8.1-3a) to express the rule state, the following rule describes an alternative sequential association of the two rules:

$$(w_1 \wedge \circ w_0) \wedge \circ(w_0 \wedge \circ w_1) \tag{5.8.1-8}$$

TwoSeqRulesEqv2 is applied to (5.8.1-8) to obtain the following equivalent rule:

$$w_1 \wedge \circ(w_0 \wedge \circ w_1) \tag{5.8.1-9}$$

Either TwoSeqRulesEqv1 can be applied to (5.8.1-8) or NextAndDistEqv can be applied to (5.8.1-9) to obtain the following equivalent conjunctive description of the state sequence associated with this rule:

$$w_1 \wedge \circ w_0 \wedge \circ\circ w_1 \qquad (5.8.1\text{-}10)$$

Both $w_0 \wedge \circ w_1 \wedge \circ\circ w_0$ and $w_1 \wedge \circ w_0 \wedge \circ\circ w_1$ describe state sequences associated with this symmetric system, depending on the initial system state such that either $w_0$ or $w_1$ holds.

The chop operator can be used to compose the two individual rules describing this two-state symmetric system. Assuming that the system is in $s_0$, (5.8.1-3a) and (5.8.1-3b) can be sequentially composed, in that order, using chop and the resulting state sequence is described as:

$$(w_0 \wedge \circ w_1) \; ; (w_1 \wedge \circ w_0) \qquad (5.8.1\text{-}11)$$

StateTwoChopRulesImp and NextChop are applied to (5.8.1-11) to yield:

$$w_0 \wedge \circ w_1 \; ; \circ w_0 \qquad (5.8.1\text{-}12)$$

With these transformations, the two chopped rules of (5.8.1-11) have been transformed into one general-form rule incorporating chop only in the specification of the rule state.

Alternatively, if the system is assumed to be in $s_1$, (5.8.1-3b) and (5.8.1-3a) can be sequentially composed, in that order, using chop and the resulting the state sequence is described as:

$$(w_1 \wedge \circ w_0) \; ; (w_0 \wedge \circ w_1) \qquad (5.8.1\text{-}13)$$

As before, StateTwoChopRulesImp and NextChop are applied to (5.8.1-13) to yield:

$$w_1 \wedge \circ w_0 \; ; \circ w_1 \qquad (5.8.1\text{-}14)$$

As before, the two chopped rules of (5.8.1-13) have been transformed into one general-form rule incorporating chop only in the specification of the rule state.

Although both $w_0 \wedge \circ w_1 ; \circ w_0$ and $w_1 \wedge \circ w_0 ; \circ w_1$ describe state sequences that may result from this symmetric system, depending on the system state at rule execution, the above analyses have assumed an initial state condition to limit the number of possible cases and facilitate analysis. Now, consider the following case, where the total description of this system, previously presented in (5.8.1-4), is used to describe both possible cases. By using the total description of the symmetric system, no knowledge, specification, or assumption of the initial system state is required. In this analysis, this total description is composed with itself using chop and the resulting state sequence is described as:

$$((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) ; ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) \qquad (5.8.1\text{-}15)$$

This sequential composition can be expanded, as described in the lemma TwoSymRulesChop presented below, to represent the possible state sequences.

LEMMA: TwoSymRulesChop

$$\begin{aligned} \vdash\ & ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) ; ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) \\ \equiv\ & (w_0 \wedge \circ w_1) ; (w_0 \wedge \circ w_1) \vee (w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_0) \\ & \vee (w_1 \wedge \circ w_0) ; (w_1 \wedge \circ w_0) \vee (w_1 \wedge \circ w_0) ; (w_0 \wedge \circ w_1) \end{aligned}$$

or

$$\begin{aligned} \vdash\ & (rule_0 \vee rule_1) ; (rule_0 \vee rule_1) \\ \equiv\ & rule_0 ; rule_0 \ \vee\ rule_0 ; rule_1 \ \vee\ rule_1 ; rule_1 \ \vee\ rule_1 ; rule_0 \end{aligned}$$

where: $rule_0 \equiv (w_0 \wedge \circ w_1)$
$rule_1 \equiv (w_1 \wedge \circ w_0)$

Proof:

| | | |
|---|---|---|
| 1 | $((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) ; ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0))$ $\equiv ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) ; ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0))$ | tautology |
| 2 | $\equiv ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) ; (w_0 \wedge \circ w_1)$ $\vee ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) ; (w_1 \wedge \circ w_0))$ | 1, ITL (ChopOrEqv) |

3 $\equiv (w_0 \wedge \circ w_1) ; (w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0) ; (w_0 \wedge \circ w_1)$ 2, ITL (OrChopEqv)
$\vee (w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_0) \vee (w_1 \wedge \circ w_0) ; (w_1 \wedge \circ w_0)$

4 $\equiv (w_0 \wedge \circ w_1) ; (w_0 \wedge \circ w_1)$ 3, commutivity of $\vee$
$\vee (w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_0)$
$\vee (w_1 \wedge \circ w_0) ; (w_1 \wedge \circ w_0)$
$\vee (w_1 \wedge \circ w_0) ; (w_0 \wedge \circ w_1)$

This lemma is also expressed in terms of specific rule definitions to simplify presentation and highlight the underlying rule structure(s).

Applying TwoSymRulesChop to (5.8.1-15) yields:

$$\begin{aligned}&(w_0 \wedge \circ w_1) ; (w_0 \wedge \circ w_1)\\ &\vee (w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_0)\\ &\vee (w_1 \wedge \circ w_0) ; (w_1 \wedge \circ w_0)\\ &\vee (w_1 \wedge \circ w_0) ; (w_0 \wedge \circ w_1)\end{aligned} \qquad (5.8.1\text{-}16)$$

With the application of TwoSymRulesChop, (5.8.1-15) is expanded and the four possible state sequences resulting from the sequential composition of the rule (5.8.1-4) with itself using chop are enumerated in the equivalent form of (5.8.1-16). The four state sequences satisfying (5.8.1-16) are presented in Figure 5.8.1-1. These four possible state sequences are organized based on the state that satisfies the rule condition of the initial rule of the sequential composition.

By inspection of the four state sequences presented in Figure 5.8.1-2, the state sequence $s_0s_1s_0$ depicted in Figure 5.8.1-2b is subsumed by the state sequence $s_0s_1s_0s_1$ depicted in Figure 5.8.1-2a. Similarly, the state sequence $s_1s_0s_1$ depicted in Figure 5.8.1-2d is subsumed by the state sequence $s_1s_0s_1s_0$ depicted in Figure 5.8.1-2c. These sequences are also consistent with the rules (5.8.1-12) and (5.8.1-14) derived based on chopping the individual rules. These sequences of alternative states are consistent with both the formal definition of a symmetric system, as presented at (5.8.1-1), and an intuitive expectation of the behavior of a two-state symmetric (and irreflexive) system. Using the complete rule-based description of this system presented in (5.8.1-4), the ITL operator chop can be used to sequentially compose the rule-based structure (5.8.1-15) that describes the possible state sequences associated with such a symmetric state

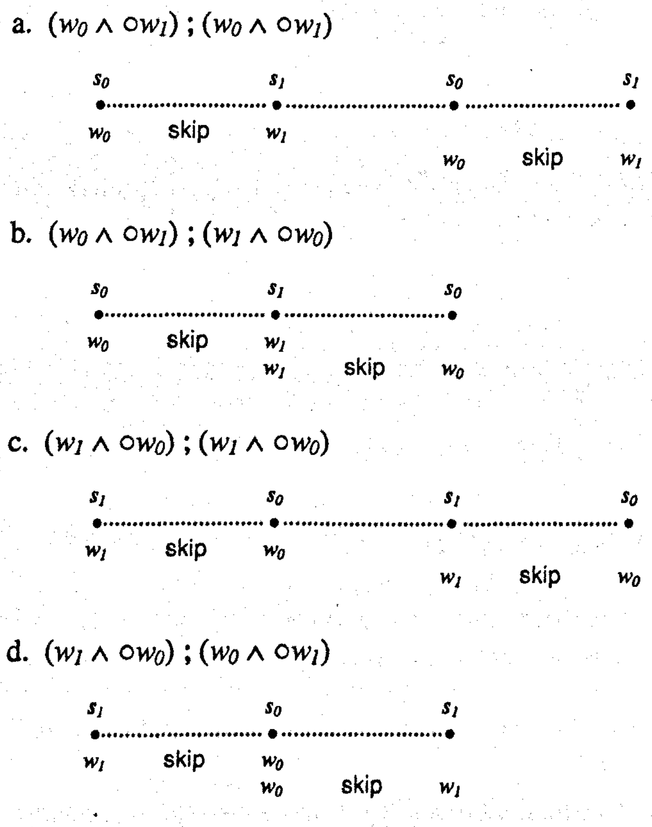

a. $(w_0 \wedge \circ w_1) ; (w_0 \wedge \circ w_1)$
$s_0$ $s_1$ $s_0$ $s_1$
$w_0$ skip $w_1$
$w_0$ skip $w_1$
b. $(w_0 \wedge \circ w_1) ; (w_1 \wedge \circ w_0)$
$s_0$ $s_1$ $s_0$
$w_0$ skip $w_1$
$w_1$ skip $w_0$
c. $(w_1 \wedge \circ w_0) ; (w_1 \wedge \circ w_0)$
$s_1$ $s_0$ $s_1$ $s_0$
$w_1$ skip $w_0$
$w_1$ skip $w_0$
d. $(w_1 \wedge \circ w_0) ; (w_0 \wedge \circ w_1)$
$s_1$ $s_0$ $s_1$
$w_1$ skip $w_0$
$w_0$ skip $w_1$

Figure 5.8.1-2: State Sequences Resulting from Sequentially Composing with chop both Rules Describing a Two-State Symmetric System

system. If the system state at the time of rule execution is known, the specific state sequence can be determined.

Alternatively, the general rule form $f_i \wedge \circ f_j$ can be applied using the complete description of this system, previously presented in (5.8.1-4), to describe the state sequence(s) that will result from the sequential composition of rule (5.8.1-4) with itself. Again, by using the complete description of the system, no knowledge, specification, or assumption of the initial system state is required. Using the general rule form $f_i \wedge \circ f_j$, this rule-form sequential composition is as follows:

$$((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) \wedge \circ((w_0 \wedge \circ w_1) \vee (w_1 \wedge \circ w_0)) \qquad (5.8.1\text{-}17)$$

This rule-form composition can be expanded, as described in the lemma TwoSymRulesAsRule presented below, to identify the possible state sequences that may result from such a sequential composition. The restriction that $f_0 \equiv \neg f_1$ has been added to this lemma to facilitate analysis and preclude the application of this lemma to a one-state system.

LEMMA: TwoSymRulesAsRule

$\vdash ((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0)) \wedge \circ((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0))$ and $\vdash f_0 \equiv \neg f_1$ implies
$\vdash (f_0 \wedge \circ f_1 \wedge \circ\circ f_0) \vee (f_1 \wedge \circ f_0 \wedge \circ\circ f_1)$

Proof:

| | | |
|---|---|---|
| 1 | $((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0)) \wedge \circ((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0))$ | premise |
| 2 | $f_0 \equiv \neg f_1$ | premise |
| 3 | $((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0))$ $\wedge (\circ(f_0 \wedge \circ f_1) \vee \circ(f_1 \wedge \circ f_0))$ | 1, ITL (ChopOrEqv) |
| 4 | $((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0))$ $\wedge ((\circ f_0 \wedge \circ\circ f_1) \vee (\circ f_1 \wedge \circ\circ f_0))$ | 3, NextAndDistEqv |
| 5 | $(f_0 \wedge \circ f_1 \wedge \circ f_0 \wedge \circ\circ f_1) \vee (f_0 \wedge \circ f_1 \wedge \circ f_1 \wedge \circ\circ f_0)$ $\vee (f_1 \wedge \circ f_0 \wedge \circ f_0 \wedge \circ\circ f_1) \vee (f_1 \wedge \circ f_0 \wedge \circ f_1 \wedge \circ\circ f_0)$ | 4, distribution of $\wedge$ over $\vee$ (three times) |
| 6 | $(f_0 \wedge \circ f_1 \wedge \circ f_0 \wedge \circ\circ f_1) \vee (f_0 \wedge \circ f_1 \wedge \circ\circ f_0)$ $\vee (f_1 \wedge \circ f_0 \wedge \circ\circ f_1) \vee (f_1 \wedge \circ f_0 \wedge \circ f_1 \wedge \circ\circ f_0)$ | 5, idempotence of $\wedge$ |
| 7 | $(f_0 \wedge \circ f_1 \wedge \circ\neg f_1 \wedge \circ\circ f_1) \vee (f_0 \wedge \circ f_1 \wedge \circ\circ f_0)$ $\vee (f_1 \wedge \circ f_0 \wedge \circ\circ f_1) \vee (f_1 \wedge \circ\neg f_1 \wedge \circ f_1 \wedge \circ\circ f_0)$ | 2, 6, equivalence substitution |
| 8 | $(f_0 \wedge \mathit{false} \wedge \circ\circ f_1) \vee (f_0 \wedge \circ f_1 \wedge \circ\circ f_0)$ $\vee (f_1 \wedge \circ f_0 \wedge \circ\circ f_1) \vee (f_1 \wedge \mathit{false} \wedge \circ\circ f_0)$ | 7, TemporalContra |
| 9 | $\mathit{false} \vee (f_0 \wedge \circ f_1 \wedge \circ\circ f_0) \vee (f_1 \wedge \circ f_0 \wedge \circ\circ f_1) \vee \mathit{false}$ | 8, zero of $\wedge$ |
| 10 | $(f_0 \wedge \circ f_1 \wedge \circ\circ f_0) \vee (f_1 \wedge \circ f_0 \wedge \circ\circ f_1)$ | 9, unit of $\vee$ |

Applying TwoSymRulesAsRule to (5.8.1-17) yields:

$$(w_0 \wedge \circ w_1 \wedge \circ\circ w_0) \vee (w_1 \wedge \circ w_0 \wedge \circ\circ w_1) \qquad (5.8.1\text{-}18)$$

This transformation describes the two sequences of alternating states that result from the rule-form sequential composition presented in (5.8.1-17). The minimum state sequences satisfying (5.8.1-18) are $s_0s_1s_0$ or $s_1s_0s_1$. As with the chop-form sequential composition, these sequences of alternative states are consistent with both the formal definition of a symmetric system and an intuitive expectation of the behavior of such a system.

Using the general rule-form sequential composition presented in (5.8.1-17), an alternative outcome is possible, as demonstrated in TwoSymRulesAsRule2:

LEMMA: TwoSymRulesAsRule2

$\vdash ((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0)) \wedge \circ((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0))$ and $\vdash f_0 \equiv \neg f_1$ implies
$\vdash (f_0 \wedge \circ(f_1 \wedge \circ f_0)) \vee (f_1 \wedge \circ(f_0 \wedge \circ f_1))$
or
$\vdash (rule_0 \vee rule_1) \wedge \circ(rule_0 \vee rule_1)$ and $\vdash f_0 \equiv \neg f_1$ implies
$\vdash (f_0 \wedge \circ rule_1) \vee (f_1 \wedge \circ rule_0)$

where: $rule_0 \equiv (f_0 \wedge \circ f_1)$
$rule_1 \equiv (f_1 \wedge \circ f_0)$

Proof:

| | | |
|---|---|---|
| 1 | $((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0)) \wedge \circ((f_0 \wedge \circ f_1) \vee (f_1 \wedge \circ f_0))$ | premise |
| 2 | $f_0 \equiv \neg f_1$ | premise |
| 3 | $(f_0 \wedge \circ f_1 \wedge \circ\circ f_0) \vee (f_1 \wedge \circ f_0 \wedge \circ\circ f_1)$ | 1, 2, TwoSymRulesAsRule |
| 4 | $(f_0 \wedge \circ(f_1 \wedge \circ f_0)) \vee (f_1 \wedge \circ(f_0 \wedge \circ f_1))$ | 3, NextAndDistEqv |

This lemma is also expressed in terms of specific rule definitions to simplify presentation and highlight the underlying rule structure(s).

Applying TwoSymRulesAsRule2 to (5.8.1-17) yields:

$$(w_0 \wedge \circ(w_1 \wedge \circ w_0)) \vee (w_1 \wedge \circ(w_0 \wedge \circ w_1)) \quad (5.8.1\text{-}19)$$

With this transformation, (5.8.1-17) is transformed into a disjunction of two general-form rules. Alternatively, (5.8.1-19) is equivalent to (5.8.1-18) because it can be obtained directly from (5.8.1-18) by applying NextAndDistEqv.

Using the complete rule-based description of the two-state symmetric system presented in (5.8.1-4), the general rule form $f_i \wedge \circ f_j$ can be used to sequentially compose the rule-based structure (5.8.1-17) that describes the possible state sequences associated with such a symmetric state system. These possible sequences can be described either conjunctively or in general rule form. If the system state at the time of rule execution is known, the specific state sequence can be determined.

## 5.8.2 Asymmetric Rules

Informally, a rule is asymmetric if it is not its own inverse. For a rule to be asymmetric, whenever that rule includes a transition from σ to σ', that rule cannot also include the transition from σ' to σ. Formally, a rule *rule* is asymmetric if:

$$\forall \sigma, \sigma' \in universe_{rule} \mid \sigma' = \delta_{rule}(\sigma) \not\supset \sigma = \delta_{rule}(\sigma') \qquad (5.8.2\text{-}1)$$

Based on this formal definition, the simplest possible asymmetric system is a two-state system containing one state transition:

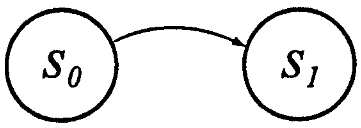


Figure 5.8.2-1: Two-State Asymmetric System

In this system, $s_0 \neq s_1$, $s_0 \models w_0$, and $s_1 \models w_1$. The one transition included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

$$\begin{array}{ll} s_0 & w_0 \wedge \circ w_1 \\ s_1 & - \end{array} \qquad (5.8.2\text{-}2)$$

Because this two-state asymmetric system contains only one state transition, the entire system is described as:

$$w_0 \wedge \circ w_1 \qquad (5.8.2\text{-}3)$$

This system is also irreflexive, as previously discussed in Section 5.7.2. The composition of two-state asymmetric rules into larger structures using either the general rule form $f_i \wedge \circ f_j$ or the ITL operator chop has been previously described in Sections 5.6.1 and 5.6.2, respectively.

Comparing the formal definition of rule symmetry presented in (5.8.1-1) with the formal definition of rule asymmetry presented in (5.8.2-1), a rule cannot be both symmetric and asymmetric, as symmetry requires that $\sigma' = \delta_{rule}(\sigma) \supset \sigma = \delta_{rule}(\sigma')$ and asymmetry requires that $\sigma' = \delta_{rule}(\sigma) \not\supset \sigma = \delta_{rule}(\sigma')$. However, a rule need not be either symmetric or asymmetric. Consider the following two-state system:

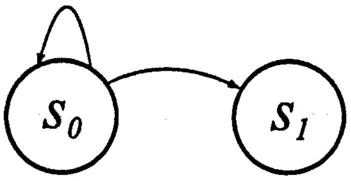


Figure 5.8.2-2: Two-State System that Is Neither Symmetric nor Asymmetric

In this system, $s_0 \neq s_1$, $s_0 \models w_0$, and $s_1 \models w_1$. The two transitions included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

| | | |
|---|---|---|
| $s_0$ | $w_0 \wedge \circ w_0$ | (5.8.2-4a) |
| $s_0$ | $w_0 \wedge \circ w_1$ | (5.8.2-4b) |
| $s_1$ | – | |

Using these two individual state transitions, the complete system is described disjunctively as:

$$(w_0 \wedge \circ w_1) \vee (w_0 \wedge \circ w_1) \qquad (5.8.2\text{-}5)$$

Propositional logic and NextAndDistEqv are applied to (5.8.2-5), and the system presented in Figure 5.8.2-2 is expressed as a single, equivalent general-form rule:

$$w_0 \wedge \circ(w_0 \vee w_1) \qquad (5.8.2\text{-}6)$$

Using the equivalent rule form (5.8.2-6) and considering the satisfaction relations bound to this system, the minimum rule domain is $\{s_0\}$ and the minimum rule codomain is $\{s_0, s_1\}$. Therefore, the rule universe for this system is $\{s_0, s_1\}$. Referencing the formal definition of rule symmetry presented at (5.8.1-1), there exists a transition from $s_0$ to $s_1$ (described by the rule $w_0 \wedge \circ w_1$) but no transition from $s_1$ to $s_0$ (and no corresponding rule $w_1 \wedge \circ w_0$). Therefore, the requirement for symmetry is not met. Referencing the formal definition of rule asymmetry presented at (5.8.2-1), because there exists a transition from $s_0$ to $s_0$ (described by the rule $w_0 \wedge \circ w_0$), the requirement for asymmetry that $\sigma' = \delta_{rule}(\sigma) \not\supset \sigma = \delta_{rule}(\sigma')$ is not met. Therefore, this system is neither symmetric nor asymmetric. However, this system is reflexive, as discussed in Section 5.7.1.

## 5.8.3 Antisymmetric Rules

For a rule to be antisymmetric, whenever that rule includes a transition from $\sigma$ to $\sigma'$ and a transition from $\sigma'$ to $\sigma$, $\sigma'$ and $\sigma$ must be equal. Formally, a rule *rule* is antisymmetric if:

$$\forall\ \sigma, \sigma' \in universe_{rule}\ \mid \sigma' = \delta_{rule}(\sigma) \wedge \sigma = \delta_{rule}(\sigma') \supset \sigma = \sigma' \qquad (5.8.3\text{-}1)$$

Using this formal definition, the simplest possible antisymmetric system involves only one state, $s_0$, and one state transition described by the rule $w_0 \wedge \circ w_0$ where $s_0 \models w_0$. This one-state rule system is also reflective and symmetric, as previously discussed in Section 5.7.1 and Section 5.8.1, respectively. As demonstrated with this case, a rule can be both symmetric and antisymmetric.

The simplest possible two-state antisymmetric system is the two-state asymmetric system presented in Figure 5.8.2-1. The minimum rule universe of that system is $\{s_0, s_1\}$. There is only one transition from $s_0$ to $s_1$ described by the rule $w_0 \wedge \circ w_1$ and $s_0 \neq s_1$. Therefore, both the antecedent $\sigma' = \delta_{rule}(\sigma) \wedge \sigma = \delta_{rule}(\sigma')$ and the consequent $\sigma = \sigma'$ of the definition at (5.8.3-1) are false. Therefore, the implication holds and requirements for antisymmetry is met. As demonstrated with this case, a rule system can be both asymmetric and antisymmetric.

However, an antisymmetric system need not be either symmetric or asymmetric. Consider the two-state system previously presented in Figure 5.8.2-2 that is neither symmetric nor asymmetric. The minimum rule universe of that system is $\{s_0, s_1\}$. There exists a transition from $s_0$ to $s_0$ described by the rule $w_0 \wedge \circ w_0$ and because $s_0 = s_0$, the requirement that $\sigma' = \delta_{rule}(\sigma) \wedge \sigma = \delta_{rule}(\sigma') \supset \sigma = \sigma'$ holds. There exists a transition from $s_0$ to $s_1$ described by the rule $w_0 \wedge \circ w_1$, but there is no transition from $s_1$ to $s_0$. Because $s_0 \neq s_1$, both sides of the required implication are false, and therefore the implication holds. Finally, there is no rule describing a transition from $s_1$ to $s_1$, but $s_1 = s_1$. Therefore, the implication is vacuously true and the requirement is met. Therefore, this system that is neither symmetric nor asymmetric is antisymmetric.

## 5.9 Transitive Rule Systems

Formally, a rule system *rs* is state transitive if:

$$\forall\ \sigma, \sigma', \sigma'' \in \mathit{universe}_{rs} \mid \sigma' = \delta_{rs}(\sigma) \wedge \sigma'' = \delta_{rs}(\sigma') \supset \sigma'' = \delta_{rs}(\sigma) \qquad (5.9\text{-}1)$$

where $\delta_{rs}$ represents any of the transition relations associated with rules comprising the rule system *rs*. Although one-state and two-state systems can be transitive, these are not addressed here. Consider the simple three-state symmetric system presented in Figure 5.9-1, containing three state transitions:

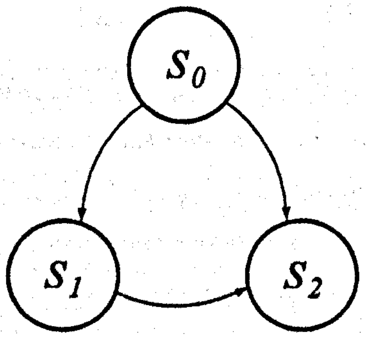


Figure 5.9-1: Three-State Transitive System

In this system, $s_0 \neq s_1$, $s_1 \neq s_2$, $s_0 \neq s_2$, $s_0 \models w_0$, $s_1 \models w_1$ and $s_2 \models w_2$. The three transitions included in this system can be described in rule form and organized based on the initial state in the state sequence satisfying the corresponding rule condition:

| | | |
|---|---|---|
| $s_0$ | $w_0 \wedge \circ w_1$ | (5.9-2a) |
| $s_0$ | $w_0 \wedge \circ w_2$ | (5.9-2b) |
| $s_1$ | $w_1 \wedge \circ w_2$ | (5.9-2c) |
| $s_2$ | – | |

The minimum rule universe for this set of rules is $\{s_0, s_1, s_2\}$. Based on this minimum rule universe and given the rule set identified in (5.9-1a), (5.9-1b), and (5.9-1c), both $\sigma' = \delta_{rs}(\sigma) \wedge \sigma'' = \delta_{rs}(\sigma')$ and $\sigma'' = \delta_{rs}(\sigma)$ holds. Therefore, this system is state transitive.

An important distinction is made here that this system must described as state transitive and not just transitive. Consider the two rules that share a state, (5.9-2a) and (5.9-2c). These two rules can be sequentially composed using the general rule form $f_i \wedge \circ f_j$ to describe the resulting state sequence:

$$(w_0 \wedge \circ w_1) \wedge \circ(w_1 \wedge \circ w_2) \tag{5.9-3}$$

TwoSeqRulesImp is applied to (5.9-3) to obtain the following state sequence:

$$w_0 \wedge \circ\circ w_2 \tag{5.9-4}$$

Comparing this inferred rule, $w_0 \wedge \circ\circ w_2$, with the native rule $w_0 \wedge \circ w_2$ at (5.9-2b), the sequential composition implemented in (5.9-3) requires one additional time step (i.e., one additional next) to reach the state satisfying $w_2$. Therefore, if rules (5.9-2b) and (5.9-4) are executed from a state satisfying $w_0$, the outcomes of those rule executions are $\circ w_2$ and $\circ\circ w_2$, respectively. Therefore, this system is described as state transitive but not temporally transitive. This simple example highlights the temporal aspects of rules and therefore one of the critical differences of the temporal logic approach to rules as presented in this thesis, as compared to simple representations of rules in non-temporal forms.

# Chapter 6

# Rule Algebra – Advanced Concepts

In this chapter, advanced concepts associated with the rule algebra are developed using the rule algebra fundamentals presented in Chapter 5. Additional compositional paradigms, including nesting, recursion, deterministic and non-deterministic guarded composition, and disjoint parallel composition, are presented. Alternative models of rule equivalence are discussed. Rule-based representations of typical legacy code structures – the if-then-else structure, the while structure, and the indexed for-loop – are developed.

## 6.1 Nesting

The general rule form $f_i \wedge \circ f_j$ is a temporal formula composed of two temporal formulas $f_i$ and $f_j$. Because either $f_i$ or $f_j$ can be instantiated with a rule, other rules can be nested within a general rule form $f_i \wedge \circ f_j$, which in turn can be nested within another rule. Such nesting can be the basis for rule encapsulation and program abstraction in the reverse engineering domain, or the basis for rule expansion and program refinement in the forward engineering domain. With nesting, numerous types of composite rules can be created. In this section, several configurations are examined, and previous rule formation models are reviewed within the context of nesting.

Consider the following nested rule, expressed in general rule form, where the rule condition is a rule:

$$(w_0 \wedge \circ w_1) \wedge \circ w_2 \qquad (6.1\text{-}1)$$

Expressed in this form, this rule conditions a property of the next state, described by $w_2$, on the concurrent satisfaction of another property in that same next state, described by $w_1$, and a property in the current state, described by $w_0$. The following lemma describes an equivalent alternative expression of this nested rule.

LEMMA: NestRuleCondEqv

$\vdash (f_0 \wedge \circ f_1) \wedge \circ f_2 \equiv f_0 \wedge \circ(f_1 \wedge f_2)$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge \circ f_1) \wedge \circ f_2 \equiv (f_0 \wedge \circ f_1) \wedge \circ f_2$ | tautology |
| 2 | $(f_0 \wedge \circ f_1) \wedge \circ f_2 \equiv f_0 \wedge (\circ f_1 \wedge \circ f_2)$ | 1, associativity of $\wedge$ |
| 3 | $(f_0 \wedge \circ f_1) \wedge \circ f_2 \equiv f_0 \wedge \circ(f_1 \wedge f_2)$ | 2, NextAndDistEqv |

Applying NestRuleCondEqv to (6.1-1) yields:

$$w_0 \wedge \circ(w_1 \wedge w_2) \tag{6.1-2}$$

In this equivalent form of (6.1-2), the logic of the original rule (6.1-1) is much more explicit – that both $w_1$ and $w_2$ must hold in the next state and therefore $w_1 \wedge w_2$ cannot be a contradiction. Although an acceptable form, nesting of rules within the rule condition must be done with great care, because the underlying rule logic may not be as transparent as other equivalent forms of rule construction.

Consider the following nested rule, expressed in general rule form, where the rule state is expressed as a rule:

$$w_0 \wedge \circ(w_1 \wedge \circ w_2) \tag{6.1-3}$$

Applying NextAndDistEqv, (6.1-3) is transformed to:

$$w_0 \wedge \circ w_1 \wedge \circ\circ w_2 \tag{6.1-4}$$

This equivalent form conjunctively describes the state sequence associated with the corresponding nested general-form rule (6.1-3). Because NextAndDistEqv is a logical equivalence, the reverse transformation strategy holds, as the conjunctive state sequence described in (6.1-4) can be transformed into the nested general-form rule (6.1-3).

Consider the following general-form rule which includes a rule nested in the rule condition and a rule nested in the rule state:

$$(w_0 \wedge \circ w_1) \wedge \circ(w_2 \wedge \circ w_3) \tag{6.1-5}$$

The rule nesting in (6.1-5) is highlighted by the following definitional substitution:

$$rule_{6.1\text{-}6} \triangleq rule_{6.1\text{-}6a} \wedge \circ rule_{6.1\text{-}6b} \qquad (6.1\text{-}6)$$

where:

$$rule_{6.1\text{-}6a} \triangleq w_0 \wedge \circ w_1 \qquad (6.1\text{-}6a)$$

$$rule_{6.1\text{-}6b} \triangleq w_2 \wedge \circ w_3 \qquad (6.1\text{-}6b)$$

The following lemma describes an equivalent expression of a double nested rule.

LEMMA: NestBothEqv

$\vdash (f_0 \wedge \circ f_1) \wedge \circ(f_2 \wedge \circ f_3) \equiv f_0 \wedge \circ(f_1 \wedge f_2) \wedge \circ\circ f_3$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge \circ f_1) \wedge \circ(f_2 \wedge \circ f_3) \equiv (f_0 \wedge \circ f_1) \wedge \circ(f_2 \wedge \circ f_3)$ | tautology |
| 2 | $(f_0 \wedge \circ f_1) \wedge \circ(f_2 \wedge \circ f_3) \equiv (f_0 \wedge \circ f_1) \wedge (\circ f_2 \wedge \circ\circ f_3)$ | 1, NextAndDistEqv |
| 3 | $(f_0 \wedge \circ f_1) \wedge \circ(f_2 \wedge \circ f_3) \equiv f_0 \wedge (\circ f_1 \wedge \circ f_2) \wedge \circ\circ f_3$ | 2, associativity of $\wedge$ |
| 4 | $(f_0 \wedge \circ f_1) \wedge \circ(f_2 \wedge \circ f_3) \equiv f_0 \wedge \circ(f_1 \wedge f_2) \wedge \circ\circ f_3$ | 3, NextAndDistEqv |

Applying NestBothEqv to (6.1-6) yields:

$$w_0 \wedge \circ(w_1 \wedge w_2) \wedge \circ\circ w_3 \qquad (6.1\text{-}7)$$

Applying NextAndDistEqv to (6.1-7) yields:

$$w_0 \wedge \circ((w_1 \wedge w_2) \wedge \circ w_3) \qquad (6.1\text{-}8)$$

Again applying definitional substitution to highlight rule nesting:

$$rule_{6.1\text{-}9} \triangleq w_0 \wedge \circ rule_{6.1\text{-}9a} \qquad (6.1\text{-}9)$$

where:

$$rule_{6.1\text{-}9a} \triangleq (w_1 \wedge w_2) \wedge \circ w_3 \qquad (6.1\text{-}9a)$$

With the application of NestBothEqv and NextAndDistEqv, the double nested rule of (6.1-6) has been transformed into an equivalent general-form rule with only a nested rule state. One important benefit of this analysis is the clear identification that $w_1 \wedge w_2$

cannot be a contradiction if this rule is to hold. This is unambiguously depicted in both (6.1-7) and (6.1-9a).

The nesting of rules in both the rule condition and rule state is the basis for the rule-based form of sequential composition previously presented in Section 5.6.1. Unlike rule-based sequential composition of Section 5.6.1, the double-nested rules as presented in (6.1-5) and transformed by NestBothEqv need not share a common temporal formula. Stated another way, sequential composition using the general rule form, as previously discussed in Section 5.6.1 and addressed with TwoSeqRulesEqv1, TwoSeqRulesEqv2, and TwoSeqRulesImp, is a special case of the double nesting addressed in NestBothEqv. Consider the following example incorporating nested rules in the rule condition and rule state that share a common temporal formula:

$$(w_0 \wedge \circ w_1) \wedge \circ(w_1 \wedge \circ w_2) \tag{6.1-10}$$

Applying NestBothEqv (6.1-10) yields:

$$w_0 \wedge \circ(w_1 \wedge w_1) \wedge \circ\circ w_2 \tag{6.1-11}$$

Applying the idempotence of $\wedge$ to (6.1-11) yields:

$$w_0 \wedge \circ w_1 \wedge \circ\circ w_2 \tag{6.1-12}$$

Because the double nesting of (6.1-10) includes a common temporal formula, this example conforms to the simple sequential composition model previously presented in Section 5.6.1. Therefore, (6.1-12) could have been achieved by applying TwoSeqRulesEqv1 to (6.1-10). One distinct difference associated with the simple sequential composition model based on shared temporal formula (supported by TwoSeqRulesEqv1) and the double-nested compositional model (supported by NestBothEqv) is that the simple sequential composition model does not include the necessity that two different formulas in the same rule hold at the same time (e.g., $w_1 \wedge w_2$ in $rule_{6.1\text{-}9a}$).

The nesting of total rules in both the rule condition and the rule state of a general-form rule is described in the following lemmas.

LEMMA: TwoNestTotalRuleEqv1

$$\vdash ((f_{a_0} \wedge \circ f_{a_1}) \vee (\neg f_{a_0} \wedge \circ f_{a_2})) \wedge \circ((f_{b_0} \wedge \circ f_{b_1}) \vee (\neg f_{b_0} \wedge \circ f_{b_2}))$$
$$\equiv ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1})) \vee ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$$
$$\vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1})) \vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$$

or

$$\vdash (rule_{a_{true}} \vee rule_{a_{false}}) \wedge \circ(rule_{b_{true}} \vee rule_{b_{false}})$$
$$\equiv (rule_{a_{true}} \wedge \circ rule_{b_{true}}) \vee (rule_{a_{true}} \wedge \circ rule_{b_{false}})$$
$$\vee (rule_{a_{false}} \wedge \circ rule_{b_{true}}) \vee (rule_{a_{false}} \wedge \circ rule_{b_{false}})$$

where: $rule_{a_{true}} \triangleq (f_{a_0} \wedge \circ f_{a_1})$
$rule_{a_{false}} \triangleq (\neg f_{a_0} \wedge \circ f_{a_2})$
$rule_{b_{true}} \triangleq (f_{b_0} \wedge \circ f_{b_1})$
$rule_{b_{false}} \triangleq (\neg f_{b_0} \wedge \circ f_{b_2})$

Proof:

| | | |
|---|---|---|
| 1 | $((f_{a_0} \wedge \circ f_{a_1}) \vee (\neg f_{a_0} \wedge \circ f_{a_2})) \wedge \circ((f_{b_0} \wedge \circ f_{b_1}) \vee (\neg f_{b_0} \wedge \circ f_{b_2}))$ $\equiv ((f_{a_0} \wedge \circ f_{a_1}) \vee (\neg f_{a_0} \wedge \circ f_{a_2})) \wedge \circ((f_{b_0} \wedge \circ f_{b_1}) \vee (\neg f_{b_0} \wedge \circ f_{b_2}))$ | tautology |
| 2 | $\equiv ((f_{a_0} \wedge \circ f_{a_1}) \vee (\neg f_{a_0} \wedge \circ f_{a_2}))$ $\wedge (\circ(f_{b_0} \wedge \circ f_{b_1}) \vee \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ | 2, NextAnd-DistEqv |
| 3 | $\equiv ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1}))$ $\vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1}))$ $\vee ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ $\vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ | 3, Distribution of $\wedge$ over $\vee$ |
| 4 | $\equiv ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1}))$ $\vee ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ $\vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1}))$ $\vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ | 3, Commutivity of $\vee$ |

LEMMA: TwoNestTotalRuleEqv2

$$\vdash ((f_{a_0} \wedge \circ f_{a_1}) \vee (\neg f_{a_0} \wedge \circ f_{a_2})) \wedge \circ((f_{b_0} \wedge \circ f_{b_1}) \vee (\neg f_{b_0} \wedge \circ f_{b_2}))$$
$$\equiv (f_{a_0} \wedge \circ(f_{a_1} \wedge f_{b_0}) \wedge \circ\circ f_{b_1}) \vee (\neg f_{a_0} \wedge \circ(f_{a_2} \wedge f_{b_0}) \wedge \circ\circ f_{b_1})$$
$$\vee (f_{a_0} \wedge \circ(f_{a_1} \wedge \neg f_{b_0}) \wedge \circ\circ f_{b_2}) \vee (\neg f_{a_0} \wedge \circ(f_{a_2} \wedge \neg f_{b_0}) \wedge \circ\circ f_{b_2})$$

Proof:

| | | |
|---|---|---|
| 1 | $((f_{a_0} \wedge \circ f_{a_1}) \vee (\neg f_{a_0} \wedge \circ f_{a_2})) \wedge \circ((f_{b_0} \wedge \circ f_{b_1}) \vee (\neg f_{b_0} \wedge \circ f_{b_2}))$ | tautology |
| | $\equiv ((f_{a_0} \wedge \circ f_{a_1}) \vee (\neg f_{a_0} \wedge \circ f_{a_2})) \wedge \circ((f_{b_0} \wedge \circ f_{b_1}) \vee (\neg f_{b_0} \wedge \circ f_{b_2}))$ | |
| 2 | $\equiv ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1}))$ | 1, TwoNestTotal- |
| | $\vee ((f_{a_0} \wedge \circ f_{a_1}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ | RuleEqv1 |
| | $\vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(f_{b_0} \wedge \circ f_{b_1}))$ | |
| | $\vee ((\neg f_{a_0} \wedge \circ f_{a_2}) \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ | |
| 3 | $\equiv ((f_{a_0} \wedge \circ(f_{a_1} \wedge f_{b_0}) \wedge \circ\circ f_{b_1})$ | 2, NestBothEqv |
| | $\vee ((f_{a_0} \wedge \circ(f_{a_1} \wedge \neg f_{b_0}) \wedge \circ\circ f_{b_2})$ | |
| | $\vee ((\neg f_{a_0} \wedge \circ(f_{a_2} \wedge f_{b_0}) \wedge \circ\circ f_{b_1})$ | |
| | $\vee ((\neg f_{a_0} \wedge \circ(f_{a_2} \wedge \neg f_{b_0}) \wedge \circ\circ f_{b_2})$ | |

These lemmas are also expressed in terms of specific rule definitions to simplify presentation and highlight the underlying rule structure(s). With TwoNestTotal-RuleEqv1, a general-form rule composition of two nested total rules is decomposed to an equivalent disjunction of four general-form rules, with each disjunct composed of an individual rule from each of the two total rules. With TwoNestTotalRuleEqv2, the original composition is decomposed to an equivalent disjunction of four conjunctive series of state sequences.

With the nesting of two individual rules as a general-form rule as considered by NestBothEqv, care must be exercised so that such a composition does not result in a contradiction, thereby invalidating the original composition. This is demonstrated with (6.1-7), where the term $\circ(w_1 \wedge w_2)$ cannot be a contradiction. Because NestBothEqv is an equivalence, if such a contradiction is created, then the original composition of (6.1-6) is not valid. This problem can be avoided with the general-form rule nesting of two total rules. By inspection of the corresponding terms in outcome of TwoNestTotalRuleEqv2 – $\circ(f_{a_1} \wedge f_{b_0})$, $\circ(f_{a_1} \wedge \neg f_{b_0})$, $\circ(f_{a_2} \wedge f_{b_0})$, and $\circ(f_{a_2} \wedge \neg f_{b_0})$ – a contradiction is created only if either $f_{a_1}$ or $f_{a_2}$ is a contradiction. Otherwise, because both are conjunctively associated with $f_{b_0}$ and $\neg f_{b_0}$, no contradiction can result. Stated another way, if two total rules, where each total rule has the form $(f_i \wedge \circ f_j) \vee (\neg f_i \wedge \circ f_k)$ for any $i$, $j$, and $k$, are valid, then the nested composition of those two total rules as a general-form rule is valid.

The ITL operator chop can be used in creating nested rules. Consider the following rule which includes both the chop operator and a nested rule in the specification of the rule state:

$$w_0 \wedge \circ(w_1 \,;\, (w_1 \wedge \circ w_2)) \qquad (6.1\text{-}13)$$

Applying StateAndNextChop yields the following equivalent form:

$$(w_0 \wedge \circ w_1) \,;\, (w_1 \wedge \circ w_2) \qquad (6.1\text{-}14)$$

In this equivalent form, (6.1-14) is an example of sequential composition using chop as previously described in Section 5.6.2. Applying StateTwoChopRulesImp allows (6.1-14) to be transformed to:

$$w_0 \wedge \circ(w_1 \,;\, \circ w_2) \qquad (6.1\text{-}15)$$

Therefore, with the application of StateAndNextChop and StateTwoChopRulesImp, the rule nesting of (6.1-13) has been eliminated and (6.1-13) has been simplified to the general rule form of (6.1-15).

Consider the following nested rule that includes two general-form rules that are chopped and nested in the rule state.

$$w_0 \wedge \circ((w_1 \wedge \circ w_2) \,;\, (w_3 \wedge \circ w_4)) \qquad (6.1\text{-}16)$$

The overall structure of this rule can be clarified with some definitional substitutions:

$$w_0 \wedge \circ(rule_{6.1\text{-}17a} \,;\, rule_{6.1\text{-}17b}) \qquad (6.1\text{-}17)$$

where:

$$rule_{6.1\text{-}17a} \triangleq w_1 \wedge \circ w_2 \qquad (6.1\text{-}17a)$$

$$rule_{6.1\text{-}17b} \triangleq w_3 \wedge \circ w_4 \qquad (6.1\text{-}17b)$$

This rule is a general-form rule that includes two general-form rules nested in the rule state. The following lemma allows the transformation of a nested rule that includes two chopped rules in the rule state:

LEMMA: StateNestRuleStateChopEqv

$\vdash\ w_0 \wedge \circ((w_1 \wedge \circ f_2) ; (w_3 \wedge \circ f_4)) \equiv (w_0 \wedge \circ w_1 \wedge \circ\circ f_2) ; (w_3 \wedge \circ f_4)$

or

$\vdash\ w_0 \wedge \circ(rule_1 ; rule_2) \equiv (w_0 \wedge \circ w_1 \wedge \circ\circ f_2) ; rule_2$

where: $rule_1 \triangleq w_1 \wedge \circ f_2$
$rule_2 \triangleq w_3 \wedge \circ f_4$

Proof:

| | | |
|---|---|---|
| 1 | $w_0 \wedge \circ((w_1 \wedge \circ f_2) ; (w_3 \wedge \circ f_4))$ | tautology |
| | $\equiv w_0 \wedge \circ((w_1 \wedge \circ f_2) ; (w_3 \wedge \circ f_4))$ | |
| 2 | $\equiv (w_0 \wedge \circ(w_1 \wedge \circ f_2)) ; (w_3 \wedge \circ f_4)$ | 1, ITL (StateAndNextChop) |
| 3 | $\equiv (w_0 \wedge \circ w_1 \wedge \circ\circ f_2) ; (w_3 \wedge \circ f_4)$ | 3, NextAndDistEqv |

This lemma is also expressed in terms of specific rule definitions to simplify presentation and highlight the underlying rule structure(s).

Applying StateNestRuleStateChopEqv to (6.1-16) yields:

$$(w_0 \wedge \circ w_1 \wedge \circ\circ w_2) ; (w_3 \wedge \circ w_4) \tag{6.1-18}$$

In this form, the sequence specified by the original rule is clear. However, applying NextAndDistEqv and TwoSeqRulesEqv1 to (6.1-18) yields an equivalent rule consisting of three component general-form rules:

$$((w_0 \wedge \circ w_1) \wedge \circ(w_1 \wedge \circ w_2)) ; (w_3 \wedge \circ w_4) \tag{6.1-19}$$

Substituting defined rule names for the component rules yields:

$$(rule_{6.1\text{-}20a} \wedge \circ rule_{6.1\text{-}17a}) ; rule_{6.1\text{-}17b} \tag{6.1-20}$$

where:

$$rule_{6.1\text{-}20a} \triangleq w_0 \wedge \circ w_1 \qquad (6.1\text{-}20a)$$

And because $rule_{6.1\text{-}20a} \wedge \circ rule_{6.1\text{-}17a}$ is a general-form rule, an additional definitional substitution can be performed:

$$rule_{6.1\text{-}21a} \, ; rule_{6.1\text{-}17b} \qquad (6.1\text{-}21)$$

where:

$$rule_{6.1\text{-}21a} \triangleq rule_{6.1\text{-}20a} \wedge \circ rule_{6.1\text{-}17a} \qquad (6.1\text{-}21a)$$

The net result of this analysis is that (6.1-16), (6.1-18), and (6.1-19) are equivalent. Because StateNestRuleStateChopEqv, TwoSeqRulesEqv1, and NextAnd-DistEqv are equivalence lemmas, all offer substantial flexibility when used together in the forward transformation of rules into equivalent forms, or the reverse transformation of observed sequences into equivalent rules.

Although a wide variety of rule nestings using chop are possible, some nestings may have unanticipated consequences. Consider the following rule which includes both the chop operator and a nested rule in the rule condition:

$$(w_0 \, ; (w_0 \wedge \circ w_1)) \wedge \circ w_2 \qquad (6.1\text{-}22)$$

The following lemmas describes the reduction of this form of nested rule.

LEMMA: NestRuleCondChopImpl

$\vdash (w_0 \, ; (w_0 \wedge \circ f_1)) \wedge \circ f_2$ implies $\vdash (w_0 \, ; \circ f_1) \wedge \circ f_2$

Proof:

| | | |
|---|---|---|
| 1 | $(w_0 \, ; (w_0 \wedge \circ f_1)) \wedge \circ f_2$ | premise |
| 2 | $w_0 \, ; (w_0 \wedge \circ f_1)$ | 1, $\wedge$ elimination |
| 3 | $w_0 \, ; w_0 \wedge w_0 \, ; \circ f_1$ | 2, ITL (ChopAndImp) |
| 4 | $w_0 \, ; \circ f_1$ | 3, $\wedge$ elimination |
| 5 | $\circ f_2$ | 1, $\wedge$ elimination |
| 6 | $(w_0 \, ; \circ f_1) \wedge \circ f_2$ | 4, 5, $\wedge$ introduction |

LEMMA: NestRuleCondChopImp2

$\vdash (w_0 ; (w_0 \wedge \circ f_1)) \wedge \circ w_2$ implies $\vdash w_0 \wedge \circ f_2$

Proof:

| | | |
|---|---|---|
| 1 | $(w_0 ; (w_0 \wedge \circ f_1)) \wedge \circ f_2$ | premise |
| 2 | $w_0 ; (w_0 \wedge \circ f_1)$ | 1, $\wedge$ elimination |
| 3 | $w_0$ | 2, ITL (StateChop) |
| 4 | $\circ f_2$ | 3, $\wedge$ elimination |
| 5 | $w_0 \wedge \circ f_2$ | 3, 4, $\wedge$ introduction |

Applying NestRuleCondChopImp1 to (6.1-22) yields:

$$w_0 ; \circ w_1 \wedge \circ w_2 \qquad (6.1\text{-}23)$$

However, because $\circ w_1$ is chopped to the state formula $w_0$, and because the satisfaction of a state formula depends only on the first state of a multi-state sequence, the chopping of $\circ w_1$ to $w_0$ holds if $w_1$ follows any state in the multi-state sequence satisfying $w_0$. Therefore, $w_1$ does not have to hold in the next state after the single state satisfying $w_0$ but after some next state after the single state satisfying $w_0$. Further, for $\circ w_2$ to hold, $w_2$ must be satisfied by the next state after the state sequence satisfying $w_0$. This is demonstrated by the application of NestRuleCondChopImp2 to (6.1-22) which yields:

$$w_0 \wedge \circ w_2 \qquad (6.1\text{-}24)$$

Therefore, with this form of nested construction, the original rule (6.1-22) and the derivative rule (6.1-23) will hold even if $w_1$ is satisfied by a state that occurred after the state satisfying $w_2$. Although not immediately evident from an initial inspection of (6.1-22), the transformations presented at (6.1-23) and (6.1-24) demonstrate the potential confusion and corresponding problems that may result from the nesting of a chopped rule in the rule condition.

## 6.2 Recursion

As applied to rules, recursion describes the circumstance where a rule is defined in terms of that rule. Expressed in terms of nesting, recursion is the nesting of a rule within itself. An example of a simple recursive rule is:

$$rule_{6.2\text{-}1} \triangleq f_0 \wedge \circ(f_1 \wedge rule_{6.2\text{-}1}) \qquad (6.2\text{-}1)$$

Substituting the definition of $rule_{6.2\text{-}1}$ into an instantiation of $rule_{6.2\text{-}1}$ yields:

$$f_0 \wedge \circ(f_1 \wedge f_0 \wedge \circ(f_1 \wedge rule_{6.2\text{-}1})) \qquad (6.2\text{-}2)$$

Applying NextAndDistEqv twice yields the equivalent form:

$$f_0 \wedge \circ f_1 \wedge \circ f_0 \wedge \circ\circ f_1 \wedge \circ\circ rule_{6.2\text{-}1} \qquad (6.2\text{-}3)$$

In this equivalent form, and with the continued substitution of the definition of $rule_{6.2\text{-}1}$, the sequence resulting from this recursive rule is clear:

$$f_0 \wedge \circ f_1 \wedge \circ f_0 \wedge \circ\circ f_1 \wedge \circ\circ f_0 \wedge \circ\circ\circ f_1 \wedge \circ\circ\circ rule_{6.2\text{-}1} \qquad (6.2\text{-}4)$$

Although $rule_{6.2\text{-}1}$ is a ideal initial example of rule recursion because of its simplicity, that simplicity compromises its applicability to more realistic situations. Referencing the previous discussion in Section 5.1 regarding total rules, $rule_{6.2\text{-}1}$ includes no specification regarding the state sequence that will result if the rule condition $f_0$ is not satisfied. Therefore, consider the following recursive rule composed as a total rule:

$$rule_{6.2\text{-}5} \triangleq (f_0 \wedge \circ(f_1 \wedge rule_{6.2\text{-}5})) \vee (\neg f_0 \wedge \circ f_{unchanged}) \qquad (6.2\text{-}5)$$

The expansion resulting from the substitution of the definition of $rule_{6.2\text{-}5}$ into an instantiation of $rule_{6.2\text{-}5}$ is described by the following lemma.

LEMMA: RecursTotalRuleExpan

$$\begin{aligned}\vdash\quad & (f_1 \wedge \circ(f_2 \wedge \mathit{rule})) \vee (\neg f_1 \wedge \circ f_{unchanged}) \equiv \\ & (f_1 \wedge \circ f_2 \wedge \circ f_1 \wedge \circ\circ f_2 \wedge \circ\circ \mathit{rule}) \vee (f_1 \wedge \circ f_2 \wedge \circ\neg f_1 \wedge \circ\circ f_{unchanged}) \\ & \quad \vee (\neg f_1 \wedge \circ f_{unchanged}) f_0 \wedge \circ(f_1 \wedge f_2) \wedge \circ\circ f_3\end{aligned}$$

where: $\mathit{rule} \equiv (f_1 \wedge \circ(f_2 \wedge \mathit{rule})) \vee (\neg f_1 \wedge \circ f_{unchanged})$

Proof:

| | | |
|---|---|---|
| 1 | $\mathit{rule} \equiv (f_1 \wedge \circ(f_2 \wedge \mathit{rule})) \vee \mathit{rule}_{false}$ | premise |
| 2 | $\equiv (f_1 \wedge \circ(f_2 \wedge ((f_1 \wedge \circ(f_2 \wedge \mathit{rule})) \vee \mathit{rule}_{false})))$ | 1, substitution of |
| | $\vee (\neg f_1 \wedge \circ f_{unchanged})$ | equivalance |
| 3 | $\equiv (f_1 \wedge \circ f_2 \wedge \circ((f_1 \wedge \circ(f_2 \wedge \mathit{rule})) \vee \mathit{rule}_{false}))$ | 2, NextAndDistEqv |
| | $\vee (\neg f_1 \wedge \circ f_{unchanged})$ | |
| 4 | $\equiv (f_1 \wedge \circ f_2 \wedge (\circ(f_1 \wedge \circ(f_2 \wedge \mathit{rule})) \vee \circ \mathit{rule}_{false}))$ | 3, NextOrDistEqv |
| | $\vee (\neg f_1 \wedge \circ f_{unchanged})$ | |
| 5 | $\equiv (f_1 \wedge \circ f_2 \wedge \circ(f_1 \wedge \circ(f_2 \wedge \mathit{rule})))$ | 4, Distribution of $\wedge$ over $\vee$ |
| | $\vee (f_1 \wedge \circ f_2 \wedge \circ(\neg f_1 \wedge \circ f_{unchanged}))$ | |
| | $\vee (\neg f_1 \wedge \circ f_{unchanged})$ | |
| 6 | $\equiv (f_1 \wedge \circ f_2 \wedge \circ f_1 \wedge \circ\circ(f_2 \wedge \mathit{rule}))$ | 5, NextAndDistEqv |
| | $\vee (f_1 \wedge \circ f_2 \wedge \circ(\neg f_1 \wedge \circ f_{unchanged}))$ | |
| | $\vee (\neg f_1 \wedge \circ f_{unchanged})$ | |
| 7 | $\equiv (f_1 \wedge \circ f_2 \wedge \circ f_1 \wedge \circ\circ f_2 \wedge \circ\circ \mathit{rule})$ | 6, NextAndDistEqv |
| | $\vee (f_1 \wedge \circ f_2 \wedge \circ(\neg f_1 \wedge \circ f_{unchanged}))$ | |
| | $\vee (\neg f_1 \wedge \circ f_{unchanged})$ | |
| 8 | $\equiv (f_1 \wedge \circ f_2 \wedge \circ f_1 \wedge \circ\circ f_2 \wedge \circ\circ \mathit{rule})$ | 7, NextAndDistEqv |
| | $\vee (f_1 \wedge \circ f_2 \wedge \circ\neg f_1 \wedge \circ\circ f_{unchanged})$ | |
| | $\vee (\neg f_1 \wedge \circ f_{unchanged})$ | |

Applying RecursTotalRuleExpan to (6.2-5) yields the following equivalent disjunctive structure:

$$\begin{aligned}& (f_0 \wedge \circ f_1 \wedge \circ f_0 \wedge \circ\circ f_1 \wedge \circ\circ \mathit{rule}_{6.2\text{-}5}) \\ & \vee (f_0 \wedge \circ f_1 \wedge \circ\neg f_0 \wedge \circ\circ f_{unchanged}) \\ & \vee (\neg f_0 \wedge \circ f_{unchanged})\end{aligned} \tag{6.2-6}$$

With this expansion, the significance of the total rule form is clear. The state sequence specified by $\mathit{rule}_{6.2\text{-}5}$ can be expanded, consistent with the recursive definition of $\mathit{rule}_{6.2\text{-}5}$, until the rule condition $f_0$ is not met, that is, until $\neg f_0$ is true. If the first state sequence does not satisfy $f_0$, the third disjunct of (6.2-6), $\neg f_0 \wedge \circ f_{unchanged}$, specifies the next state as unchanged. If the first state sequence satisfies $f_0$ but the second state sequence does not, the second disjunct of (6.2-6), $f_0 \wedge \circ f_1 \wedge \circ\neg f_0 \wedge \circ\circ f_{unchanged}$, specifies the third state as unchanged. This unchanged status of the third state is unchanged relative to the second state which satisfies $f_1$ (as specified by $\circ f_1$) but not $f_0$. The sequence specified in (6.2-6) can be expanded further, as needed, by the substitution of the definition of $rule_{6.2\text{-}5}$ into (6.2-6) and the application of RecursTotalRuleExpan.

An important issue associated with recursive rules is termination of the rule. Consider the following simple rule:

$$rule_{6.2\text{-}7} \triangleq f_0 \wedge \circ(f_0 \wedge rule_{6.2\text{-}7}) \qquad (6.2\text{-}7)$$

Substituting the definition of $rule_{6.2\text{-}7}$ into an instantiation of $rule_{6.2\text{-}7}$ and applying the applying the appropriate ITL and propositional logic yields the equivalent form:

$$f_0 \wedge \circ f_0 \wedge \circ\circ f_0 \wedge \circ\circ rule_{6.2\text{-}7} \qquad (6.2\text{-}8)$$

If the initial rule condition $f_0$ is satisfied, then the rule state, that is, the next state specified by the rule, will also satisfy $f_0$. Because all rule states reached by the rule satisfy the rule condition, this recursive rule will never terminate. Therefore, for a recursive rule to terminate, the rule codomain must contain at one least state that is not in the rule domain. Formally, for the rule $rule_{recursive}$ to terminate:

$$\exists \sigma \in codomain(rule_{recursive}) \mid (\sigma \notin domain(rule_{recursive})\,) \qquad (6.2\text{-}9)$$

A common application of rule recursion is to implement loops. Through the use of counter variables or logical tests, for-loops, while-loops, or similar looping programming structures can be created. As many legacy and non-legacy applications include such looping structures, rule recursion offers a powerful rule-based technique for reasoning about such code structures.

## 6.3 Guarded Composition

Dijkstra (1975, 1976) introduced the logical concept of a 'guarded command' to allow operational non-determinacy with respect to the final system state based on, and subject to, the current state of a given system. This guarded command approach was originally conceived as a reliable method of evaluating and executing simultaneous I/O interrupts, thereby avoiding machine deadlock resulting from the consistent and deterministic choice and service of one interrupt over another. Guarded command concepts have been explicitly incorporated into various programming languages including Occam (Roscoe and Hoare, 1986) and WSL (Ward, 2001). Whereas the bar $[]$ is frequently used as the guarded command operator to link unordered alternatives, in this thesis, disjunction is used to compose rules into guarded command systems.

Although originally conceived to represent non-determinacy, guarded composition can be used to implement both non-deterministic and deterministic choice depending on the implementation of the guards. Under guarded composition, only those logical structures bound to a guard that is satisfied by the current system state sequence are candidates for selection and execution. If the guards do not overlap and each guarded logical structure in the guarded composition is satisfied by a different system state, deterministic choice results. Such a deterministic guarded structure functions like the switch or case constructs found in many programming paradigms.

Within the context of the rule model presented in this thesis, the total rule form $(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2)$ is an example of a simple, deterministic guarded composition. In this rule-based implementation of guarded composition, the rule condition of each rule serves as the guard, guarding the next state sequence defined by the rule state formula. For total rule $(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2)$, the state sequence satisfying $\circ f_1$ is guarded by $f_0$ in that $f_1$ can occur in the next state only if the guard $f_0$ is satisfied. Conversely, the state sequence satisfying $\circ f_2$ is guarded by $\neg f_0$ in that $f_2$ can occur in the next state only if the guard $f_0$ is not satisfied (i.e., $\neg f_0$ is true). Because no state sequence can satisfy both $f_0$ and $\neg f_0$ (i.e., $f_0 \wedge \neg f_0 \equiv false$), the guards cannot overlap and deterministic choice is implemented. Subject to the requirement that the rule conditions not overlap, this approach to deterministic composition can be expanded as necessary by disjunctively incorporating additional rules.

However, if two or more guards overlap such that they are satisfied by the same state sequence, nondeterministic choice is implemented. With such an overlapping guarded command approach, multiple alternative state sequences can be associated with a single guard state. Therefore, when a guarded command system with overlapping guards is executed repetitively, different final states may result from the same initial state. With regard to implementation, the selection of the one alternative state sequence from the set of multiple alternative state sequences bound to a satisfied guard must be random to meet the expectation of fairness with respect to the nondeterminacy. Abandoning this random approach and adding a probabilistic technique to the selection of a single rule state from the set of multiple alternative state sequences bound to a satisfied guard forms the basis for a probabilistic guarded composition, analogous to a probabilistic guarded command language (He et al., 1997; Morgan and McIver, 1999).

A simple nondeterministic guarded command system is described in terms of general-form rules as:

$$(f_0 \wedge \circ f_1) \vee (f_0 \wedge \circ f_2) \vee (\neg f_0 \wedge \circ f_{unchanged}) \tag{6.3-1}$$

The state sequences satisfying $\circ f_1$ and $\circ f_2$ are both guarded by $f_0$ in that $f_1$ or $f_2$ can occur in the next state only if the guard $f_0$ is satisfied. Because these rules share a common formula expressing the rule condition, (6.3-1) is transformed by applying propositional logic to yield the equivalent form:

$$(f_0 \wedge (\circ f_1 \vee \circ f_2 ) \vee (\neg f_0 \wedge \circ f_{unchanged}) \tag{6.3-2}$$

Applying NextOrDistEqv to (6.3-2) yields the equivalent form:

$$(f_0 \wedge \circ(f_1 \vee f_2)) \vee (\neg f_0 \wedge \circ f_{unchanged}) \tag{6.3-3}$$

With these transformations, the three-rule nondeterministic guarded command structure of (6.3-1) has been transformed into a nondeterministic total rule – total in that all state sequences will either satisfy $f_0$ or $\neg f_0$, and nondeterministic in that a state sequence satisfying either $f_1$ or $f_2$ will follow one satisfying $f_0$. Whereas (6.3-3) has been limited to two rule conditions and three rule states, there are no limitations with regard to the number or nature of the rules used to described a rule-based guarded command system.

Critical to the composition of any guarded command system is the unambiguous representation of the logical expectations of the system. Consider the following guarded command system disjunctively composed of two total rules:

$$\begin{aligned}&((f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_{unchanged}))\\ &\vee ((f_2 \wedge \circ f_3) \vee (\neg f_2 \wedge \circ f_{unchanged}))\end{aligned} \tag{6.3-4}$$

As previously discussed, a total rule is a simple implementation of a deterministic guarded command system. Therefore, (6.3-4) can be described as a guarded command system composed of two deterministic guarded command systems. However, careful analysis of (6.3-4) demonstrates that such a composition yields a nondeterministic guarded command system. Applying propositional logic to (6.3-4) yields:

$$(f_0 \wedge \circ f_1) \vee (f_2 \wedge \circ f_3) \vee ((\neg f_0 \vee \neg f_2) \wedge \circ f_{unchanged}) \tag{6.3-5}$$

In this equivalent form consisting of three rules, it is evident that the guards may overlap. If $f_0 \not\equiv f_2$, then the guards can overlap. Therefore, (6.3-5) is nondeterministic. In contrast, the following rule system, not derivative of (6.3-4), is a deterministic guarded command system as only one rule state can be satisfied:

$$(f_0 \wedge \circ f_1) \vee (f_2 \wedge \circ f_3) \vee (\neg(f_0 \vee f_2) \wedge \circ f_{unchanged}) \tag{6.3-6}$$

In presenting these contrasting examples, no assertion is made that either (6.3-5) or (6.3-6) is correct or incorrect, better or worse, preferred or not. Instead, they are presented to demonstrate the necessity of analyzing guarded command system formations to assure that the implementations are consistent with the underlying logical expectations for that system.

## 6.4 Parallel Composition

In contrast to the explicitly linear execution order that results from sequential composition, parallel composition allows for two or more programming structures to be executed concurrently. Practically, parallel composition allows for two or more programming structures to be executed under some defined model of concurrency. Parallel composition is expressed using the parallel operator || to connect the structures that are to be executed in parallel. Applied to rules, $(f_0 \wedge \circ f_1) \,||\, (f_2 \wedge \circ f_3)$ specifies that the rules $f_0 \wedge \circ f_1$ and $f_2 \wedge \circ f_3$ are to be executed concurrently.

Apt and Olderog (1997) identify three common types of parallel composition in programs – disjoint parallelism, parallelism with share variables, and parallelism with synchronization. Parallel rules with shared variables may potentially interfere with each other, whereas parallel rules with synchronization require rule execution to be suspended and then restarted. Disjoint parallelism is the most restricted form of parallel composition and is probably the most applicable to legacy code analysis. This section is limited to the analysis of rules and rule formation within the context of disjoint parallelism.

The concept of disjoint parallel programs was introduced by Hoare (1975) in an attempt to define the conditions under which certain parallel programs can be reduced to equivalent sequential programs. Two programs are considered disjoint if neither change the variables accessed and used by the other. Extending this concept to rules, two rules are disjoint if neither rule updates variables used by the other rule in assessing satisfaction of the rule condition or establishing the rule state associated with that rule condition. Stated another way, for two rules to be disjoint, the variables in the frame of one rule cannot be used in the other rule in the formulas that specify the rule condition or the next rule state.

The variables in the frame of a formula have been previously described as $W$. For the same formula, let $V$ be the set of all variables used to define, specify, or calculate the new values of the variables in $W$. Because some variables in $W$ may be used to calculate other variables in $W$, including the recursive definition of a new variable value, $V$ may include variables from $W$.

Thus, for any formula, there exists some set of variables $V$ and $W$ and that formula may be described by the set of variable $V \cup W$. Consider two formulas, $f_0$ and $f_1$, such that each is described by $V_0 \cup W_0$ and $V_1 \cup W_1$. $f_0$ is independent of $f_1$ if the variables of $V_0$ do not include any variables in $W_1$, or $V_0 \cap W_1 = \varnothing$. Similarly, $f_1$ is independent of $f_0$ if the variables of $V_1$ do not include any variables in $W_0$, or $V_1 \cap W_0 = \varnothing$. Therefore, the two formulas, $f_0$ and $f_1$, are independent or disjoint of each other if:

$$(V_0 \cap W_1 = \varnothing) \wedge (V_1 \cap W_0 = \varnothing) \qquad (6.4\text{-}1)$$

Expanding this concept to rules, let rule *rule* be a general-form rule defined as $f_0 \wedge \bigcirc f_1$. The variables in the frame of *rule* have been previously described in Section 4.5 as $W_{rule}$. For *rule*, let $V_{rule}$ be the set of all variables used to specify $f_0$ and used in $f_1$ to calculate the next values of the variables in $W_{rule}$. Because some variables in $W_{rule}$ may be used to calculate other variables in $W_{rule}$, $V_{rule}$ may include variables from $W_{rule}$. Consider parallel two rules, $rule_a$ and $rule_b$, defined as $f_{a_0} \wedge \bigcirc f_{a_1}$ and $f_{b_0} \wedge \bigcirc f_{b_1}$, respectively. Parallel rules $rule_a$ and $rule_b$ are disjoint if:

$$(V_{rule_a} \cap W_{rule_b} = \varnothing) \wedge (V_{rule_b} \cap W_{rule_a} = \varnothing) \qquad (6.4\text{-}2)$$

Because disjoint parallel rules are independent of each other with respect to the variables used to express the rule conditions and updated in the rule states, they can be expressed as sequential rules using either of the two previously presented techniques for the sequential composition. Similarly, because they are disjoint, they may also be expressed as parallel rules should the need arise.

## 6.5 Equivalent Rules

Numerous models of equivalence exist for comparing objects and structures in computer science. This section offers a brief review of some of the more relevant concepts as a basis for deriving equivalence models that are applicable to general-form rules. The rule algebra presented in this research is used to demonstrate three forms of rule equivalence – strong equivalence (or strong bisimulation), transformational equivalence, and non-temporal equivalence.

Apt and Olderog (1997) declare two computations input/output equivalent if they start in the same state and then result in the same final state. For parallel programs, they extend this model to the notion of permutation equivalence, that two computations are permutation equivalent if they are input/output equivalent and the sequences of transitions in each computation are permutations of each other. Fokkink (2000) describes two processes as trace equivalent if they can execute exactly the same strings of actions and observes that trace equivalence ignores the effect of branching and may be inadequate in describing concurrency. Pitts (1997) describes two program expressions as contextually equivalent if they can be interchanged in a program without changing the program outcome. De Nicola and Hennessey (1984) offer a testing approach to demonstrate natural equivalence; two processes are equivalent if they pass exactly the same set of relevant tests.

Many formal models of equivalence are related to the concept of bisimulation. Park (1981) introduced the formal model of bisimulation as an approach to assessing the equivalence of two finite automata. One automaton bisimulates another automaton if there exists a single relationship that relates all states of the first automaton to the states of the second automaton and relates all states of the second automaton to the states of the first automaton. This concept has been extended to numerous computational paradigms, including process graphs (Baeten and Weijland, 1990), finite transition systems (Arnold, 1994), and calculus of communicating systems (Milner, 1989). Under these paradigms, the system nodes, states, or agents and the transitions that connect them must be considered in the relationship that defines a bisimulation between two systems. Fokkink (2000) offers a general and informal description of bisimulation applicable to these computational paradigms – two processes are bisimilar if they can execute the same string of actions and have the same branching structure. Many bisimulation models and the corresponding equivalence models are differentiated as weak or strong models, depending on whether the silent actions of two systems (i.e., those transitions that are invisible or unobservable to the external observer) must be matched one-for-one. Using the concept of weak bisimulation, Milner (1989) offers a model of observational equivalence where the external, observable behavior of two systems follows the same pattern, but the internal behaviors of the two systems may differ substantially.

This thesis will use a general framework for equivalence based on the assertion that two temporal formulas are equivalent if they are satisfied by the same state sequences. Because rules are themselves temporal formula, two rules are equivalent if they are satisfied by the same state sequences.

Consider the following two rules:

$$rule_{6.5\text{-}1} \triangleq f_0 \wedge \circ f_1 \qquad (6.5\text{-}1)$$

$$rule_{6.5\text{-}2} \triangleq f_{0'} \wedge \circ f_{1'} \qquad (6.5\text{-}2)$$

Demonstrating the equivalence of these two rules requires either the assertion or proof that $f_0 \equiv f_{0'}$ and $\circ f_1 \equiv \circ f_{1'}$. With such substitutions, both rules describe and/or are satisfied by the same state sequences. Such substitutions of individual temporal formula yield the strongest claim of equivalence for the associated rules as no other transformations or reductions on the original rules are required. Because the equivalences between individual formulas forming $rule_{6.5\text{-}1}$ and $rule_{6.5\text{-}2}$ are instantiations of a single relationship that is reflexive, symmetric, and transitive, $rule_{6.5\text{-}1}$ and $rule_{6.5\text{-}2}$ are described as strongly equivalent.

Consider the following two rules, each composed of two rules:

$$rule_{6.5\text{-}3} \triangleq (f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \qquad (6.5\text{-}3)$$

$$rule_{6.5\text{-}4} \triangleq (f_0 \wedge \circ f_{1'}) \wedge \circ(f_{1'} \wedge \circ f_2) \qquad (6.5\text{-}4)$$

Applying TwoSeqRulesEqv1 to each yields the equivalent forms:

$$f_0 \wedge \circ f_1 \wedge \circ\circ f_2 \qquad (6.5\text{-}5)$$

$$f_0 \wedge \circ f_{1'} \wedge \circ\circ f_2 \qquad (6.5\text{-}6)$$

In the absence of any knowledge that $\circ f_1 \equiv \circ f_{1'}$, the strong equivalence discussed above cannot be claimed. However, applying TwoSeqRulesImp to (6.5-3) and (6.5-4) yields (6.5-7) and (6.5-8), respectively:

$$f_0 \wedge \circ\circ f_2 \qquad (6.5\text{-}7)$$

$$f_0 \wedge \circ\circ f_2 \qquad (6.5\text{-}8)$$

Whereas (6.5-7) and (6.5-8) are identical (and therefore equivalent), TwoSeqRulesImp is not an equivalence preserving transformation. Therefore, $rule_{6.5\text{-}3}$ and $rule_{6.5\text{-}4}$ are considered transformationally equivalent. Alternatively, with these transformations, $rule_{6.5\text{-}3}$ and $rule_{6.5\text{-}4}$ are described as input/output equivalent, because both have been transformed into a general-form rule that is satisfied by the same input, specified by the rule condition $f_0$, and is associated with the same output, as described by the rule state $\circ\circ f_2$.

Consider the following two rules:

$$rule_{6.5\text{-}9} \triangleq (f_0 \wedge \circ f_1) \wedge \circ(f_1 \wedge \circ f_2) \tag{6.5-9}$$

$$rule_{6.5\text{-}10} \triangleq (f_0 \wedge \circ f_{1'}) \wedge \circ((f_{1'} \wedge \circ f_{1''}) \wedge \circ(f_{1''} \wedge \circ f_2)) \tag{6.5-10}$$

Applying TwoSeqRulesImp to (6.5-9) yields:

$$f_0 \wedge \circ\circ f_2 \tag{6.5-11}$$

Using NextAndDistEqv, TwoSeqRulesImp, and propositional logic, (6.5-10) is transformed to:

$$f_0 \wedge \circ\circ\circ f_2 \tag{6.5-12}$$

Comparing (6.5-11) and (6.5-12), and given that $\circ\circ f_2 \not\equiv \circ\circ\circ f_2$, $rule_{6.5\text{-}9}$ and $rule_{6.5\text{-}10}$ are not transformationally equivalent. However, both rules are described as non-temporally equivalent as they differ only by the number of skip constructs (i.e., the ITL next operator $\circ$) chopped ahead of the common rule state $f_2$.

Three forms of equivalence – strong equivalence (or strong bisimulation), transformational equivalence, and non-temporal equivalence – have been presented in this section. No doubt, other forms or other models of rule equivalence are possible. The formalization of the models presented above and the development of alternative equivalence models applicable to general-form rules remain open questions.

## 6.6 Rules in Programming Structures

In legacy programs, three programming structures are frequently used to represent rules – the if-then-else programming structure, the while structure, and the indexed for-loop structure. In this section, these three structures are examined in relation to the general form-rule $f_i \wedge \circ f_j$.

### 6.6.1 If-Then-Else Structures

If-then-else programming structures are a common and widely used method in many imperative-programming languages for implementing deterministic choice between two complementary alternatives. An if-then-else structure such as 'if P then Q else R' is commonly represented in non-temporal propositional logic as $(P \wedge Q) \vee (\neg P \wedge R)$ or the equivalent form $(\neg P \vee Q) \wedge (P \vee R)$ (Hoare, 1985). As the latter form includes the definition of implication, that form is equivalent to $(P \supset Q) \wedge (\neg P \supset R)$.

Moszkowski (1986) defines an if-then-else structure in ITL as:

$$\text{if } b \text{ then } w_1 \text{ else } w_2 \triangleq (b \supset w_1) \wedge (\neg b \supset w_2) \qquad (6.6.1\text{-}1)$$

where $b$ is a Boolean expression. As presented in Table 4.3-5, the 'if $f_0$ then $f_1$ else $f_2$' structure in ITL is now defined as:

$$\text{if } f_0 \text{ then } f_1 \text{ else } f_2 \triangleq (f_0 \wedge f_1) \vee (\neg f_0 \wedge f_2) \qquad (6.6.1\text{-}2)$$

As previously mentioned, the conjunctive form $(f_0 \wedge f_1) \vee (\neg f_0 \wedge f_2)$ and the implication form $(f_0 \supset f_1) \vee (\neg f_0 \supset f_2)$ are provably equivalent.

In this thesis, a variation of the current ITL definition is used, and the if-then-else structure is implemented as a pair of general-form rules as:

$$\text{if } f_0 \text{ then } \circ f_1 \text{ else } \circ f_2 \triangleq (f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2) \qquad (6.6.1\text{-}3)$$

As previously described in Section 5.1, this rule form is also described as a total rule because all possible cases of the rule condition are considered, either $f_0$ or $\neg f_0$. As previously described in Section 6.3, this rule form is an example of a simple, deterministic guarded composition that includes two non-overlapping guards, $f_0$ and $\neg f_0$.

A more limited if-then programming construct is implemented by substituting $f_{unchanged}$ for $f_2$, leaving the system in an unchanged state if the condition $f_0$ is not met.

The semantics of $f_{unchanged}$ have been previously described in Section 5.1. Alternatively and with a minor deviation from the general rule form $f_i \wedge \circ f_j$, the silent transition associated with $\circ f_{unchanged}$ can be avoided with the use of the ITL construct empty:

$$\text{if } f_0 \text{ then } \circ f_1 \triangleq (f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \text{empty}) \tag{6.6.1-4}$$

An important use of the if-then-else programming construct is in creating nested if-then-else constructs. With such nested constructs, multiple guards can be applied systematically and bound to specific outcomes. Within the context of rules, using nested if-then-else constructs allows the hierarchical association of multiple rule states to a given rule state. Using the rule-based definition of the if-then-else construct presented in (6.6.1-3), a nested if-then-else is created by instantiating an if-then-else construct as each of the respective rule states $f_1$ and $f_2$. Consider the following example of a nested if-then-else construct.

$$(w_0 \wedge \circ rule_{6.6.1\text{-}5a}) \vee (\neg w_0 \wedge \circ rule_{6.6.1\text{-}5b}) \tag{6.6.1-5}$$

where:

$$rule_{6.6.1\text{-}5a} \triangleq rule_{6.6.1\text{-}5a_{true}} \vee rule_{6.6.1\text{-}5a_{false}}$$
$$rule_{6.6.1\text{-}5b} \triangleq rule_{6.6.1\text{-}5b_{true}} \vee rule_{6.6.1\text{-}5b_{false}}$$
$$rule_{6.6.1\text{-}5a_{true}} \triangleq (w_{6.6.1\text{-}5a_0} \wedge \circ w_{6.6.1\text{-}5a_1})$$
$$rule_{6.6.1\text{-}5a_{false}} \triangleq (\neg w_{6.6.1\text{-}5a_0} \wedge \circ w_{6.6.1\text{-}5a_2})$$
$$rule_{6.6.1\text{-}5b_{true}} \triangleq (w_{6.6.1\text{-}5b_0} \wedge \circ w_{6.6.1\text{-}5b_1})$$
$$rule_{6.6.1\text{-}5b_{false}} \triangleq (\neg w_{6.6.1\text{-}5b_0} \wedge \circ w_{6.6.1\text{-}5b_2})$$

The following lemmas describe equivalence transformations of rule-based, nested if-then-else constructs.

LEMMA: NestIfThenElseEqv1 (proved at 4, below)

$$\vdash (f_0 \wedge \circ rule_a) \vee (\neg f_0 \wedge \circ rule_b)$$
$$\equiv (f_0 \wedge \circ rule_{a_{true}}) \vee (f_0 \wedge \circ rule_{a_{false}}) \vee (\neg f_0 \wedge \circ rule_{b_{true}}) \vee (\neg f_0 \wedge \circ rule_{b_{false}})$$

LEMMA: NestIfThenElseEqv2 (proved at 6, below)

$$\vdash (f_0 \wedge \circ rule_a) \vee (\neg f_0 \wedge \circ rule_b)$$
$$\equiv (f_0 \wedge \circ f_{a_0} \wedge \circ\circ f_{a_1}) \vee (f_0 \wedge \circ\neg f_{a_0} \wedge \circ\circ f_{a_2})$$
$$\vee (\neg f_0 \wedge \circ f_{b_0} \wedge \circ\circ f_{b_1}) \vee (\neg f_0 \wedge \circ\neg f_{b_0} \wedge \circ\circ f_{b_2})$$

where:
$$\begin{aligned}
rule_a &\triangleq rule_{a_{true}} \vee rule_{a_{false}} \\
rule_b &\triangleq rule_{b_{true}} \vee rule_{b_{false}} \\
rule_{a_{true}} &\triangleq (f_{a_0} \wedge \circ f_{a_1}) \\
rule_{a_{false}} &\triangleq (\neg f_{a_0} \wedge \circ f_{a_2}) \\
rule_{b_{true}} &\triangleq (f_{b_0} \wedge \circ f_{b_1}) \\
rule_{b_{false}} &\triangleq (\neg f_{b_0} \wedge \circ f_{b_2})
\end{aligned}$$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge \circ rule_a) \vee (\neg f_0 \wedge \circ rule_b)$ | premise |
| | $\equiv (f_0 \wedge \circ rule_a) \vee (\neg f_0 \wedge \circ rule_b)$ | |
| 2 | $\equiv (f_0 \wedge \circ(rule_{a_{true}} \vee rule_{a_{false}}))$ $\vee (\neg f_0 \wedge \circ(rule_{b_{true}} \vee rule_{b_{false}}))$ | 1, definitional substitution |
| 3 | $\equiv (f_0 \wedge (\circ rule_{a_{true}} \vee \circ rule_{a_{false}}))$ $\vee (\neg f_0 \wedge (\circ rule_{b_{true}} \vee \circ rule_{b_{false}}))$ | 2, NextOrDistEqv |
| 4 | $\equiv (f_0 \wedge \circ rule_{a_{true}}) \vee (f_0 \wedge \circ rule_{a_{false}})$ $\vee (\neg f_0 \wedge \circ rule_{b_{true}}) \vee (\neg f_0 \wedge \circ rule_{b_{false}})$ | 3, Distribution of $\wedge$ over $\vee$ |
| 5 | $\equiv (f_0 \wedge \circ(f_{a_0} \wedge \circ f_{a_1})) \vee (f_0 \wedge \circ(\neg f_{a_0} \wedge \circ f_{a_2}))$ $\vee (\neg f_0 \wedge \circ(f_{b_0} \wedge \circ f_{b_1})) \vee (\neg f_0 \wedge \circ(\neg f_{b_0} \wedge \circ f_{b_2}))$ | 4, definitional substitution |
| 6 | $\equiv ((f_0 \wedge \circ f_{a_0}) \wedge \circ\circ f_{a_1}) \vee ((f_0 \wedge \circ\neg f_{a_0}) \wedge \circ\circ f_{a_2})$ $\vee ((\neg f_0 \wedge \circ f_{b_0}) \wedge \circ\circ f_{b_1}) \vee ((\neg f_0 \wedge \circ\neg f_{b_0}) \wedge \circ\circ f_{b_2})$ | 5, NextAndDistEqv and propositional logic |

With NestIfThenElseEqv1, a nested if-then-else construct is transformed into an equivalent disjunction of four general-form rules. With NestIfThenElseEqv2, a nested if-then-else construct is transformed to explicitly identify each pair of rule conditions associated the each of the four rule states.

Applying NestIfThenElseEqv2 to (6.6.1-5) yields:

$$\begin{aligned}
&((w_0 \wedge \circ w_{6.6.1\text{-}5a_0}) \wedge \circ\circ w_{6.6.1\text{-}5a_1}) \\
&\vee ((w_0 \wedge \circ\neg w_{6.6.1\text{-}5a_0}) \wedge \circ\circ w_{6.6.1\text{-}5a_2}) \\
&\vee ((\neg w_0 \wedge \circ w_{6.6.1\text{-}5b_0}) \wedge \circ\circ w_{6.6.1\text{-}5b_1}) \\
&\vee ((\neg w_0 \wedge \circ\neg w_{6.6.1\text{-}5b_0}) \wedge \circ\circ w_{6.6.1\text{-}5b_2})
\end{aligned} \qquad (6.6.1\text{-}6)$$

With this transformation, the state associations of the nested if-the-else structure of (6.6.1-5) are clear. With this nested if-the-else structure, each of four state sequences is associated with the satisfaction or non-satisfaction of three conditions defined by the state formulas $w_0$, $w_{6.6.1\text{-}5a_0}$, and $w_{6.6.1\text{-}5b_0}$. Using this model, deeper if-then-else structures can be created as necessary by using additional nesting to incorporate additional conditions and rule states. Because NestIfThenElseEqv1 and NestIfThenElseEqv2 are equivalences, both can be applied to either expand or encapsulate such nested structures as required.

### 6.6.2 While Structures

While structures are a common method in many imperative-programming languages for implementing a conditional loop. Using ITL, Moszkowski (1986) defines a while structure recursively as:

$$\text{while } w_0 \text{ do } w_1 \triangleq \text{if } w_0 \text{ then } (w_1;\text{ while } w_0 \text{ do } w_1) \text{ else empty} \qquad (6.6.2\text{-}1)$$

Applying the Moszkowski (1986) model of the if-then-else structure from (6.6.1-1), (6.6.2-1) is restated as:

$$\text{while } w_0 \text{ do } w_1 \triangleq (w_0 \supset (w_1;\text{ while } w_0 \text{ do } w_1)) \wedge (\neg w_0 \supset \text{empty}) \qquad (6.6.2\text{-}2)$$

Applying the ITL definition of the if-then-else structure from Table 4.3-5, (6.6.1-1) is restated as:

$$\text{while } w_0 \text{ do } w_1 \triangleq (w_0 \wedge (w_1;\text{ while } w_0 \text{ do } w_1)) \vee (\neg w_0 \wedge \text{empty}) \qquad (6.6.2\text{-}3)$$

Cau and Zedan (1997) define the while structure in terms of temporal formulas:

$$\text{while} f_0 \text{ do} f_1 \triangleq ((f_0 \wedge f_1)\,;\text{ while} f_0 \text{ do} f_1) \vee (\neg f_0 \wedge \text{empty}) \qquad (6.6.2\text{-}4)$$

As presented in Table 4.3-5, the while structure in ITL is now defined using chopstar as:

$$\text{while} f_0 \text{ do} f_1 \triangleq (f_0 \wedge f_1)^* \wedge \text{fin } \neg f_0 \qquad (6.6.2\text{-}5)$$

In this form, ITL operator fin denotes that the final subinterval of the interval defined by the while construct does not satisfy the guard $f_0$.

Using the general rule form of this research, the recursion implicit in the while structure is expressed using the if-then structure of (6.6.1-4) as:

$$\text{while} f_0 \text{ do} f_1 \triangleq ((f_0 \wedge \circ f_1) ; \text{while} f_0 \text{ do} f_1) \vee (\neg f_0 \wedge \text{empty}) \qquad (6.6.2\text{-}6)$$

Alternatively, a rule-based while structure is described using chop-star as:

$$\text{while} f_0 \text{ do} f_1 \triangleq (f_0 \wedge \circ f_1)^* \vee (\neg f_0 \wedge \text{empty}) \qquad (6.6.2\text{-}7)$$

## 6.6.3 Indexed For-Loop Structures

Consider the following general indexed for loop:

$$\text{for } A = b \text{ to } c \text{ do} f_1 \qquad (6.6.3\text{-}1)$$

where $A$ is a state variable that can change value over the interval, and $a$ and b are static variables that cannot change in value over the interval. This indexed for loop can be described in terms of ITL using the while structure as:

$$\text{for } A = b \text{ to } c \text{ do} f_1 \triangleq (\circ A = b) ; rule' \qquad (6.6.3\text{-}2)$$

where:

$$rule' \triangleq \text{while } (A \leq c) \text{ do } (f_1 ; \circ A = A + 1)$$

In the form, the index variable $A$ is initialized with the assignment $\circ A = b$ and incremented by 1 after each interval described by $f_1$. This incrementing is achieved with the chopped assignment formula $\circ A = A + 1$. The definition of assignment in ITL is presented in Table 4.3-6. Applying the if-then definition of the while construct presented at (6.6.2-6) and NextChop, the indexed for-loop is described as:

$$\text{for } A = b \text{ to } c \text{ do} f_1 \triangleq (\circ A = b) ; rule' \qquad (6.6.3\text{-}3)$$

where:

$$rule' \triangleq (((A \leq c) \wedge \circ f_1 ; \circ A = A + 1) ; rule') \vee (\neg(A \leq c) \wedge \mathrm{empty})$$

## 6.7 Some Other Interesting Rules

In this section, several interesting instantiations of the general-form rule are examined – interesting in that these simple rules unambiguously capture and express a single fundamental concept.

### 6.7.1 Excluding a Rule State with Negation

Specific rule states can be excluded with negation, as demonstrated in the following rule:

$$f_0 \wedge \circ\neg f_1 \qquad (6.7.1\text{-}1)$$

(6.7.1-1) is a maximally nondeterministic rule, because the satisfaction of this rule will allow the system to exhibit in the next state any valid state sequences except those satisfying $f_1$.

Such a maximally nondeterministic rule is extremely expressive and therefore very valuable in specific circumstances. Consider a set of state sequences described by $f_0$ that are extremely undesirable or troublesome. A simple 'get out of trouble' rule can be formed as:

$$f_0 \wedge \circ\neg f_0 \qquad (6.7.1\text{-}2)$$

Under this rule, if the system exhibits a trouble state sequence described as $f_0$, this rule specifies that the system be moved in the next state to any valid state sequence other than one satisfying $f_0$, thereby moving the system out of the troublesome state sequence associated with $f_0$. Whereas the details of how a new state sequence satisfying $\neg f_0$ is to be chosen are important in the refinement of this rule and ultimately the final implementation of the system, this rule expresses clearly what is of critical importance regarding reasoning about the system – in this case moving the system out of a troublesome state immediately. Although such a simple representation may seem trivial at first glance, it does succinctly and unambiguously express the intended notion – if the system is in the undesirable state described by $f_0$, get out of that undesirable state immediately. To that end, such a rule-base representation in this minimal form achieves what Dijkstra (1976) calls the "clear separation" between the mathematical concerns about desired states and the specific engineering and implementation concerns regarding how these states are achieved.

### 6.7.2 Enforcement of Specific Criteria

Consider the following simple rule:

$$\neg f_0 \wedge \circ f_0 \tag{6.7.2-1}$$

Under this rule, if the system state does not meet the criteria specified by $f_0$, then the next state sequence is required to meet these criteria. As with the example in the preceding section, this simple rule succinctly and unambiguously expresses the intended notion – if a system does not meet the criteria expressed by $f_0$, then require that the next state meet those criteria.

### 6.7.3 System Inverter

An interesting rule variant can be formed by combining the concepts of Sections 6.7.1 and 6.7.1, as demonstrated in (6.7.1-2) and (6.7.2-1), into the following total rule system:

$$(f_0 \wedge \circ\neg f_0) \vee (\neg f_0 \wedge \circ f_0) \tag{6.7.3-1}$$

In this form, (6.7.3-1) describes a system inverter relative to the state sequence specified by $f_0$. If the system state sequence satisfies $f_0$, then the next state sequence must not, and if the system state sequence does not satisfy $f_0$, then the next state sequence must.

### 6.7.4 Identity Rule

An identity rule leaves the system state unchanged. A simple example of an identity rule is:

$$f_0 \wedge \circ f_{unchanged} \tag{6.7.4-1}$$

If the rule condition $f_0$ is satisfied, then the system state remains in next state sequence. With regard to typical programming constructs, the most common implementation or use of an identity rule is as the non-satisfaction half of a total rule where the rule condition $f_0$ of (6.7.4-1) is instantiated with $\neg f_{condition}$ to form $\neg f_{condition} \wedge \circ f_{unchanged}$.

That such an identity rule as expressed in (6.7.4-1) functions as the programming construct skip relative to the satisfaction of $f_0$ is consistent with the definition of the ITL next operator $\circ$. Referencing the definition of of the ITL next operator $\circ$ as presented in Table 4.3-4, $\circ f_{unchanged}$ may be described as skip ; $f_{unchanged}$. Substituting, (6.7.3-1) can be read as $f_0 \wedge$ skip ; $f_{unchanged}$.

### 6.7.5 Any Possible Rule State

Consider the following rule:

$$f_0 \wedge \circ true \qquad (6.7.5\text{-}1)$$

This rule can be satisfied by the satisfaction of the rule condition $f_0$ and by any valid next state sequence, as the formula *true* describes all states and is therefore satisfied by any state sequence. With respect to finite state machines, Hartmanis and Stearns (1966) described this as a "don't care" condition. Such a rule may be relevant if the system designer does not care what the resulting system state is. Possible reasons for the use of such a general rule are that a logical placeholder is needed, that the current system state may be ignored, or that the system state may be reset by some subsequent action.

# Chapter 7

# Analysis of Rules in Models and Specifications

In this chapter, the formal rule extraction framework of Chapter 3, the formal temporal rule model of Chapter 4, and the rule algebra of Chapters 5 and 6 are applied to the extraction of rules from a variety of existing systems and the analysis of those rules. In Section 7.1, rules are extracted from an existing finite state machine; these extracted rules are then used to identify the state sequence that results from the application of an example input sequence to that machine. In Section 7.2, rules are extracted from a detailed formal specification; with these extracted rules, alternative formal transformations are presented, thereby allowing a formal, rule-based analysis of the original specification. In Section 7.3, statecharts are investigated within the context of the formal rule model and the corresponding rule algebra as developed in the research; generic visual formalisms of various rule-based coding paradigms are developed and the rules extracted in Section 7.2 are represented as statecharts.

## 7.1 Analysis of Rules from a Finite State Machine

In this section, rules are extracted from an existing finite state machine and analyzed. Consider the finite state machine depicted in Figure 7.1-1 from STRL (2003).

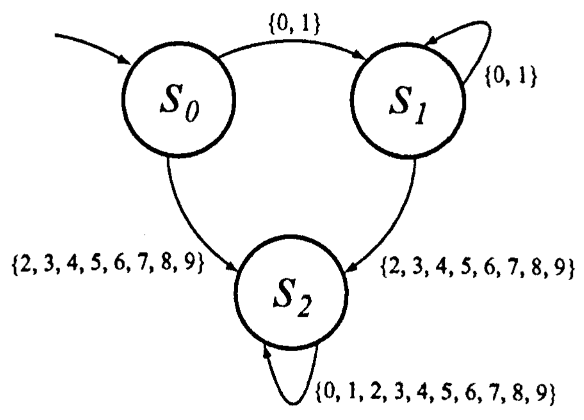


Figure 7.1-1: Three-State Finite State Machine

The system consists of three states ($s_0$, $s_1$, and $s_2$), and three state formulas, $w_0$, $w_1$, and $w_2$, are used to describe this system, where $s_0 \models w_0$, $s_1 \models w_1$, and $s_2 \models w_2$. For the purposes of describing this system, $x$ is the next symbol read from the input and the acceptable input alphabet is the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. For the purposes of this analysis and consistent with the transitions depicted in Figure 7.1-1, this input alphabet is divided into two sets, $a_1 = \{0, 1\}$ and $a_2 = \{2, 3, 4, 5, 6, 7, 8, 9\}$. The state transitions included in this system can be described based on the starting state, the recognized input, and the ending state. Within the context of the general form rule, the starting state and the recognized input are the rule conditions and the ending state is the rule state. These five state transitions, described as rules and organized based on the state satisfying the corresponding rule condition, are as follows:

$$s_0 \qquad (w_0 \wedge x \in a_1) \wedge \circ w_1 \tag{7.1-1a}$$
$$s_0 \qquad (w_0 \wedge x \in a_2) \wedge \circ w_2 \tag{7.1-1b}$$
$$s_1 \qquad (w_1 \wedge x \in a_1) \wedge \circ w_1 \tag{7.1-1c}$$
$$s_1 \qquad (w_1 \wedge x \in a_2) \wedge \circ w_2 \tag{7.1-1d}$$
$$s_2 \qquad (w_2 \wedge (x \in a_1 \vee x \in a_2)) \wedge \circ w_2 \tag{7.1-1e}$$

This set of five individual rules describes all five transitions in the finite state machine, and can be combined disjunctively to describe the entire system as:

$$\begin{aligned}
&(w_0 \wedge x \in a_1) \wedge \circ w_1 \\
&\vee (w_0 \wedge x \in a_2) \wedge \circ w_2 \\
&\vee (w_1 \wedge x \in a_1) \wedge \circ w_1 \\
&\vee (w_1 \wedge x \in a_2) \wedge \circ w_2 \\
&\vee (w_2 \wedge (x \in a_1 \vee x \in a_2)) \wedge \circ w_2
\end{aligned} \tag{7.1-2}$$

This rule system can be assessed by considering the individual rules that share a common rule state. Consider the following rule-pair from (7.1-2) that shares the common formula $w_1$ for the rule state:

$$((w_0 \wedge x \in a_1) \wedge \circ w_1) \vee ((w_1 \wedge x \in a_1) \wedge \circ w_1) \tag{7.1-3}$$

Applying CommonRuleStateEqv and propositional logic to (7.1-3) yields the following equivalent form:

$$((w_0 \vee w_1) \wedge x \in a_1) \wedge \circ w_1 \qquad (7.1\text{-}4)$$

Consider the following rule-pair from (7.1-2) that shares the common formula $w_2$ for the rule state:

$$((w_0 \wedge x \in a_2) \wedge \circ w_2) \vee ((w_1 \wedge x \in a_2) \wedge \circ w_2) \qquad (7.1\text{-}5)$$

Applying CommonRuleStateEqv and propositional logic to (7.1-5) yields the following equivalent form:

$$((w_0 \vee w_1) \wedge x \in a_2) \wedge \circ w_2 \qquad (7.1\text{-}6)$$

With these equivalent transformations, (7.1-2) is transformed by substituting (7.1-4) and (7.1-6) for the rule pairs considered in (7.1-3) and (7.1-5), respectively, to yield:

$$\begin{aligned}&(((w_0 \vee w_1) \wedge x \in a_1) \wedge \circ w_1)\\ &\vee ((w_0 \vee w_1) \wedge x \in a_2) \wedge \circ w_2)\\ &\vee ((w_2 \wedge (x \in a_1 \vee x \in a_2)) \wedge \circ w_2)\end{aligned} \qquad (7.1\text{-}7)$$

With (7.1-7), the finite state machine depicted in Figure 7.1-1, including the corresponding five transitions presented in (7.1-1a) through (7.1-1e), is described by three general-form rules.

Given that two component rules included in (7.1-7) share a common rule state described by $w_2$, a further simplification is possible. Consider the following rule-pair from (7.1-7) that shares the common formula $w_2$ for the rule state:

$$((w_0 \vee w_1) \wedge x \in a_2) \wedge \circ w_2) \vee ((w_2 \wedge (x \in a_1 \vee x \in a_2)) \wedge \circ w_2) \qquad (7.1\text{-}8)$$

Applying CommonRuleStateEqv to (7.1-8) yields the following equivalent form:

$$(((w_0 \vee w_1) \wedge x \in a_2) \vee (w_2 \wedge (x \in a_1 \vee x \in a_2))) \wedge \circ w_2 \tag{7.1-9}$$

Applying propositional logic to (7.1-9) yields the following equivalent form:

$$(((w_0 \vee w_1 \vee w_2) \wedge x \in a_2) \vee (w_2 \wedge x \in a_1)) \wedge \circ w_2 \tag{7.1-10}$$

With these equivalent transformations, (7.1-7) is transformed by substituting (7.1-10) for the rule pairs considered in (7.1-8) to yield:

$$\begin{aligned}&(((w_0 \vee w_1) \wedge x \in a_1) \wedge \circ w_1)\\ &\vee ((((w_0 \vee w_1 \vee w_2) \wedge x \in a_2) \vee (w_2 \wedge x \in a_1)) \wedge \circ w_2)\end{aligned} \tag{7.1-11}$$

With (7.1-11), the finite state machine depicted in Figure 7.1-1, including the corresponding five transitions presented in (7.1-1a) through (7.1-1e), is described by two general-form rules.

As depicted in Figure 7.1-1, the finite state machine is initially in $s_0$. Therefore, the initial behavior of this finite state machine prior to any input can be described as:

$$w_0 \tag{7.1-12}$$

Letting $rule_{7.1\text{-}11}$ represent the rule system presented in (7.1-11), the behavior of this finite state machine in response to a single input from the input alphabet can be described using the ITL operator chop as:

$$w_0\, ;\, rule_{7.1\text{-}11} \tag{7.1-13}$$

The behavior of this finite state machine in response to two inputs from the input alphabet can be described as:

$$w_0\, ;\, rule_{7.1\text{-}11}\, ;\, rule_{7.1\text{-}11} \tag{7.1-14}$$

For an infinite series of inputs from the input alphabet, the behavior of this finite state machine can be described using ITL chop-star operator as:

$$w_0 \,;\, rule_{7.1\text{-}11}^{*} \qquad (7.1\text{-}15)$$

Alternatively, rule composition based on the general rule form can be used to describe the behavior of this system to multiple inputs. For this description, consider the three-rule disjunctive description previously presented in (7.1-7):

$$\begin{aligned}&(((w_0 \vee w_1) \wedge x \in a_1) \wedge \circ w_1)\\ &\vee ((w_0 \vee w_1) \wedge x \in a_2) \wedge \circ w_2)\\ &\vee ((w_2 \wedge (x \in a_1 \vee x \in a_2)) \wedge \circ w_2)\end{aligned} \qquad (7.1\text{-}7)$$

Letting (7.1-7) be represented by $rule_{7.1\text{-}7}$, a longer state sequence is described by composing $rule_{7.1\text{-}7}$ with itself using the general rule form:

$$rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7} \qquad (7.1\text{-}16)$$

Using the general rule form, (7.1-16) is composed with itself to describe even longer state sequences:

$$(rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7}) \wedge \circ(rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7}) \qquad (7.1\text{-}17)$$

(7.1-17) is transformed using TwoSeqRuleEqv1 (from Section 5.6.1) and yields the equivalent form:

$$rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7} \wedge \circ\circ rule_{7.1\text{-}7} \qquad (7.1\text{-}18)$$

Alternatively, (7.1-17) is transformed using TwoSeqRuleEqv2 (from Section 5.6.1) and yields the equivalent form:

$$rule_{7.1\text{-}7} \wedge \circ(rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7}) \qquad (7.1\text{-}19)$$

As previously demonstrated in Section 5.6.1, the forms of (7.1-17), (7.1-18), and (7.1-19) are equivalent.

Considering the repetitive forms presented in (7.1-18) and (7.1-19), recursion can be used to describe finite state machine behaviors. For multiple inputs, the behavior of the finite state machine presented in Figure 7.1-1 is defined recursively as:

$$rule_{FSM} \triangleq rule_{7.1\text{-}7} \wedge \circ rule_{FSM} \qquad (7.1\text{-}20)$$

As a demonstration of the use of this recursive rule, $rule_{FSM}$ is instantiated using the definition of $rule_{FSM}$ as:

$$rule_{FSM} \equiv rule_{7.1\text{-}7} \wedge \circ(rule_{7.1\text{-}7} \wedge \circ rule_{FSM}) \qquad (7.1\text{-}21)$$

Applying NextAndDistEqv to (7.1-21) yields:

$$rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7} \wedge \circ\circ rule_{FSM} \qquad (7.1\text{-}22)$$

Applying the definition of $rule_{FSM}$ at (7.1-20) to (7.1-22) yields:

$$rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7} \wedge \circ\circ(rule_{7.1\text{-}7} \wedge \circ rule_{FSM}) \qquad (7.1\text{-}23)$$

Applying NextAndDistEqv to (7.1-23) yields:

$$rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7} \wedge \circ\circ rule_{7.1\text{-}7} \wedge \circ\circ\circ rule_{FSM} \qquad (7.1\text{-}24)$$

Given the recursive form of $rule_{FSM}$, $rule_{FSM}$ can be used to describe an infinite behavior associated with the finite state machine presented in Figure 7.1-1. Alternatively, the finite behavior associated the finite state machine (i.e., the behavior associated with a finite input sequence) can be described by applying propositional logic to the infinite, recursive description. For example, the application of propositional logic (conjunction elimination) to (7.1-24) yields:

$$rule_{7.1\text{-}7} \wedge \circ rule_{7.1\text{-}7} \wedge \circ\circ rule_{7.1\text{-}7} \qquad (7.1\text{-}25)$$

With the transformation presented at (7.1-25), the recursive definition of the system presented at (7.1-20) can be easily manipulated to yield the same description of the system behavior previously presented at (7.1-18).

In the demonstration that follows, $rule_{FSM}$ is used to identify the specific sequence of states that results from a specific input. For this analysis, a minor algebraic simplification is made to the various rule representations to transform terms containing the set membership operator $\in$. As originally defined, input to the system can be any of the ten digits defined by the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. In the original system description of (7.1-1a) through (7.1-1e) and in the corresponding $rule_{7.1\text{-}7}$ and $rule_{FSM}$, input associated with each specific transition is described in terms of x such that $x \in a_1$ or $x \in a_2$, where $a_1 = \{0, 1\}$ and $a_2 = \{2, 3, 4, 5, 6, 7, 8, 9\}$. Alternative terms are defined such that:

$$y \triangleq x \in a_1 \tag{7.1-26a}$$

$$\neg y \triangleq x \in a_2 \tag{7.1-26b}$$

Applying these definitions to $rule_{7.1\text{-}7}$ yields:

$$\begin{aligned}&(((w_0 \vee w_1) \wedge y) \wedge \circ w_1)\\ &\vee ((w_0 \vee w_1) \wedge \neg y) \wedge \circ w_2)\\ &\vee ((w_2 \wedge (y \vee \neg y)) \wedge \circ w_2)\end{aligned} \tag{7.1-27}$$

Applying propositional logic to (7.1-27) to eliminate the tautology $(y \vee \neg y)$ yields:

$$\begin{aligned}&(((w_0 \vee w_1) \wedge y) \wedge \circ w_1)\\ &\vee ((w_0 \vee w_1) \wedge \neg y) \wedge \circ w_2)\\ &\vee (w_2 \wedge \circ w_2)\end{aligned} \tag{7.1-28}$$

Henceforth, (7.1-28) is described as $rule_{7.1\text{-}28}$. The definition of $rule_{FSM}$ is updated such that:

$$rule_{FSM} \triangleq rule_{7.1\text{-}28} \wedge \circ rule_{FSM} \tag{7.1-29}$$

Instantiating the definition (7.1-29) with itself yields:

$$rule_{FSM} \equiv rule_{7.1\text{-}28} \wedge \circ(rule_{7.1\text{-}28} \wedge \circ rule_{FSM}) \qquad (7.1\text{-}30)$$

In the following demonstration, the input sequence 012 is tested against $rule_{FSM}$ to determine the system response. In terms of the original input variable $x$, the sequence is described as formula $x \wedge \circ x \wedge \circ\circ x$, where $x = 0$, $\circ x = 1$, and $\circ\circ x = 2$. Using the algebraic transformations described at (7.1-26a) and (7.1-26b), the input sequence 012 is represented as the formula $y \wedge \circ y \wedge \circ\circ\neg y$. Decomposing this formula, y represents $x \in a_1$ which holds for $x = 0$. With the next term, $\circ y$ represents $\circ x \in a_1$ which holds for $\circ x = 1$. And the final term, $\circ\circ\neg y$ represents $\circ\circ x \in a_2$, which holds for $\circ\circ x = 2$.

In this demonstration, four premises are asserted. In the first premise, a finite state machine exists and is described by $rule_{FSM}$ and the associated definitions. In the second premise, the input stream to be processed by the finite state machine is described by the temporal formula $y \wedge \circ y \wedge \circ\circ\neg y$. In the third premise, the finite state machine is started in a state satisfying $w_0$. In the fourth premise, the relative uniqueness of the three formulas describing the three states of finite state machine is asserted.

The processing of the input stream 012 with the finite state machine described by $rule_{FSM}$ is as follows:

| | | |
|---|---|---|
| 1 | $rule_{FSM}$ | premise |
| | where: | |
| | $rule_{FSM} \triangleq rule_{7.1\text{-}28} \wedge \circ rule_{FSM}$ | |
| | $rule_{7.1\text{-}28} \triangleq (((w_0 \vee w_1) \wedge y) \wedge \circ w_1)$ | |
| | $\vee (((w_0 \vee w_1) \wedge \neg y) \wedge \circ w_2)$ | |
| | $\vee (w_2 \wedge \circ w_2)$ | |
| 2 | $y \wedge \circ y \wedge \circ\circ\neg y$ | premise |
| 3 | $w_0$ | premise |
| 4 | $(w_0 \supset (\neg w_1 \wedge \neg w_2))$ | premise |
| | $\wedge (w_1 \supset (\neg w_0 \wedge \neg w_2))$ | |
| | $\wedge (w_2 \supset (\neg w_0 \wedge \neg w_1))$ | |
| 5 | $rule_{7.1\text{-}28} \wedge \circ rule_{FSM}$ | 1, definition substitution |
| 6 | $rule_{7.1\text{-}28}$ | 5, $\wedge$ elimination |
| 7 | $y$ | 2, $\wedge$ elimination |

| | | |
|---|---|---|
| 8 | $w_0 \supset (\neg w_1 \wedge \neg w_2)$ | 4, $\wedge$ elimination |
| 9 | $\neg w_1 \wedge \neg w_2$ | 3, 8, MP |
| 10 | $\neg w_1$ | 9, $\wedge$ elimination |
| 11 | $\neg w_2$ | 9, $\wedge$ elimination |
| 12 | $(((w_0 \vee w_1) \wedge y) \wedge \circ w_1)$<br>$\vee (((w_0 \vee w_1) \wedge \neg y) \wedge \circ w_2)$<br>$\vee (w_2 \wedge \circ w_2)$ | 6, definition substitution |
| 13 | $((\mathit{true} \vee \mathit{false}) \wedge \mathit{true} \wedge \circ w_1)$<br>$\vee ((\mathit{true} \vee \mathit{false}) \wedge \mathit{false} \wedge \circ w_2)$<br>$\vee (\mathit{false} \wedge \circ w_2)$ | 3, 7, 10, 11, 12, prop. logic |
| 14 | $(\mathit{true} \wedge \mathit{true} \wedge \circ w_1)$<br>$\vee (\mathit{true} \wedge \mathit{false} \wedge \circ w_2)$<br>$\vee (\mathit{false})$ | 13, unit of $\vee$, zero of $\wedge$ |
| 15 | $\circ w_1$ | 14, unit of $\wedge$, zero of $\wedge$, unit of $\vee$ |
| 16 | $\circ y$ | 2, $\wedge$ elimination |
| 17 | $w_1 \supset (\neg w_0 \wedge \neg w_2)$ | 4, $\wedge$ elimination |
| 18 | $\circ w_1 \supset \circ(\neg w_0 \wedge \neg w_2)$ | 17, ITL (NextImpNext) |
| 19 | $\circ w_1 \supset \circ\neg w_0 \wedge \circ\neg w_2$ | 18, NextAndDistEqv |
| 20 | $\circ\neg w_0 \wedge \circ\neg w_2$ | 11, 19, MP |
| 21 | $\circ\neg w_0$ | 20, $\wedge$ elimination |
| 22 | $\circ\neg w_2$ | 20, $\wedge$ elimination |
| 23 | $\circ \mathit{rule}_{FSM}$ | 5, $\wedge$ elimination |
| 24 | $\circ(\mathit{rule}_{7.1\text{-}28} \wedge \circ \mathit{rule}_{FSM})$ | 23, definition substitution |
| 25 | $\circ \mathit{rule}_{7.1\text{-}28} \wedge \circ\circ \mathit{rule}_{FSM}$ | 24, NextAndDistEqv |
| 26 | $\circ \mathit{rule}_{7.1\text{-}28}$ | 25, $\wedge$ elimination |
| 27 | $\circ((((w_0 \vee w_1) \wedge y) \wedge \circ w_1)$<br>$\vee (((w_0 \vee w_1) \wedge \neg y) \wedge \circ w_2)$<br>$\vee (w_2 \wedge \circ w_2))$ | 26, definition substitution |
| 28 | $\circ(((w_0 \vee w_1) \wedge y) \wedge \circ w_1)$<br>$\vee \circ(((w_0 \vee w_1) \wedge \neg y) \wedge \circ w_2)$<br>$\vee \circ(w_2 \wedge \circ w_2)$ | 27, NextOrDistEqv |
| 29 | $(\circ((w_0 \vee w_1) \wedge y) \wedge \circ\circ w_1)$<br>$\vee (\circ((w_0 \vee w_1) \wedge \neg y) \wedge \circ\circ w_2)$<br>$\vee (\circ w_2 \wedge \circ\circ w_2)$ | 28, NextAndDistEqv |
| 30 | $((\circ(w_0 \vee w_1) \wedge \circ y) \wedge \circ\circ w_1)$<br>$\vee ((\circ(w_0 \vee w_1) \wedge \circ\neg y) \wedge \circ\circ w_2)$<br>$\vee (\circ w_2 \wedge \circ\circ w_2)$ | 29, NextAndDistEqv |

| | | |
|---|---|---|
| 31 | $((\bigcirc w_0 \vee \bigcirc w_1) \wedge \bigcirc y \wedge \bigcirc\bigcirc w_1)$ | 30, NextOrDistEqv |
| | $\vee\ ((\bigcirc w_0 \vee \bigcirc w_1) \wedge \bigcirc\neg y \wedge \bigcirc\bigcirc w_2)$ | |
| | $\vee\ (\bigcirc w_2 \wedge \bigcirc\bigcirc w_2)$ | |
| 32 | $((\mathit{false} \vee \mathit{true}) \wedge \mathit{true} \wedge \bigcirc\bigcirc w_1)$ | 15, 16, 21, 23, 31, prop. logic |
| | $\vee\ ((\mathit{false} \vee \mathit{true}) \wedge \mathit{false} \wedge \bigcirc\bigcirc w_2)$ | |
| | $\vee\ (\mathit{false} \wedge \bigcirc\bigcirc w_2)$ | |
| 33 | $(\mathit{true} \wedge \mathit{true} \wedge \bigcirc\bigcirc w_1)$ | 32, unit of $\vee$, zero of $\wedge$ |
| | $\vee\ (\mathit{true} \wedge \mathit{false} \wedge \bigcirc\bigcirc w_2)$ | |
| | $\vee\ (\mathit{false})$ | |
| 34 | $\bigcirc\bigcirc w_1$ | 33, unit of $\wedge$, zero of $\wedge$, unit of $\vee$ |
| 35 | $\bigcirc\bigcirc\neg y$ | 2, $\wedge$ elimination |
| 36 | $w_1 \supset (\neg w_0 \wedge \neg w_2)$ | 4, $\wedge$ elimination |
| 37 | $\bigcirc w_1 \supset \bigcirc(\neg w_0 \wedge \neg w_2)$ | 36, ITL (NextImpNext) |
| 38 | $\bigcirc w_1 \supset (\bigcirc\neg w_0 \wedge \bigcirc\neg w_2)$ | 37, NextAndDistEqv |
| 39 | $\bigcirc\bigcirc w_1 \supset \bigcirc(\bigcirc\neg w_0 \wedge \bigcirc\neg w_2)$ | 38, ITL (NextImpNext) |
| 40 | $\bigcirc\bigcirc w_1 \supset (\bigcirc\bigcirc\neg w_0 \wedge \bigcirc\bigcirc\neg w_2)$ | 39, NextAndDistEqv |
| 41 | $\bigcirc\bigcirc\neg w_0 \wedge \bigcirc\bigcirc\neg w_2$ | 34, 40, MP |
| 42 | $\bigcirc\bigcirc\neg w_0$ | 41, $\wedge$ elimination |
| 43 | $\bigcirc\bigcirc\neg w_2$ | 41, $\wedge$ elimination |
| 44 | $\bigcirc\bigcirc \mathit{rule}_{FSM}$ | 25, $\wedge$ elimination |
| 45 | $\bigcirc\bigcirc(\mathit{rule}_{7.1\text{-}28} \wedge \bigcirc\mathit{rule}_{FSM})$ | 44, definition substitution |
| 46 | $\bigcirc(\bigcirc\mathit{rule}_{7.1\text{-}28} \wedge \bigcirc\bigcirc\mathit{rule}_{FSM})$ | 45, NextAndDistEqv |
| 47 | $\bigcirc\bigcirc\mathit{rule}_{7.1\text{-}28} \wedge \bigcirc\bigcirc\bigcirc\mathit{rule}_{FSM}$ | 46, NextAndDistEqv |
| 48 | $\bigcirc\bigcirc\mathit{rule}$ | 47, $\wedge$ elimination |
| 49 | $\bigcirc\bigcirc((((w_0 \vee w_1) \wedge y) \wedge \bigcirc w_1)$ | 48, definition substitution |
| | $\vee\ (((w_0 \vee w_1) \wedge \neg y) \wedge \bigcirc w_2)$ | |
| | $\vee\ (w_2 \wedge \bigcirc w_2))$ | |
| 50 | $\bigcirc(\bigcirc(((w_0 \vee w_1) \wedge y) \wedge \bigcirc w_1)$ | 49, NextOrDistEqv |
| | $\vee\ \bigcirc(((w_0 \vee w_1) \wedge \neg y) \wedge \bigcirc w_2)$ | |
| | $\vee\ \bigcirc(w_2 \wedge \bigcirc w_2))$ | |
| 51 | $\bigcirc((\bigcirc((w_0 \vee w_1) \wedge y) \wedge \bigcirc\bigcirc w_1)$ | 50, NextAndDistEqv |
| | $\vee\ (\bigcirc((w_0 \vee w_1) \wedge \neg y) \wedge \bigcirc\bigcirc w_2)$ | |
| | $\vee\ (\bigcirc w_2 \wedge \bigcirc\bigcirc w_2))$ | |
| 52 | $\bigcirc((\bigcirc(w_0 \vee w_1) \wedge \bigcirc y \wedge \bigcirc\bigcirc w_1)$ | 51, NextAndDistEqv |
| | $\vee\ (\bigcirc(w_0 \vee w_1) \wedge \bigcirc\neg y \wedge \bigcirc\bigcirc w_2)$ | |
| | $\vee\ (\bigcirc w_2 \wedge \bigcirc\bigcirc w_2))$ | |

| | | |
|---|---|---|
| 53 | $\circ(((\circ w_0 \vee \circ w_1) \wedge \circ y \wedge \circ\circ w_1)$<br>$\vee ((\circ w_0 \vee \circ w_1) \wedge \circ\neg y \wedge \circ\circ w_2)$<br>$\vee (\circ w_2 \wedge \circ\circ w_2))$ | 52, NextOrDistEqv |
| 54 | $\circ((\circ w_0 \vee \circ w_1) \wedge \circ y \wedge \circ\circ w_1)$<br>$\vee\, \circ((\circ w_0 \vee \circ w_1) \wedge \circ\neg y \wedge \circ\circ w_2)$<br>$\vee\, \circ(\circ w_2 \wedge \circ\circ w_2)$ | 53, NextOrDistEqv |
| 55 | $(\circ(\circ w_0 \vee \circ w_1) \wedge \circ\circ y \wedge \circ\circ\circ w_1)$<br>$\vee (\circ(\circ w_0 \vee \circ w_1) \wedge \circ\circ\neg y \wedge \circ\circ\circ w_2)$<br>$\vee (\circ\circ w_2 \wedge \circ\circ\circ w_2)$ | 54, NextAndDistEqv |
| 56 | $((\circ\circ w_0 \vee \circ\circ w_1) \wedge \circ\circ y \wedge \circ\circ\circ w_1)$<br>$\vee ((\circ\circ w_0 \vee \circ\circ w_1) \wedge \circ\circ\neg y \wedge \circ\circ\circ w_2)$<br>$\vee (\circ\circ w_2 \wedge \circ\circ\circ w_2)$ | 55, NextOrDistEqv |
| 57 | $((\mathit{false} \vee \mathit{true}) \wedge \mathit{false} \wedge \circ\circ\circ w_1)$<br>$\vee ((\mathit{false} \vee \mathit{true}) \wedge \mathit{true} \wedge \circ\circ\circ w_2)$<br>$\vee (\mathit{false} \wedge \circ\circ\circ w_2)$ | 34, 35, 42, 43, 56, prop. logic |
| 58 | $(\mathit{true} \wedge \mathit{false} \wedge \circ\circ\circ w_1)$<br>$\vee (\mathit{true} \wedge \mathit{true} \wedge \circ\circ\circ w_2)$<br>$\vee (\mathit{false})$ | 57, unit of $\vee$, zero of $\vee$ |
| 59 | $\circ\circ\circ w_2$ | 58, zero of $\wedge$, unit of $\wedge$, unit of $\vee$ |
| 60 | $w_0 \wedge \circ w_1 \wedge \circ\circ w_1 \wedge \circ\circ\circ w_2$ | 3, 15, 34, 59, $\wedge$ introduction |

With this processing of the input sequence 012, subject to the four premises, the resulting state sequence is described at sequent 60 by the temporal formula:

$$w_0 \wedge \circ w_1 \wedge \circ\circ w_1 \wedge \circ\circ\circ w_2 \qquad (7.1\text{-}31)$$

Associating the states that are satisfied by these state formulas, the corresponding state sequence is:

$$s_0\ s_1\ s_1\ s_2 \qquad (7.1\text{-}32)$$

Although relatively lengthy, this analysis is quite straightforward. With each recursive iteration, the verity of each rule element is assessed based on the available information. These verities are then applied to the rule model, and propositional logic is applied to determine which of the future states described by the rule model is *true*. For example, in the first iteration, $w_0$ is *true* by premise; and $y$, isolated using propositional logic at sequent 7, is *true* also by premise. Given that $w_0$ is *true*, the relevant portion of the uniqueness assertion premise is isolated using propositional logic, and $\neg w_1$ and $\neg w_2$ are concluded using *modus ponens* and propositional logic (at sequent 8 through 11). The verity of each or their complement are substituted into the rule model and the only formula that can hold is identified (at sequent 12 through 15). With this, $\circ w_1$ is shown to hold, and the next iteration is performed. The process is repeated with minor variations to account for the iterative application of rule model to describe the next terms in the state sequence. The ITL lemma NextImpNext (previously defined in Table 4.3-8) is applied (at sequent 18) to the relevant portion of the uniqueness assertion premise to temporalize it. The definition of $rule_{FSM}$ is applied recursively to define the next possible states in the state sequence (at sequent 24). In all cases, NextOrDistEqv and NextAndDistEqv are applied to distribute the ITL next operator $\circ$ across the formula. With the ITL next operator $\circ$ fully distributed, the verities of all known terms are assigned, the formula reduced using propositional logic, and the only formula that can hold is identified. With this, $\circ\circ w_1$ is shown to hold (at sequent 34). The next iteration is performed using the same logic, and $\circ\circ\circ w_2$ is shown to hold (at sequent 59). For a longer input sequence, this process of iteration and resolution is repeated as needed.

Whereas the finite state machine of Figure 7.1-1 is purposefully limited in scope to facilitate examination, this example does demonstrate that general-form rules can be extracted from the graphical depiction of a finite state machine, and that those general-form rules can be used to effectively describe the behavior of that finite state machine. As demonstrated above, once extracted, these rules can be methodically applied to identify the specific behavior of the machine for a specific input sequence.

## 7.2 Analysis of Rules from a Specification

In this section, rules are extracted from an existing concrete specification and analyzed. The following specification for cash withdrawal from an automatic teller machine, developed by Cau and Zedan (2000), is considered.

var $c, M, Cu, \{Card_j : j \in ac\}, \{Pin_i, A_i : i \in c\}$
$atm\_int$

```
while true do (
while atm_non_empty do (
    wait_customer;
    read_card;
    if card_disabled then take_disabled_card
    else (
        get_pin;
        if max_pin then (
            disable_card ;
            take_disabled_card
            )
            else (
                if pin_exit then take_card_pin_exit
                else (
                    request_money ;
                    if money_exit then take_card_money_exit
                    else (
                          debit_account;
                          take_card_money
                )
            )
        )
    )
);
refill_atm
)
```

Based on a review of the specification and within the context of the rule extraction framework presented in Chapter 3, this specification contains two types of rule structures – the if-then-else structure and the while structure. The specification is processed from the top down (i.e., outside in), analyzing each structure as it occurs, and replacing that structure with the appropriate rule-based formation.

Starting with the outermost or top while structure, the entire specification is represented as:

var $c, M, Cu, \{Card_j : j \in ac\}, \{Pin_i, A_i : i \in c\}$
*atm_int*
$rule_{7.2\text{-}a}$

Within the context of the general rule extraction framework and the stated context for this analysis, the var declaration and the initialization *atm_int* are not rules in that they are not if-then-else or while structures. However, they are included here for completeness. In the above representation, *rule*$_{7.2\text{-}a}$ represents the following portion of the original specification:

```
while true do (
while atm_non_empty do (
     wait_customer;
     read_card;
     if card_disabled then take_disabled_card
     else (
          get_pin;
          if max_pin then (
               disable_card ;
               take_disabled_card
               )
               else (
                    if pin_exit then take_card_pin_exit
                    else (
                         request_money ;
                         if money_exit then take_card_money_exit
                         else (
                              debit_account;
                              take_card_money
                    )
               )
          )
     )
);
refill_atm
)
```

Applying the rule-form definition of the while structure, previously presented at (6.6.2-6), in this portion of the specification, *rule*$_{7.2\text{-}a}$ is defined as:

$$rule_{7.2\text{-}a} \triangleq (true \wedge \circ rule_{7.2\text{-}b} \, ; \text{refill\_atm}) \, ; rule_{7.2\text{-}a}) \\ \vee (\neg true \wedge \text{empty}) \qquad (7.2\text{-}1)$$

Consistent with the disjunctive structure of the rule-form while structure, *rule*$_{7.2\text{-}a}$ is presented as a disjunction of two rules, even though the falsity of the disjunct $\neg true \wedge empty$ is certain. In this form, and consistent with the specification, *rule*$_{7.2\text{-}a}$ describes an infinite sequence.

In (7.2-1), *rule*$_{7.2\text{-}b}$ represents the following portion of the original specification:

```
while atm_non_empty do (
    wait_customer;
    read_card;
    if card_disabled then take_disabled_card
    else (
        get_pin;
        if max_pin then (
            disable_card ;
            take_disabled_card
            )
            else (
                if pin_exit then take_card_pin_exit
                else (
                    request_money ;
                    if money_exit then take_card_money_exit
                    else (
                        debit_account;
                        take_card_money
                )
            )
        )
    )
)
```

Applying the rule-form definition to the top while structure in this remaining portion of the specification, $rule_{7.2\text{-}b}$ is defined as:

$$rule_{7.2\text{-}b} \triangleq \begin{array}{l} ((atm\_non\_empty \\ \wedge \circ \text{wait\_customer} ; \text{read\_card}; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b}) \\ \vee (\neg atm\_non\_empty \wedge \text{empty}) \end{array} \qquad (7.2\text{-}2)$$

In (7.2-2), $rule_{7.2\text{-}c}$ represents the following portion of the original specification:

```
if card_disabled then take_disabled_card
else (
    get_pin;
    if max_pin then (
        disable_card ;
        take_disabled_card
        )
        else (
            if pin_exit then take_card_pin_exit
            else (
                request_money ;
                if money_exit then take_card_money_exit
```

```
                    else (
                         debit_account;
                         take_card_money
                    )
               )
          )
     )
```

Applying the rule-form definition to the top if-then-else structure in this remaining portion of the specification, $rule_{7.2\text{-}c}$ is defined as:

$$rule_{7.2\text{-}c} \triangleq (card\_disabled \wedge \circ\text{take\_disabled\_card}) \vee (\neg card\_disabled \wedge \circ\text{get\_pin} \,;\, rule_{7.2\text{-}d}) \qquad (7.2\text{-}3)$$

In (7.2-3), $rule_{7.2\text{-}d}$ represents the following portion of the original specification:

```
if max_pin then (
     disable_card ;
     take_disabled_card
     )
     else (
          if pin_exit then take_card_pin_exit
          else (
               request_money ;
               if money_exit then take_card_money_exit
               else (
                    debit_account;
                    take_card_money
          )
     )
)
```

Applying the rule-form definition to the top if-then-else structure in this remaining portion of the specification, $rule_{7.2\text{-}d}$ is defined as:

$$rule_{7.2\text{-}d} \triangleq (max\_pin \wedge \circ\text{disable\_card} \,;\, \text{take\_disabled\_card}) \vee (\neg max\_pin \wedge \circ rule_{7.2\text{-}e}) \qquad (7.2\text{-}4)$$

In (7.2-4), $rule_{7.2\text{-}e}$ represents the following portion of the original specification:

```
if pin_exit then take_card_pin_exit
else (
                  request_money ;
                  if $money\_exit$ then take_card_money_exit
                  else (
                        debit_account;
                        take_card_money
             )
       )
```

Applying the rule-form definition to the top if-then-else structure in this remaining portion of the specification, $rule_{7.2\text{-}e}$ is defined as:

$$rule_{7.2\text{-}e} \triangleq \quad (pin\_exit \wedge \circ\text{take\_card\_pin\_exit}) \vee (\neg pin\_exit \wedge \circ\text{request\_money} ; rule_{7.2\text{-}f}) \qquad (7.2\text{-}5)$$

In (7.2-5), $rule_{7.2\text{-}f}$ represents the following portion of the original specification:

```
                  if $money\_exit$ then take_card_money_exit
                  else (
                        debit_account;
                        take_card_money
             )
```

Applying the rule-form definition to the if-then-else structure in this remaining portion of the specification, $rule_{7.2\text{-}f}$ is defined as:

$$rule_{7.2\text{-}f} \triangleq \quad (money\_exit \wedge \circ\text{take\_card\_money\_exit}) \vee (\neg money\_exit \wedge \circ\text{debit\_account} ; \text{take\_card\_money}) \qquad (7.2\text{-}6)$$

The total rule $rule_{7.2\text{-}f}$ contains two disjunctively connected rules reflecting the satisfaction and non-satisfaction of the rule conditions *money_exit* and $\neg$*money_exit.* Because there are no additional rules nested in this rule, the entire specification has been assessed and the rule extraction is complete.

Based on this analysis, six rules have been extracted from the original specification:

$$rule_{7.2\text{-}a} \triangleq ((true \wedge \circ rule_{7.2\text{-}b} ; \text{refill\_atm}) ; rule_{7.2\text{-}a}) \vee (\neg true \wedge \text{empty})$$

$$rule_{7.2\text{-}b} \triangleq ((atm\_non\_empty \wedge \circ\text{wait\_customer} ; \text{read\_card}; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b}) \vee (\neg atm\_non\_empty \wedge \text{empty})$$

$$\begin{aligned}
rule_{7.2\text{-}c} &\triangleq (\mathit{card\_disabled} \wedge \circ\text{take\_disabled\_card}) \\
&\quad \vee (\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} \,;\, rule_{7.2\text{-}d}) \\
rule_{7.2\text{-}d} &\triangleq (\mathit{max\_pin} \wedge \circ\text{disable\_card} \,;\, \text{take\_disabled\_card}) \\
&\quad \vee (\neg \mathit{max\_pin} \wedge \circ rule_{7.2\text{-}e}) \\
rule_{7.2\text{-}e} &\triangleq (\mathit{pin\_exit} \wedge \circ\text{take\_card\_pin\_exit}) \\
&\quad \vee (\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} \,;\, rule_{7.2\text{-}f}) \\
rule_{7.2\text{-}f} &\triangleq (\mathit{money\_exit} \wedge \circ\text{take\_card\_money\_exit}) \\
&\quad \vee (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} \,;\, \text{take\_card\_money})
\end{aligned}$$

In this form, the specification can be analyzed as desired using ITL and other techniques. Because these rules are temporal formulas that describe sequences of states, the behavior of the specified system (that is, the sequence of states that results from the execution of the specification) can be tested and assessed by manipulating these formulas.

In the following analysis, the specific system behavior required to take money from the automatic teller machine is assessed. Using the desired final state sequence take_card_money as the goal (i.e., take_card_money is asserted to be true for this specific analysis), the required state sequence necessary to reach this goal is assembled from the extracted rules by starting with the desired final state sequence and working backwards through the rules. Within this goal-oriented re-assembly, those portions of the rules associated with state sequences that do not lead to the desired final state sequence are discarded, and the remaining rules are re-assembled, leaving only those rules that lead to the state sequence take_card_money.

Consider the extracted rule $rule_{7.2\text{-}f}$ that includes the goal take_card_money:

$$\begin{aligned}
rule_{7.2\text{-}f} &\triangleq (\mathit{money\_exit} \wedge \circ\text{take\_card\_money\_exit}) \\
&\quad \vee (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} \,;\, \text{take\_card\_money})
\end{aligned}$$

The total rule $rule_{7.2\text{-}f}$ contains two disjunctively connected rules. However, only one rule includes the goal take_card_money. Given the imposition of this goal, the rule ($\mathit{money_exit} \wedge \circ$take_card_money_exit) must evaluate *false*. Applying propositional logic, this disjunct is eliminated and the remaining sequence is described as:

$$rule_{7.2\text{-}f_{\text{take\_card\_money}}} \triangleq (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} ; \text{take\_card\_money})$$

Stated another way, $rule_{7.2\text{-}f_{\text{take\_card\_money}}}$ is the only half of the total rule $rule_{7.2\text{-}f}$ that leads to and includes the sequence take_card_money.

Continuing to move backwards through the extracted rules, consider $rule_{7.2\text{-}e}$:

$$rule_{7.2\text{-}e} \triangleq (\mathit{pin\_exit} \wedge \circ\text{take\_card\_pin\_exit}) \vee (\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} ; rule_{7.2\text{-}f})$$

With the imposition of the goal take_card_money, the disjunct ($\mathit{pin_exit} \wedge \circ$take_card_pin_exit) must evaluate *false* because it does not include $rule_{7.2\text{-}f}$ and therefore cannot lead to the goal take_card_money. Applying propositional logic, this disjunct is eliminated and the remaining sequence is described as:

$$rule_{7.2\text{-}e_{\text{take\_card\_money}}} \triangleq (\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} ; rule_{7.2\text{-}f_{\text{take\_card\_money}}})$$

Substituting the definition of $rule_{7.2\text{-}f_{\text{take\_card\_money}}}$ into $rule_{7.2\text{-}e_{\text{take\_card\_money}}}$ yields:

$$rule_{7.2\text{-}e_{\text{take\_card\_money}}} \equiv (\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} ; (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} ; \text{take\_card\_money}))$$

Continuing to work backwards through the extracted rules by eliminating disjuncts that do not lead to the goal and expanding the remaining pruned rules by substituting equivalences yields:

$$rule_{7.2\text{-}d_{\text{take\_card\_money}}} \triangleq (\neg \mathit{max\_pin} \wedge \circ rule_{7.2\text{-}e_{\text{take\_card\_money}}})$$

$$rule_{7.2\text{-}d_{\text{take\_card\_money}}} \equiv (\neg \mathit{max\_pin} \wedge \circ(\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} ; (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} ; \text{take\_card\_money})))$$

$$rule_{7.2\text{-}c_{\text{take\_card\_money}}} \triangleq (\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} ; rule_{7.2\text{-}d_{\text{take\_card\_money}}})$$

$$rule_{7.2\text{-}c_{\text{take\_card\_money}}} \equiv (\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} ; (\neg \mathit{max\_pin} \wedge \circ(\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} ; (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} ; \text{take\_card\_money}))))$$

$$rule_{7.2\text{-}b\ \mathrm{take\_card\_money}} \triangleq (atm\_non\_empty \wedge \circ\mathrm{wait\_customer}\ ;\ \mathrm{read\_card};\ rule_{7.2\text{-}c\ \mathrm{take\_card\_money}})$$

$$rule_{7.2\text{-}b\ \mathrm{take\_card\_money}} \equiv (atm\_non\_empty \wedge \circ\mathrm{wait\_customer}\ ;\ \mathrm{read\_card};\ (\neg card\_disabled \wedge \circ\mathrm{get\_pin}\ ;\ (\neg max\_pin \wedge \circ(\neg pin\_exit \wedge \circ\mathrm{request\_money}\ ;\ (\neg money\_exit \wedge \circ\mathrm{debit\_account}\ ;\ \mathrm{take\_card\_money})))))\ ;\ rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$$

With $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$, the state sequence that leads to take_card_money is identified and described. Because $rule_{7.2\text{-}a}$ is a system rule and not a business/knowledge rule (that is, the rule condition of $rule_{7.2\text{-}a}$ is *true* and therefore reflects no domain-specific knowledge), $rule_{7.2\text{-}a}$ is not included in this analysis.

With this methodical elimination of those elements of the various rules that do not contribute to the defined goal, it is tempting to view the above analysis as a form of backwards slicing. However, unlike most other slicing techniques that target program code, this analysis considers state sequences explicitly described by ITL formulas. Unlike traditional slicing that returns a fragment of code that leads to or from a specific data state, this analysis yields the entire behavior of a system, representing a sequence of states. Unlike traditional slicing where the result is the sliced code, this analysis yields a formal description of a state sequence. Thus, if the above analysis is to be viewed as slicing, it is best described as state sequence slicing.

With this series of reduced rules, reduced in that only the rules associated with take_card_money are included, a variety of analyses are possible. For example, because each of these take_card_money rules are general-form rules consisting of a rule condition and a rule state in the form $f_i \wedge \circ f_j$, the rule conditions necessary to achieve the goal take_card_money are easily identified by assessing the individual rules:

| Rule | Rule Condition |
|---|---|
| $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$ | *atm_non_empty* |
| $rule_{7.2\text{-}c\ \mathrm{take\_card\_money}}$ | $\neg$*card_disabled* |
| $rule_{7.2\text{-}d\ \mathrm{take\_card\_money}}$ | $\neg$*max_pin* |
| $rule_{7.2\text{-}e\ \mathrm{take\_card\_money}}$ | $\neg$*pin_exit* |
| $rule_{7.2\text{-}f\ \mathrm{take\_card\_money}}$ | $\neg$*money_exit* |

Whereas $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$ is a complete description of the state sequence that includes take_card_money, partial state sequences that contribute to the complete sequence described by $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$ can be derived with the application of ITL and the rule algebra of Chapters 5 and 6. These partial state sequences can be used to describe and reason about the behavior of the rule in alternative ways. Consider $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$:

$$
\begin{aligned}
rule_{7.2\text{-}b\ \mathrm{take\_card\_money}} \equiv\ & (atm\_non\_empty \wedge \circ\mathrm{wait\_customer} ; \mathrm{read\_card};\\
& (\neg card\_disabled \wedge \circ\mathrm{get\_pin} ; (\neg max\_pin\\
& \wedge \circ(\neg pin\_exit \wedge \circ\mathrm{request\_money} ; (\neg money\_exit\\
& \wedge \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money}))))) ;\\
& rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}
\end{aligned}
$$

As presented, $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$ considers multiple instances of take_card_money, because it includes a recursive reference to $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$ as the last element of the sequence. However, for the following analysis, only one instance of take_card_money is considered. By limiting this to one instance of take_card_money, the requirements and actions for one customer can be assessed. Therefore, for this analysis, all state sequences after the first instance of take_card_money will be dropped. Assuming the that formula describing the state sequence up to and including take_card_money is known to hold for a given customer, this elimination of the trailing chopped sequence (i.e., the recursive $rule_{7.2\text{-}b\ \mathrm{take\_card\_money}}$) is allowed in ITL based on the semantics of chop. Therefore, using this approach, the state sequence that precedes and includes a single instance of take_card_money is:

$$
\begin{aligned}
& atm\_non\_empty \wedge \circ\mathrm{wait\_customer} ; \mathrm{read\_card} ;\\
& (\neg card\_disabled \wedge \circ\mathrm{get\_pin} ; (\neg max\_pin\\
& \wedge \circ(\neg pin\_exit \wedge \circ\mathrm{request\_money} ;\\
& (\neg money\_exit \wedge \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money}))))
\end{aligned}
\qquad (7.2\text{-}7)
$$

Applying NextAndDistEqv to (7.2-7) yields:

$$
\begin{aligned}
& atm\_non\_empty \wedge \circ\mathrm{wait\_customer} ; \mathrm{read\_card};\\
& (\neg card\_disabled \wedge \circ\mathrm{get\_pin} ; (\neg max\_pin\\
& \wedge \circ\neg pin\_exit \wedge \circ\circ\mathrm{request\_money} ;\\
& (\neg money\_exit \wedge \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money})))
\end{aligned}
\qquad (7.2\text{-}8)
$$

Applying propositional logic (conjunction elimination) to (7.2-8) yields:

$\circ$wait_customer ; read_card;
$(\neg$*card_disabled* $\wedge$ $\circ$get_pin ; $(\neg$*max_pin*
$\wedge$ $\circ\neg$*pin_exit* $\wedge$ $\circ\circ$request_money ;
$(\neg$*money_exit* $\wedge$ $\circ$debit_account ; take_card_money))) (7.2-9)

Applying ITL (AndChopImp) to (7.2-9) yields :

$\circ$wait_customer ; read_card ; $\neg$*card_disabled*
$\wedge$ $\circ$wait_customer ; read_card ; $\circ$get_pin ;
$(\neg$*max_pin* $\wedge$ $\circ\neg$*pin_exit* $\wedge$ $\circ\circ$request_money ;
$(\neg$*money_exit* $\wedge$ $\circ$debit_account ; take_card_money)) (7.2-10)

Applying propositional logic (conjunction elimination) to (7.2-10) yields:

$\circ$wait_customer ; read_card ; $\circ$get_pin ;
$(\neg$*max_pin* $\wedge$ $\circ\neg$*pin_exit* $\wedge$ $\circ\circ$request_money ;
$(\neg$*money_exit* $\wedge$ $\circ$debit_account ; take_card_money))) (7.2-11)

Applying ITL (ChopAndImp) to (7.2-11) yields :

$\circ$wait_customer ; read_card ; $\circ$get_pin ; $\neg$*max_pin*
$\wedge$ $\circ$wait_customer ; read_card ; $\circ$get_pin ; $\circ\neg$*pin_exit*
$\wedge$ $\circ$wait_customer ; read_card ; $\circ$get_pin ; $\circ\circ$request_money ;
$(\neg$*money_exit* $\wedge$ $\circ$debit_account ; take_card_money) (7.2-12)

Applying propositional logic (conjunction elimination) to (7.2-12) yields:

$\circ$wait_customer ; read_card ; $\circ$get_pin ; $\circ\circ$request_money ;
$(\neg$*money_exit* $\wedge$ $\circ$debit_account ; take_card_money) (7.2-13)

Applying ITL (ChopAndImp) to (7.2-13) yields :

$\circ$wait_customer ; read_card ; $\circ$get_pin ;
$\circ\circ$request_money ; $\neg$*money_exit*
$\wedge$ $\circ$wait_customer ; read_card ; $\circ$get_pin ;
$\circ\circ$request_money ; $\circ$debit_account ; take_card_money (7.2-14)

With these transformations, the state sequence leading to take_card_money, as presented in (7.2-7), has been transformed into a series of conjunctively connected chopped state sequences. To demonstrate this, the following conjuncts are extracted from (7.2-8), (7.2-10), (7.2-12), and (7.2-14) using propositional logic:

$$\mathit{atm\_non\_empty} \tag{7.2-15a}$$

$$\circ\text{wait\_customer} ; \text{read\_card} ; \neg\mathit{card\_disabled} \tag{7.2-15b}$$

$$\circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ; \neg\mathit{max\_pin} \tag{7.2-15c}$$

$$\circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ; \circ\neg\mathit{pin\_exit} \tag{7.2-15d}$$

$$\begin{aligned}&\circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ;\\ &\quad \circ\circ\text{request\_money} ; \neg\mathit{money\_exit}\end{aligned} \tag{7.2-15e}$$

$$\begin{aligned}&\circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ;\\ &\quad \circ\circ\text{request\_money} ; \circ\text{debit\_account} ; \text{take\_card\_money}\end{aligned} \tag{7.2-15f}$$

(7.2-15a) through (7.2-15f) are combined using propositional logic to form a single statement:

$$\begin{aligned}&\mathit{atm\_non\_empty}\\ &\wedge \circ\text{wait\_customer} ; \text{read\_card} ; \neg\mathit{card\_disabled}\\ &\wedge \circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ; \neg\mathit{max\_pin}\\ &\wedge \circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ; \circ\neg\mathit{pin\_exit}\\ &\wedge \circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ;\\ &\quad \circ\circ\text{request\_money} ; \neg\mathit{money\_exit}\\ &\wedge \circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ;\\ &\quad \circ\circ\text{request\_money} ; \circ\text{debit\_account} ; \text{take\_card\_money}\end{aligned} \tag{7.2-16}$$

With the above analysis, (7.2-7) has been transformed into (7.2-16). Both describe the rule conditions that must be met prior to and the state sequences that lead to a single instance of take_card_money. Whereas (7.2-7) describes the entire rule form necessary to reach a instance of take_card_money, the state sequences and rule conditions associated with (7.2-7) can be assessed easily using (7.2-16).

For example, because (7.2-16) is a conjunction, all conjuncts must hold. Therefore, the rule condition *atm_non_empty* must hold to achieve take_card_money. In addition to *atm_non_empty*, the state sequence satisfying $\circ$wait_customer ; read_card must hold and then the rule condition $\neg$*card_disabled* must hold. In addition, a state sequence satisfying $\circ$wait_customer ; read_card ; $\circ$get_pin must hold and then the rule condition $\neg$*max_pin* must hold. This continues for all conjuncts, including the final conjunct that specifies that a state sequence satisfying $\circ$wait_customer ; read_card ; $\circ$get_pin ; $\circ\circ$request_money ; $\circ$debit_account must hold before the state sequence take_card_money.

With this individual and collective assessment of the conjuncts that compose (7.2-16), the state sequence and rule condition ordering associated with (7.2-7) are clearly, succinctly, and unambiguously presented. Let there be no misunderstanding – (7.2-16) is not a replacement for (7.2-7). However, (7.2-16) allows a simple and clear presentation of the conditions and behaviors associated with (7.2-7). Given that (7.2-16) is formally derived from (7.2-7), conclusions can be drawn from (7.2-16) with the certain knowledge that those conclusions are applicable to (7.2-7).

In (7.2-16), each of the conjuncts terminate with a rule condition or, in the case of the last conjunct, the target sequence take_card_money. This pattern is explored as another basis for rule transformation, analysis, and understanding, as follows.

The sequences described in (7.2-15a) through (7.2-15f) are derived from (7.2-7), which is asserted to be a finite sequence. Therefore, each of the sequences (7.2-15a) through (7.2-15f) must be finite sequences. Using the ITL sometimes operator $\Diamond$ (also readable as eventually), if two temporal formulas $f_0$ and $f_1$ are finite, then $f_0 ; f_1 \supset \Diamond f_1$ (Cau, 2006, personal communication). Therefore, from (7.2-15b), (7.2-15c), (7.2-15d), and (7.2-15e), the following statements are concluded:

$$\Diamond\neg card\_disabled \qquad (7.2\text{-}17a)$$

$$\Diamond\neg max\_pin \qquad (7.2\text{-}17b)$$

$$\Diamond\circ\neg pin\_exit \qquad (7.2\text{-}17c)$$

$$\Diamond\neg money\_exit \qquad (7.2\text{-}17d)$$

Applying DiamondNextImpDiamond to (7.2-17c) yields:

$$\Diamond\neg pin\_exit \qquad (7.2\text{-}18)$$

Applying propositional logic to (7.2-15a), (7.2-15f), (7.2-17a), (7.2-17b), (7.2-17d), and (7.2-18) yields:

$$\begin{array}{l} \mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge \\ \Diamond\neg\mathit{max\_pin} \wedge \Diamond\neg\mathit{pin\_exit} \wedge \Diamond\neg\mathit{money\_exit} \\ \wedge \circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ; \\ \circ\circ\text{request\_money} ; \circ\text{debit\_account} ; \text{take\_card\_money} \end{array} \qquad (7.2\text{-}19)$$

With the application of parentheses to this conjunctive structure, the fundamental general rule form structure of (7.2-19) is highlighted:

$$\begin{array}{l} (\mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge \\ \quad \Diamond\neg\mathit{max\_pin} \wedge \Diamond\neg\mathit{pin\_exit} \wedge \Diamond\neg\mathit{money\_exit}) \\ \wedge \circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ; \\ \quad \circ\circ\text{request\_money} ; \circ\text{debit\_account} ; \text{take\_card\_money} \end{array} \qquad (7.2\text{-}20)$$

With this series of transformations, the state sequence necessary to achieve the state sequence take_card_money, presented in (7.2-7), has been transformed to the general-form rule presented in (7.2-20). With the form $f_i \wedge \circ f_j$, the rule condition $f_i$ is described as the conjunction $\mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge \Diamond\neg\mathit{max\_pin} \wedge \Diamond\neg\mathit{pin\_exit} \wedge \Diamond\neg\mathit{money\_exit}$ and the rule state $f_j$ is described by the chopped sequence wait_customer ; read_card ; $\circ$get_pin ; $\circ\circ$request_money ; $\circ$debit_account ; take_card_money. Stated another way, to achieve the sequence take_card_money, *atm_non_empty* must hold, $\neg$*card_disabled*, $\neg$*max_pin*, $\neg$*pin_exit*, and $\neg$*money_exit* must eventually hold, and the sequence wait_customer ; read_card ; $\circ$get_pin ; $\circ\circ$request_money ; $\circ$debit_account ; take_card_money must hold. As before, (7.2-20) is not a replacement for (7.2-7), nor is it intended to be. However, it does afford a different analysis path with regard to understanding the conditions and the state sequences that must hold to achieve the take_card_money.

In this analysis, the final transformation incorporating the ITL sometimes operator $\Diamond$, presented at (7.2-20), is based on an extended series of individual transformations starting with (7.2-7). These transformations are generalized with the following lemma:

LEMMA: ChopRuleDiaRuleImp

$\vdash f_0 \wedge \circ f_1 ; (f_2 \wedge \circ f_3)$ and $\vdash f_0 \wedge \circ f_1 ; (f_2 \wedge \circ f_3) \supset \mathit{finite}$ implies
$\vdash f_0 \wedge \Diamond f_2 \wedge \circ f_1 ; \circ f_3$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 \wedge \circ f_1 ; (f_2 \wedge \circ f_3)$ | premise |
| 2 | $f_0 \wedge \circ f_1 ; (f_2 \wedge \circ f_3) \supset \mathit{finite}$ | premise |
| 3 | $\mathit{finite}$ | 1, 2, MP |
| 4 | $\circ f_1 ; (f_2 \wedge \circ f_3)$ | 1, $\wedge$ elimination |
| 5 | $\circ f_1 ; f_2 \wedge \circ f_1 ; \circ f_3$ | 4, ITL (ChopAndImp) |
| 6 | $\circ f_1 ; f_2$ | 5, $\wedge$ elimination |
| 7 | $\Diamond f_2$ | 3, 6, ITL $((\mathit{finite} \wedge f_a ; f_b) \supset \Diamond f_b)$ |
| 8 | $f_0$ | 1, $\wedge$ elimination |
| 9 | $f_0 \wedge \Diamond f_2$ | 7, 8, $\wedge$ introduction |
| 10 | $\circ f_1 ; \circ f_3$ | 5, $\wedge$ elimination |
| 11 | $f_0 \wedge \Diamond f_2 \wedge \circ f_1 ; \circ f_3$ | 9, 10, $\wedge$ introduction |

The use of this lemma to simplify the transformation of a state sequence is demonstrated with the following reanalysis of the reduced version of $\mathit{rule}_{7.2\text{-}b\ \text{take\_card\_money}}$, previously presented at (7.2-7). For convenience, (7.2-7) is reiterated:

$$\begin{aligned}&\mathit{atm\_non\_empty} \wedge \circ\text{wait\_customer} ; \text{read\_card};\\ &(\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} ; (\neg \mathit{max\_pin}\\ &\wedge \circ(\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} ; (\neg \mathit{money\_exit}\\ &\wedge \circ\text{debit\_account} ; \text{take\_card\_money}))))\end{aligned} \qquad (7.2\text{-}7)$$

Applying NextAndDistEqv to (7.2-7) yields:

$$\begin{aligned}&\mathit{atm\_non\_empty} \wedge \circ\text{wait\_customer} ; \text{read\_card};\\ &(\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} ; (\neg \mathit{max\_pin}\\ &\wedge (\circ\neg \mathit{pin\_exit} \wedge \circ\circ\text{request\_money} ; (\neg \mathit{money\_exit}\\ &\wedge \circ\text{debit\_account} ; \text{take\_card\_money}))))\end{aligned} \qquad (7.2\text{-}21)$$

Applying ChopRuleDiaRuleImp to (7.2-21) yields:

$$\begin{aligned}&\mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge\\&\circ\mathrm{wait\_customer} ; \mathrm{read\_card} ; \circ\mathrm{get\_pin} ; (\neg\mathit{max\_pin}\\&\wedge (\circ\neg\mathit{pin\_exit} \wedge \circ\circ\mathrm{request\_money} ; (\neg\mathit{money\_exit}\\&\wedge \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money})))\end{aligned} \tag{7.2-22}$$

Applying ChopRuleDiaRuleImp to (7.2-22) yields:

$$\begin{aligned}&\mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge \Diamond\neg\mathit{max\_pin}\\&\wedge \circ\mathrm{wait\_customer} ; \mathrm{read\_card} ; \circ\mathrm{get\_pin} ;\\&(\circ\neg\mathit{pin\_exit} \wedge \circ\circ\mathrm{request\_money} ; (\neg\mathit{money\_exit}\\&\wedge \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money}))\end{aligned} \tag{7.2-23}$$

Applying ChopRuleDiaRuleImp to (7.2-23) yields:

$$\begin{aligned}&\mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge \Diamond\neg\mathit{max\_pin}\\&\wedge \Diamond\circ\neg\mathit{pin\_exit} \wedge \circ\mathrm{wait\_customer} ; \mathrm{read\_card} ; \circ\mathrm{get\_pin} ;\\&\circ\circ\mathrm{request\_money} ; (\neg\mathit{money\_exit}\\&\wedge \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money})\end{aligned} \tag{7.2-24}$$

Applying ChopRuleDiaRuleImp to (7.2-24) yields:

$$\begin{aligned}&\mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge \Diamond\neg\mathit{max\_pin}\\&\wedge \Diamond\circ\neg\mathit{pin\_exit} \wedge \Diamond\neg\mathit{money\_exit} \wedge\\&\circ\mathrm{wait\_customer} ; \mathrm{read\_card} ; \circ\mathrm{get\_pin} ;\\&\circ\circ\mathrm{request\_money} ; \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money}\end{aligned} \tag{7.2-25}$$

To transform $\Diamond\circ\neg\mathit{pin\_exit}$ to $\Diamond\neg\mathit{pin\_exit}$, propositional logic, and ITL (DiamondNextImpDiamond) are applied to (7.2-25) to yield:

$$\begin{aligned}&\mathit{atm\_non\_empty} \wedge \Diamond\neg\mathit{card\_disabled} \wedge \Diamond\neg\mathit{max\_pin}\\&\wedge \Diamond\neg\mathit{pin\_exit} \wedge \Diamond\neg\mathit{money\_exit} \wedge\\&\circ\mathrm{wait\_customer} ; \mathrm{read\_card} ; \circ\mathrm{get\_pin} ;\\&\circ\circ\mathrm{request\_money} ; \circ\mathrm{debit\_account} ; \mathrm{take\_card\_money}\end{aligned} \tag{7.2-26}$$

With (7.2-26), the transformation is complete because the chopped sequence $\circ$wait_customer ; read_card ; $\circ$get_pin ; $\circ\circ$request_money ; $\circ$debit_account ; take_card_money contains no general form rules of the form $f_i \wedge \circ f_j$. Although the strict and total temporal ordering of (7.2-7) is not preserved with respect to when the various conditions must be met, (7.2-26) does provide an alternative view of the conditions and sequences that comprise the total state sequence leading to take_card_money.

The preceding analyses have focused on only one portion of a set of rule, or more specifically, the one path through the set of rules that leads to a single outcome. In these preceding analyses, this involved selecting only those rules (i.e., the halves of a total rule) that lead to the goal take_card_money. Alternatively, sets of total rules can be analyzed. Analyzing the total rule precludes the necessity of state sequence slicing on a specific outcome. By analyzing the total rules, all possible outcomes can be assessed.

To demonstrate the analysis of an entire set of rules, five of the previously extracted total rules are considered:

$$\begin{aligned}
\mathit{rule}_{7.2\text{-}b} &\triangleq ((\mathit{atm\_non\_empty} \wedge \circ\text{wait\_customer} ; \text{read\_card}; \mathit{rule}_{7.2\text{-}c}) ; \mathit{rule}_{7.2\text{-}b}) \\
&\quad\vee (\neg \mathit{atm\_non\_empty} \wedge \text{empty}) \\
\mathit{rule}_{7.2\text{-}c} &\triangleq (\mathit{card\_disabled} \wedge \circ\text{take\_disabled\_card}) \\
&\quad\vee (\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} ; \mathit{rule}_{7.2\text{-}d}) \\
\mathit{rule}_{7.2\text{-}d} &\triangleq (\mathit{max\_pin} \wedge \circ\text{disable\_card} ; \text{take\_disabled\_card}) \\
&\quad\vee (\neg \mathit{max\_pin} \wedge \circ \mathit{rule}_{7.2\text{-}e}) \\
\mathit{rule}_{7.2\text{-}e} &\triangleq (\mathit{pin\_exit} \wedge \circ\text{take\_card\_pin\_exit}) \\
&\quad\vee (\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} ; \mathit{rule}_{7.2\text{-}f}) \\
\mathit{rule}_{7.2\text{-}f} &\triangleq (\mathit{money\_exit} \wedge \circ\text{take\_card\_money\_exit}) \\
&\quad\vee (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} ; \text{take\_card\_money})
\end{aligned}$$

As previously discussed, $\mathit{rule}_{7.2\text{-}a}$ is always *true* and is implemented to assure the repetitive and non-terminating application of $\mathit{rule}_{7.2\text{-}b}$. Therefore, $\mathit{rule}_{7.2\text{-}a}$ is not considered in this analysis.

In the following analysis, these five rules – $\mathit{rule}_{7.2\text{-}b}$, $\mathit{rule}_{7.2\text{-}c}$, $\mathit{rule}_{7.2\text{-}d}$, $\mathit{rule}_{7.2\text{-}e}$, and $\mathit{rule}_{7.2\text{-}f}$ – are used as premises, and a transformation is derived that describes the rule conditions and rule states associated with these five rules. One assumption is made for this analysis. The rule conditions of these five rules – *atm_non_empty*, *card_disabled*, *money_exit*, *max_pin*, and *pin_exit* – are asserted to be state formulas. Given the expected operation of a typical ATM machine – that the satisfaction of these individual conditions is based on the current state of the system and not some future state or sequence of states – this is a reasonable assertion.

To implement these rule transformations, six lemmas are introduced – AndChopDrop, ChopSwapImp1, ChopSwapImp2, ChopSwapImp3, RuleChopTwoRuleImp, TwoChopRulesImp, and TwoChopRulesImp2. ChopSwapImp1, ChopSwapImp2, and ChopSwapImp3 are used to replace an ITL formula (typically a general form rule) in a chopped sequence with a transformed formula, thereby maintaining the order and associations of the original chopped sequence. AndChopDrop, RuleChopTwoRuleImp, TwoChopRulesImp, and TwoChopRulesImp2 are used to separate and collect the rule conditions and rule states of the two chopped rules and transform them into a single general form rule. TwoChopRulesImp is a generalization of StateTwoChopRulesImp3, previously presented in Section 5.6.2. In a deviation from previously consistent use of the general rule form $f_i \wedge \circ f_j$ in the development of other lemmas that comprise this rule algebra, these lemmas are developed using $f_i \wedge f_j$ to accommodate both the general rule form $f_i \wedge \circ f_j$ and special cases such as $f_i \wedge$ empty. Given that $f_n$ can be instantiated with $\circ f_n$ as needed in these lemmas, this alternative form is fully expressive with regard to the general rule form used throughout this thesis.

LEMMA: AndChopDrop

$\vdash (f_0 \wedge f_1) ; f_2$ implies $\vdash f_0 \wedge f_1 ; f_2$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge f_1) ; f_2$ | premise |
| 2 | $f_0 ; f_2 \wedge f_1 ; f_2$ | 1, ITL (AndChopImp) |
| 3 | $f_0 ; f_2$ | 2, $\wedge$ elimination |
| 4 | $f_0$ | 3, semantics of chop |
| 5 | $f_1 ; f_2$ | 2, $\wedge$ elimination |
| 6 | $f_0 \wedge f_1 ; f_2$ | 4, 5, $\wedge$ introduction |

LEMMA: ChopSwapImp1

$\vdash f_0 ; f_1$ and $\vdash f_1 \supset f_2$ implies $\vdash f_0 ; f_2$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 ; f_1$ | premise |
| 2 | $f_1 \supset f_2$ | premise |
| 3 | $f_0 ; f_1 \supset f_0 ; f_2$ | 2, ITL (RightChopImpChop) |
| 4 | $f_0 ; f_2$ | 1, 3, MP |

LEMMA: ChopSwapImp2

$\vdash f_0 ; f_2$ and $\vdash f_0 \supset f_1$ implies $\vdash f_1 ; f_2$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 ; f_2$ | premise |
| 2 | $f_0 \supset f_1$ | premise |
| 3 | $f_0 ; f_2 \supset f_1 ; f_2$ | 2, ITL (LeftChopImpChop) |
| 4 | $f_1 ; f_2$ | 1, 3, MP |

LEMMA: ChopSwapImp3

$\vdash f_0 ; f_1 ; f_3$ and $\vdash f_1 \supset f_2$ implies $\vdash f_0 ; f_2 ; f_3$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 ; f_1 ; f_3$ | premise |
| 2 | $f_1 \supset f_2$ | premise |
| 3 | $f_1 ; f_3 \supset f_2 ; f_3$ | 2, ITL (LeftChopImpChop) |
| 4 | $f_0 ; f_1 ; f_3 \supset f_0 ; f_2 ; f_3$ | 3, ITL (RightChopImpChop) |
| 5 | $f_0 ; f_2 ; f_3$ | 1, 4, MP |

LEMMA: TwoChopRulesImp

$\vdash (f_0 \wedge f_1) ; (f_2 \wedge f_3)$ implies $\vdash (f_0 ; f_2) \wedge (f_1 ; f_3)$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge f_1) ; (f_2 \wedge f_3)$ | premise |
| 2 | $((f_0 \wedge f_1) ; f_2) \wedge ((f_0 \wedge f_1) ; f_3)$ | 1, ITL (ChopAndImp) |
| 3 | $(f_0 \wedge f_1) ; f_2$ | 2, $\wedge$ elimination |
| 4 | $(f_0 ; f_2) \wedge (\circ f_1 ; f_2)$ | 3, ITL (AndChopImp) |

| | | |
|---|---|---|
| 5 | $(f_0 \wedge f_1) ; f_3$ | 2, $\wedge$ elimination |
| 6 | $(f_0 ; f_3) \wedge (f_1 ; f_3)$ | 5, ITL (AndChopImp) |
| 7 | $(f_0 ; f_2)$ | 4, $\wedge$ elimination |
| 8 | $(f_1 ; f_3)$ | 6, $\wedge$ elimination |
| 9 | $(f_0 ; f_2) \wedge (f_1 ; f_3)$ | 7, 8, $\wedge$ introduction |

LEMMA: TwoChopRulesImp2

$\vdash (f_0 \wedge f_1) ; (f_2 \wedge f_3) ; f_4$ implies $\vdash (f_0 ; f_2) \wedge (f_1 ; f_3 ; f_4)$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge f_1) ; (f_2 \wedge f_3) ; f_4$ | premise |
| 2 | $\quad (f_2 \wedge f_3) ; f_4$ | CP assumption |
| 3 | $\quad f_2 \wedge f_3 ; f_4$ | 2, AndChopDrop |
| 4 | $(f_2 \wedge f_3) ; f_4 \supset f_2 \wedge f_3 ; f_4$ | 2-3, $\supset$ introduction |
| 5 | $(f_0 \wedge f_1) ; (f_2 \wedge f_3) ; f_4 \supset (f_0 \wedge f_1) ; (f_2 \wedge f_3 ; f_4)$ | 4, ITL (RightChopImpChop) |
| 6 | $(f_0 \wedge f_1) ; (f_2 \wedge f_3 ; f_4)$ | 1, 5, MP |
| 7 | $(f_0 ; f_2) \wedge (f_1 ; f_3 ; f_4)$ | 6, TwoChopRulesImp |

LEMMA: RuleChopTwoRuleImp

$\vdash (f_0 \wedge f_1) ; ((f_2 \wedge f_3) \vee (f_4 \wedge f_5))$ implies $\vdash ((f_0 ; f_2) \wedge (f_1 ; f_3)) \vee ((f_0 ; f_4) \wedge (f_1 ; f_5))$

Proof:

| | | |
|---|---|---|
| 1 | $(f_0 \wedge f_1) ; ((f_2 \wedge f_3) \vee (f_4 \wedge f_5))$ | premise |
| 2 | $((f_0 \wedge f_1) ; (f_2 \wedge f_3)) \vee ((f_0 \wedge f_1) ; (f_4 \wedge f_5))$ | 1, ITL (ChopOrEqv) |
| 3 | $\quad (f_0 \wedge f_1) ; (f_2 \wedge f_3)$ | CP assumption |
| 4 | $\quad (f_0 ; f_2) \wedge (f_1 ; f_3)$ | 3, TwoChopRulesImp |
| 5 | $\quad ((f_0 ; f_2) \wedge (f_1 ; f_3)) \vee ((f_0 ; f_4) \wedge (f_1 ; f_5))$ | 4, $\vee$ introduction |
| 6 | $\quad (f_0 \wedge f_1) ; (f_4 \wedge f_5)$ | CP assumption |
| 7 | $\quad (f_0 ; f_4) \wedge (f_1 ; f_5)$ | 6, TwoChopRulesImp |
| 8 | $\quad ((f_0 ; f_4) \wedge (f_1 ; f_5)) \vee ((f_0 ; f_2) \wedge (f_1 ; f_3))$ | 7, $\vee$ introduction |
| 9 | $\quad ((f_0 ; f_2) \wedge (f_1 ; f_3)) \vee ((f_0 ; f_4) \wedge (f_1 ; f_5))$ | 8, commutativity of $\vee$ |
| 10 | $((f_0 ; f_2) \wedge (f_1 ; f_3)) \vee ((f_0 ; f_4) \wedge (f_1 ; f_5))$ | 2, 3-5, 6-9, $\vee$ elimination |

The general transformation strategy for this complete analysis of all state sequences or behaviors associated with $rule_{7.2\text{-}b}$ (and $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$ by inclusion) is to cleave each contributory rule into the component rule condition and rule state, and then add those components, in order, into the aggregate descriptions of rule conditions and corresponding system behaviors. This disassembly and subsequent reassembly is performed using ITL and the rule algebra presented in this research. Because this is an assessment of all possible behaviors associated with an entire set of rules, these alternative behaviors are expressed disjunctively. The target rules are processed in reverse order, that is, from the deepest rule upwards. In this way, behaviors are transformed systematically, and each subsequent behavior associated with a specific rule rests on the behavior defined by that rule's component rules.

This transformation of the five rules $rule_{7.2\text{-}b}$, $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$ is presented in Appendix B. As the deepest rule, $rule_{7.2\text{-}f}$ includes no other rules and therefore, by definition, totally describes all behaviors associated with $rule_{7.2\text{-}f}$. Therefore, $rule_{7.2\text{-}f}$ needs no transformation. Therefore, $rule_{7.2\text{-}e}$ is transformed first and incorporates the behaviors associated with $rule_{7.2\text{-}f}$. Then, $rule_{7.2\text{-}d}$ is transformed and incorporates the behaviors derived from $rule_{7.2\text{-}e}$ and $rule_{7.2\text{-}f}$. Then, $rule_{7.2\text{-}c}$ is transformed and incorporates the behaviors derived from $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$. Finally, $rule_{7.2\text{-}b}$ is transformed and incorporates the behaviors derived from $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$.

The transformations for $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$ are presented in Appendix B. The transformation for $rule_{7.2\text{-}b}$ is presented below:

$$\begin{aligned}
&(\mathit{atm\_non\_empty} \,;\, \neg \mathit{card\_disabled} \,;\\
&\ \neg \mathit{max\_pin} \,;\, \neg \mathit{pin\_exit} \,;\, \neg \mathit{money\_exit}\\
&\ \wedge\ \circ\text{wait\_customer} \,;\, \text{read\_card} \,;\, \circ\text{get\_pin} \,;\, \circ\circ\text{request\_money} \,;\\
&\qquad \circ\text{debit\_account} \,;\, \text{take\_card\_money} \,;\, rule_{7.2\text{-}b}) && (7.2\text{-}27a)\\
\\
&\vee\ (\mathit{atm\_non\_empty} \,;\, \neg \mathit{card\_disabled} \,;\\
&\ \neg \mathit{max\_pin} \,;\, \neg \mathit{pin\_exit} \,;\, \mathit{money\_exit}\\
&\ \wedge\ \circ\text{wait\_customer} \,;\, \text{read\_card} \,;\, \circ\text{get\_pin} \,;\\
&\qquad \circ\circ\text{request\_money} \,;\, \circ\text{take\_card\_money\_exit} \,;\, rule_{7.2\text{-}b}) && (7.2\text{-}27b)\\
\\
&\vee\ (\mathit{atm\_non\_empty} \,;\, \neg \mathit{card\_disabled} \,;\, \neg \mathit{max\_pin} \,;\, \mathit{pin\_exit}\\
&\ \wedge\ \circ\text{wait\_customer} \,;\, \text{read\_card} \,;\, \circ\text{get\_pin} \,;\\
&\qquad \circ\circ\text{take\_card\_pin\_exit} \,;\, rule_{7.2\text{-}b}) && (7.2\text{-}27c)
\end{aligned}$$

$\vee$ (*atm_non_empty* ; $\neg$*card_disabled* ; *max_pin*
$\wedge$ $\circ$wait_customer ; read_card ; $\circ$get_pin ;
$\circ$disable_card ; take_disabled_card ; $rule_{7.2\text{-}b}$) (7.2-27d)

$\vee$ (*atm_non_empty* ; *card_disabled*
$\wedge$ $\circ$wait_customer ; read_card ;
take_disabled_card ; $rule_{7.2\text{-}b}$) (7.2-27e)

$\vee$ ($\neg$*atm_non_empty* $\wedge$ empty) (7.2-27f)

Although (7.2-27) is a single disjunctive statement, each component disjunct is numbered individually to facilitate discussion.

With the above transformation and given the premises $rule_{7.2\text{-}b}$, $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$, (7.2-27) is proven to hold. Stated another way, for a system where $rule_{7.2\text{-}b}$, $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$ are known to hold, (7.2-27) describes the behaviors that are associated with that system. Using (7.2-27) and knowing the verity of the five rule conditions *atm_non_empty*, *card_disabled*, *max_pin*, *pin_exit* and *money_exit* for a specific instance, the system behavior for that instance can be determined. For example and as depicted in (7.2-27f), if $\neg$*atm_non_empty* is satisfied, then empty holds and the system behavior described by $rule_{7.2\text{-}b}$ ends. Similarly and as depicted in (7.2-27e), if *atm_non_empty* and then *card_disabled* holds, then take_disabled_card holds (after both wait_customer and read_card hold). Alternatively, using the transformation presented in (7.2-27), the rule conditions necessary for a desired rule state can be identified. For example and as depicted in (7.2-27a), to achieve the rule state take_card_money, the rule conditions *atm_non_empty*, $\neg$*card_disabled*, $\neg$*max_pin*, $\neg$*pin_exit*, and $\neg$*money_exit* must hold and must hold in that order.

Another important issue with these transformations is that the recursive nature of $rule_{7.2\text{-}b}$ is preserved. Specifically, each alternative behavior, except that behavior associated with $\neg$*atm_non_empty*, ends with an instance of $rule_{7.2\text{-}b}$. For example and as depicted in (7.2-27d), if the rule condition *atm_non_empty* ; $\neg$*card_disabled* ; *max_pin* is satisfied, then the sequence $\circ$wait_customer ; read_card ; $\circ$get_pin ; $\circ$disable_card ; take_disabled_card must hold and then that sequence is followed by $rule_{7.2\text{-}b}$.

A critical issue in this overall approach is that the transformation of $rule_{7.2\text{-}b}$ as presented in (7.2-27) and the corresponding transformations of $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, and $rule_{7.2\text{-}e}$ as presented in Appendix B are disjunctively connected sets of general form rules. Therefore, as a rule system of general form rules, these transformations can be used for additional reasoning about the overall system. Just as the transformation of $rule_{7.2\text{-}c}$ is used to reason about $rule_{7.2\text{-}b}$ and so on, this transformation of $rule_{7.2\text{-}b}$ presented in (7.2-27) can be used to reason about other systems that include $rule_{7.2\text{-}b}$, this including this transformation itself.

The results produced with this transformation are consistent with the results from previous analyses. For example, consider the results of the previous transformation on the reduced rule $rule_{7.2\text{-}b\ \text{take\_card\_money}}$ presented at (7.2-26) and reiterated below for convenience:

$$\begin{aligned}&(\mathit{atm\_non\_empty} \wedge \Diamond\neg \mathit{card\_disabled} \wedge \\ &\quad \Diamond\neg \mathit{max\_pin} \wedge \Diamond\neg \mathit{pin\_exit} \wedge \Diamond\neg \mathit{money\_exit}) \\ &\wedge \circ\text{wait\_customer} ; \text{read\_card} ; \circ\text{get\_pin} ; \\ &\quad \circ\circ\text{request\_money} ; \circ\text{debit\_account} ; \text{take\_card\_money} \qquad (7.2\text{-}26)\end{aligned}$$

Recall that (7.2-26) is a transformation of $rule_{7.2\text{-}b\ \text{take\_card\_money}}$ and that $rule_{7.2\text{-}b\ \text{take\_card\_money}}$ is the result of a state slicing analysis of $rule_{7.2\text{-}b}$, $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$ to identify only the portions of those rules that describe the sequence ending with take_card_money. Therefore, (7.2-26) is compared against (7.2-27a) which, with the exception of the recursive inclusion of $rule_{7.2\text{-}b}$, describes the state sequence that ends with take_card_money. Both (7.2-26) and (7.2-27a) describe the same state sequence starting with wait_customer and ending with take_card_money. Both (7.2-26) and (7.2-27a) describe the same rule conditions that must be met: *atm_non_empty*, ¬*card_disabled*, ¬*max_pin*, ¬*pin_exit*, and ¬*money_exit*. However, (7.2-27a) specifies the ordering with which these conditions must be met relative to each other. Therefore, the transformation concluding with (7.2-27a) retains more information relative to the original rules than the transformation concluding with (7.2-26). However, both are consistent with each other.

In this section, a concrete specification is analyzed and the rules are extracted using the rule algebra developed in this research. These extracted rules offer an equivalent, more manipulatable, and more understandable depiction of the logic, conditions, and nesting associated with each element of the original specification. These six extracted rules, presented at (7.2-1) through (7.2-6), can be manipulated and analyzed in numerous ways as demonstrated herein. However, the various rule analysis examples are offered without prejudice. Which of these techniques are more, or less, useful with respect to analyzing a given system depends on the overall expectations and objectives of a specific rule extraction process. With the state slice of (7.2-7), only those rule components leading to a specific outcome are identified. The transformations (7.2-15a) through (7.2-15f) are derived from these state-slice rule components and are examples of ordered individual sequences leading to the specific conditions that must be met to achieve a specific outcome. These ordered individual sequences are conjunctively connected into the single structure presented at (7.2-16). Using the transformations (7.2-15a) through (7.2-15f), all rule conditions that must be met to achieve a specific outcome and the associated state sequence that supports that outcome are succinctly represented in a single structure in (7.2-26). All rule conditions and the order in which they must be met to achieve all possible behaviors are succinctly represented in a single structure in (7.2-27). Regardless of the specific scrutiny that is subsequently applied, as demonstrated here, once a concrete specification is represented as a set of equivalent general-form rules, a wide range of logical analyses are possible.

## 7.3 Rule Analysis and the Statechart Approach

Although conceptually sound, the general formal framework for rule extraction, as previously presented in Chapter 3, may be compromised by the 'state explosion' problem – the exponential growth of the number of states under analysis – if applied directly to larger programs. Such a problem represents a potentially significant scalability issue regarding the application of this formal framework to real-world rule extraction problems. In this section, statecharts are used to address this problem. Statecharts are an extension of the finite state machines used to represent rule systems in Section 7.1. Coupled with the rule model introduced in Chapter 4 and the rule algebra developed in Chapters 5 and 6, statecharts represent a robust approach to managing the 'state explosion' problem that may result in the extraction and analysis of rules in real-world legacy systems.

### 7.3.1 Overview of Statecharts

Statecharts are a visual formalism for representing the behavior of state systems, especially event-driven, reactive systems (Harel, 1987). In an attempt to counter objections associated with conventional state transition diagrams, statecharts extend state transition diagrams through the inclusion of hierarchy, concurrency, and broadcast communication. To deal with the state explosion problem traditionally associated with a finite state machine representation of larger systems, statecharts include depth so that states and events can be well structured and hierarchical. Statecharts provide for the clustering or abstraction of substates into superstates, and the refinement of superstates into supporting substates. Orthogonality between states, achieved by allowing combinations of synchronization and independence, allows system concurrency. Graphically, statecharts are an ideal aid to system understanding as they provide the ability to move up or down, or zoom, between various levels of the user-defined system abstraction. The semantics of statecharts as implemented in STATEMATE are described by Harel and Naamad (1996). The semantics of UML-statecharts are described by von der Beeck (2001).

State transitions, changes from one system state to another system state, are the core element of the event-driven, reactive system described by statecharts. Harel (1987) describes a state transition as "when event α occurs in state A, if condition C is true at that time, the system transfers to state B." These state transitions are depicted graphically on statecharts as labeled arrows between two states. The general syntax of these state transitions is $\alpha\,[C]\ /\ \beta$ where $\alpha$ is the event that triggers the state transition, $C$ is the guarding condition that must be true for the state transition to occur, and $\beta$ is the action that is executed when the transition occurs (Harel et al., 1990). All of the elements are optional. In general, $\alpha$ and $C$ are inputs and $\beta$ is an output; however, $\beta$ may also serve as an input, i.e., a triggering event, to a state transition in an orthogonal system component. Although actions are represented as part of the transitions between states, as in a Mealy automaton, actions may also be associated with the entrance to or the exit from a specific state, thereby conceptually representing the system as a Moore automaton (Harel, 1987; Harel et al., 1990). Multiple events, conditions, and actions are allowed using Boolean combinations. Wide latitude is afforded regarding what can be defined as an event, condition, or action.

The original motivation for statecharts was reactive systems. Such systems must respond to multiple internal and external inputs, each occurring under different temporal constraints. System changes must occur only when specific triggers occur and only when the corresponding conditions are satisfied. State changes may occur independently of or be synchronized with other subsystems, but will typically occur subject to strict temporal requirements. An example of a complex reactive system is the flight control system in a modern military jet.

Although not typically thought of as a temporally based reactive system, legacy procedural code can be viewed under the same general model. Legacy system code is a defined sequence of code that effectively creates an internal, but enforced, linear temporal system logic. Procedures and functions are called in a specified order, executed, and the mandated state changes made. Control is then returned to the calling object. When executed or 'triggered,' test conditionals are evaluated, and state changes are made subject to explicit instructions specified by the code bound to that conditional. Although such legacy code typically does not involve concurrency and the external inputs may be relatively limited in number and/or monotonic, legacy procedural code can be described as a simple reactive system – simple in that the system logic is explicitly linear with no concurrency requirement and reactive in that the system behavior is determined by internal and possibly external events.

### 7.3.2 Previous Application of Statecharts to Legacy Code Analysis

The use of statecharts and finite state machines (FSMs) for legacy code and reverse engineering analysis has been very limited. Although frequently used for new model verification, there are very few reported cases of the use of statecharts or FSMs for rule extraction or specification recovery from legacy systems. A review of these applications of statecharts or FSMs to legacy code is presented in this section. Given the limited experience in this area, some new system verification work involving statecharts or FSMs that is potentially applicable to legacy system analysis is also reviewed.

Britt (1994) describes the process of comparing a legacy pseudo-code specification for a critical aircraft collision avoidance system with a replacement system requirements specification developed using statecharts. This was largely a cross-mapping exercise by two separate teams, with one team mapping the pseudo-code to the statechart system, and a second team mapping the statecharts to the pseudo-code. Although not explicitly stated, it appears that this mapping was primarily a manual, human-driven process, as opposed to an automated comparison. The correlation between the two systems was not straightforward, as one pseudo-code process might map to several statechart transitions, or one statechart transition might map to several pseudo-code processes.

Corbert et al. (2000) reported on the development and use of an integrated collection of program analysis tools, called Bandera, that can be used to extract finite state models from Java source code. Whereas finite-state verification techniques offer potential with regard to checking hardware design, the authors opine that a major impediment to practical application of finite-state verification techniques is the "model construction problem." Currently, most FSM model construction is manual, which is expensive, prone to error, and difficult to optimize. Further, unlike most system development, which is performed in common general-purpose languages, most model checking programs accept specifications only in a highly specialized, tool-specific input language. To address this semantic and syntactic gap, Bandera takes Java source code as input and generates FSM model code for use in one of several existing verification tools. Bandera was designed to achieve multiple functional criteria: use of existing model checking technologies; automated support for abstractions; model customization; extensibility; and integration of testing and debugging techniques. Bandera consists of a slicer, an abstraction engine, a model generator (including model checker language generation), and a graphical user interface to facilitate component analysis. In model building, three major techniques are applied in the construction of tractable models: irrelevant component elimination, data abstraction, and restriction of the components that are included in the final model. The completed model can then be translated into language for the model checkers Spin, SMV, or SAL.

Popovic et al. (2002) describe the extraction of FSMs from communication software and their use in formal software verification and automated theorem proving.

FSMs are extracted as well-formulated formulas. As the original software was written in C++ and all target FSMs were written as instances of the same class, automated extraction was possible. The left-hand sides of the well-formulated formulas were constructed by looking for two specific functions and extracting the associated state and event names. The right-hand sides of the well-formulated formulas were constructed by analyzing transition functions. These extracted, well-formulated formulas, representing the FSMs in the original code, were then analyzed using the automatic theorem prover THEO to compare the extracted FSMs against the original system specifications.

Giomi (1995) presents a series of techniques for extracting FSMs from hardware description languages (HDLs) such as VHDL and Verilog. In these HDLs, system behaviors can be described, after parsing, by a control flow graph and data flow information. However, FSM description of the system requires the set of inputs, the set of outputs, and a state transition graph consisting of states, state transitions, and transition labels. Techniques are presented for implicitly and explicitly extracting FSMs from HDL sequential behaviors. The implicit technique requires evaluating all executable paths between wait states. The explicit technique requires the construction and evaluation of an explicit state register defining the state machine at the clock edge.

Wang and Edsall (1998) investigated the extraction of FSMs from Verilog code from an industrial/commercial operation. Faced with the substantial challenge of extracting FSMs from different Verilog coding styles, a standardized FSM coding style was implemented. By standardizing the coding style, a custom parser was created to extract the FSMs directly from the Verilog code. These extracted FSMs were then analyzed using various proprietary and commercially available analysis tools. Verification activities included reachability and terminal state analysis, dynamic verification of function coverage, and visual verification of the FSM bubble diagram.

### 7.3.3 Visual Formalisms of Rule-Based Legacy Code Structures

Recalling the underlying basis of the general rule model presented in Section 4.1, a rule is a formal description of a relationship between two states. As refined in Section 4.2, a rule describes a temporal relationship between two states – a state and a future state. As presented in Section 4.5, the general rule form $f_i \wedge \circ f_j$ describes a rule as a relationship between two state sequences, where $f_i$ describes the rule condition in terms of the state sequence properties that must be met for the relationship to hold and where $f_j$ describes the next state sequence that must occur for the relationship to hold. Both in its basis and as formally implemented, a rule is a conditioned relationship between two state sequences. And as demonstrated in this research, rules can be refined so that rules – relationships between states – can be incorporated within other rules. Therefore, because statecharts describe relationships between state sequences, because statecharts allow for the explicit association of conditions with the transitions describing these relationships, and because statecharts allow for the hierarchical representation of state sequences within state sequences, there exists a strong correspondence between the critical elements of rules as defined in this research and statecharts. In this section, statecharts are used to represent rules.

In general, these statecharts will be represented using the STATEMATE syntax, with any exceptions or assumptions noted. In the STATEMATE syntax, transitions are labeled as $\alpha$ $[C]$ / $\beta$, where $\alpha$ is the event that triggers the state transition, $C$ is the guarding condition that must be true for the state transition to occur, and $\beta$ is the action that is executed when the transition occurs (Harel et al., 1990). All transition elements are optional.

As an introductory exercise to using statecharts to describe rules, consider the simple two-state system presented in Figure 7.3.3-1.

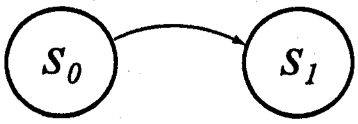


Figure 7.3.3-1: A Simple Two-State System

As noted in Chapter 5, this simple two-state system is irreflexive, asymmetric, and antisymmetric. This system is the simplest possible two-state system, because it contains only one state transition and therefore is described, without manipulation, by a single general-form rule.

In this system, $s_0 \models w_0$, and $s_1 \models w_1$. The one transition included in this system can be described in rule form as $w_0 \wedge \circ w_1$. In this rule, the rule condition is described by $w_0$ and the rule state is described by $w_1$.

This simple two-state system is represented as a statechart in Figure 7.3.3-2.

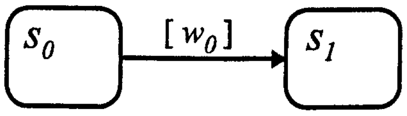


Figure 7.3.3-2: A Simple Two-State Statechart

States (or state sequences) are represented using rounded rectangles and the state transition between the two states $s_0$ and $s_1$ is represented using the labeled arrow. The rule condition $w_0$ is associated with this transition using the STATEMATE syntax described above. For this transition, there is no event $\alpha$ or action $\beta$ associated with the transition. Consistent with the rule $w_0 \wedge \circ w_1$ describing the simple two-state system presented in Figure 7.3.3-1, an interpretation of the simple statechart presented in Figure 7.3.3-1 is that the transition between state $s_0$ and state $s_1$ occurs only if the condition $w_0$ is met. With the previous specification that $s_1 \models w_1$, the rule $w_0 \wedge \circ w_1$ holds under this statechart.

This example is purposefully simple to facilitate demonstration. To facilitate the analysis and extraction of rules in legacy code, several generic visual formalisms of rule-based legacy code structures have been developed using statecharts. Four common rule-based legacy code structures are analyzed: the 'if-then-else' structure, the 'while' structure, the 'indexed for loop' structure, and the 'switch' structure. Using statechart concepts, generic visual formalisms are developed for each of these legacy structures.

### 7.3.3.1 Statechart of the 'if-then-else' Structure

In Section 6.6.1, a rule-based 'if-then-else' structure is defined as:

$$(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \circ f_2) \qquad (7.3.3.1\text{-}1)$$

With this rule pair, two state sequence relationships are described. If $f_0$ is satisfied, the next state sequence must satisfy $f_1$. Conversely, if $\neg f_0$ is satisfied, the next state sequence must satisfy $f_2$. Letting $\sigma_1$ represent the state sequence that satisfies $f_1$ (i.e., $\sigma_1 \models f_1$) and letting $\sigma_2$ represent the state sequence that satisfies $f_2$ (i.e., $\sigma_2 \models f_2$), the generic visual formalism for the 'if-then-else' structure of (7.3.3.1-1) is presented in Figure 7.3.3.1-1.

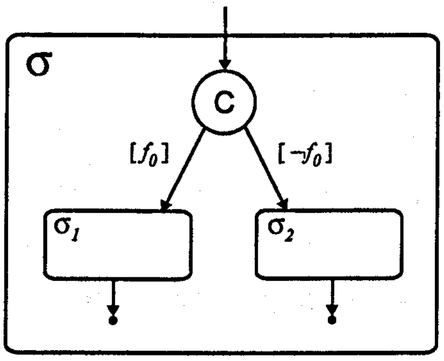


Figure 7.3.3.1-1: Generic Visual Formalism of the 'if-then-else' Structure

Within the super-state $\sigma$, the branching between the state sequences $\sigma_1$ and $\sigma_2$ is depicted with the C-connector. If the condition $f_0$ is satisfied, then the next state sequence is $\sigma_1$. That this condition is met is denoted by the labeling of the transition as $[f_0]$, consistent with STATEMATE labeling conventions. Alternatively, if the condition $\neg f_0$ is satisfied, then the next state sequence is $\sigma_2$.

A variation of the rule-based 'if-then-else' structure is the 'if-then' structure. This structure is defined in Section 6.6.1 as:

$$(f_0 \wedge \circ f_1) \vee (\neg f_0 \wedge \text{empty}) \qquad (7.3.3.1\text{-}2)$$

Letting $\sigma_1$ represent the state sequence that satisfies $f_1$ (i.e., $\sigma_1 \models f_1$), the generic visual formalism for the 'if-then' structure of (7.3.3.1-2) is presented in Figure 7.3.3.1-2.

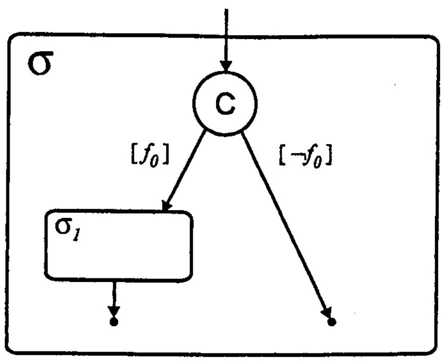


Figure 7.3.3.1-2: Generic Visual Formalism of the 'if-then' Structure

### 7.3.3.2 Statechart of the 'while' Structure

In Section 6.6.2, a rule-based 'while' structure has been defined as a recursive loop as:

$$\text{while} f_0 \text{ do } \circ f_1 \triangleq ((f_0 \wedge \circ f_1) \,;\, \text{while} f_0 \text{ do} f_1) \vee (\neg f_0 \wedge \text{empty}) \qquad (7.3.3.2\text{-}1)$$

In this structure, if $f_0$ is satisfied, the next state sequence must satisfy $f_1$, and this relationship between $f_0$ and $\circ f_1$ exists until $f_0$ is no longer satisfied. Letting $\sigma_1$ represent the state sequence that satisfies $f_1$ (i.e., $\sigma_1 \models f_1$), the generic visual formalism for the 'while' structure of (7.3.3.2-1) is presented in Figure 7.3.3.2-1.

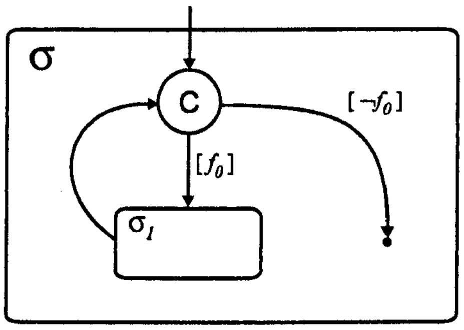


Figure 7.3.3.2-1: Generic Visual Formalism of the 'while' Structure

### 7.3.3.3 Statechart of the 'indexed for-loop' Structure

In Section 6.6.3, a rule-based indexed for-loop structure is defined in terms of a 'while' structure as :

$$\text{for } A = b \text{ to } c \text{ do } f_1 \triangleq (\circ A = b) \,;\, rule' \qquad (7.3.3.3\text{-}1)$$

where:

$$rule' \triangleq (((A \leq c) \land \circ f_1 \,;\, \circ A = A + 1) \,;\, rule') \lor (\neg(A \leq c) \land \text{empty})$$

Letting $\sigma_1$ represent the state sequence that satisfies $f_1$ (i.e., $\sigma_1 \models f_1$), the generic visual formalism for the indexed for-loop structure of (7.3.3.3-1) is presented in Figure 7.3.3.3-1.

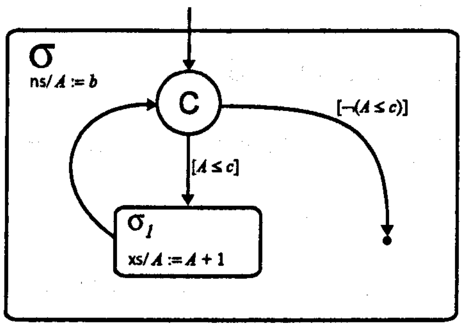


Figure 7.3.3.3-1: Generic Visual Formalism of the Indexed 'for-loop' Structure

This visual formalism incorporates two additional elements associated with the STATEMATE statecharts. On entry to the super-state $\sigma$, the index counter $A$ is initialized and set to $b$. This is denoted by the ns/ $A := b$ statement. With each exit from the state sequence $\sigma_1$, the index counter $A$ is incremented by 1. This is denoted by the xs/ $A := A + 1$ statement. With these two additions, the visual similarities between the indexed for-loop and the 'while' statement are evident, reflecting the underlying logical similarities.

### 7.3.3.4 Statechart of the 'switch' Structure

Consider the following guarded command statement:

$$(f_0 \land \circ f_2) \,\square\, (f_1 \land \circ f_3) \,\square\, ((\neg f_0 \land \neg f_1) \land \circ f_4) \qquad (7.3.3.4\text{-}1)$$

This guarded command concept has various implementations in different languages, including the *switch* statement in C and Java, the *evaluate* statement in COBOL, and the *case* statement in Pascal and Ada. Although details vary with language, all implementations of the switch-type construct follow the same general concept. As discussed in Section 6.3, guarded command statements can be logically represented with disjunction. Therefore, (7.3.3.4-1) can be represented as:

$$(f_0 \wedge \circ f_2) \vee (f_1 \wedge \circ f_3) \vee ((\neg f_0 \wedge \neg f_1) \wedge \circ f_4) \qquad (7.3.3.4\text{-}2)$$

Given three state sequences $\sigma_2$, $\sigma_3$, and $\sigma_4$, such that $\sigma_2 \models f_2$, $\sigma_3 \models f_3$, and $\sigma_4 \models f_4$, the generic visual formalism for the 'switch' structure of (7.3.3.4-1) is presented in Figure 7.3.3.4-1.

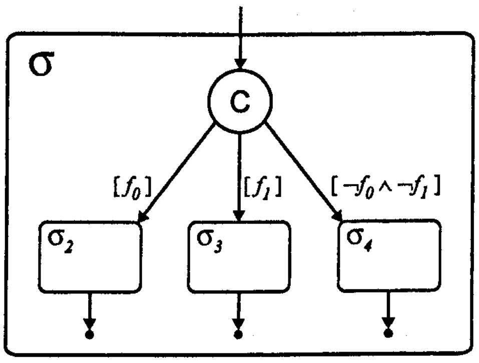


Figure 7.3.3.4-1: Generic Visual Formalism of the 'switch' Structure

Similarities in both structure and function between 'switch' structures and 'if-then-else' structures are noted frequently in comparisons of languages and language structures (e.g., Sebesta, 2002; Scott, 2000). The similarities between the 'switch' structure in Figure 7.3.3.4-1 and the 'if-then-else' structure in Figure 7.3.3.1-1 are evident.

### 7.3.4 Representing Extracted Rules with Statecharts

In this section, statecharts are used to depict the rules that were extracted from the automatic teller machine specification in Section 7.2. In that section, the following six rules were extracted from the original specification:

$$rule_{7.2\text{-}a} \triangleq ((true \wedge \circ rule_{7.2\text{-}b} \,;\, \text{refill\_atm}) \,;\, rule_{7.2\text{-}a}) \vee (\neg true \wedge \text{empty})$$

$$rule_{7.2\text{-}b} \triangleq ((atm\_non\_empty \wedge \circ\text{wait\_customer} \,;\, \text{read\_card};\, rule_{7.2\text{-}c}) \,;\, rule_{7.2\text{-}b}) \vee (\neg atm\_non\_empty \wedge \text{empty})$$

$$rule_{7.2\text{-}c} \triangleq (card\_disabled \wedge \circ\text{take\_disabled\_card}) \vee (\neg card\_disabled \wedge \circ\text{get\_pin} \,;\, rule_{7.2\text{-}d})$$

$$\begin{aligned} rule_{7.2\text{-}d} &\triangleq (max\_pin \wedge \circ\text{disable\_card} ; \text{take\_disabled\_card}) \\ &\quad \vee (\neg max\_pin \wedge \circ rule_{7.2\text{-}e}) \end{aligned}$$

$$\begin{aligned} rule_{7.2\text{-}e} &\triangleq (pin\_exit \wedge \circ\text{take\_card\_pin\_exit}) \\ &\quad \vee (\neg pin\_exit \wedge \circ\text{request\_money} ; rule_{7.2\text{-}f}) \end{aligned}$$

$$\begin{aligned} rule_{7.2\text{-}f} &\triangleq (money\_exit \wedge \circ\text{take\_card\_money\_exit}) \\ &\quad \vee (\neg money\_exit \wedge \circ\text{debit\_account} ; \text{take\_card\_money}) \end{aligned}$$

To highlight how these hierarchical properties of statecharts allow for rules and the corresponding state sequences to be embedded in each subsequent statechart, these six rules will be processed from the top down. Statecharts are developed for each rule, and each statechart depicts the state sequence that satisfies the corresponding temporal formula in each rule. For example, $\sigma_{rule_{7.2\text{-}a}}$ satisfies $rule_{7.2\text{-}a}$, $\sigma_{\text{refill\_atm}}$ satisfies refill_atm, etc. With the exception of the last rule, which contains no explicit rules, each statechart includes a state sequence described by another rule. Just as the above six rules describe a logical connection and hierarchy between one rule and the next, the corresponding statecharts depict those connections and hierarchy graphically.

Starting with the first or top rule, $rule_{7.2\text{-}a}$ is defined as:

$$rule_{7.2\text{-}a} \triangleq ((true \wedge \circ rule_{7.2\text{-}b} ; \text{refill\_atm}) ; rule_{7.2\text{-}a}) \vee (\neg true \wedge \text{empty})$$

The statechart depicting the state sequences satisfying $rule_{7.2\text{-}a}$ is presented in Figure 7.3.4-1. The statechart for $rule_{7.2\text{-}a}$ is based on the generic visual formalism for the 'while' structure as previously presented in Section 7.3.3.2. $rule_{7.2\text{-}a}$ is a system rule, and the corresponding state sequence is described such that it does not terminate. (The termination case of $\neg true$ is shown on this statechart for completeness.) This statechart includes the state sequences described by $rule_{7.2\text{-}b}$ and refill_atm. After $\sigma_{\text{refill\_atm}}$, $\sigma_{rule_{7.2\text{-}a}}$ is repeated. Thus, the formula $(true \wedge \circ rule_{7.2\text{-}b} ; \text{refill\_atm}) ; rule_{7.2\text{-}a}$ is satisfied by $\sigma_{rule_{7.2\text{-}a}}$ as depicted in this statechart.

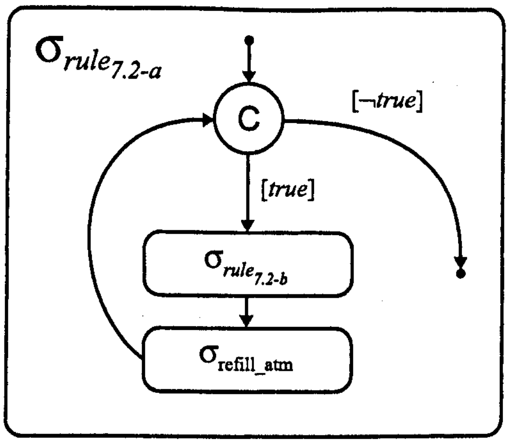

$\sigma_{rule_{7.2\text{-}a}}$
C
[¬*true*]
[*true*]
$\sigma_{rule_{7.2\text{-}b}}$
$\sigma_{\text{refill\_atm}}$

Figure 7.3.4-1: Statechart for $rule_{7.2\text{-}a}$

$rule_{7.2\text{-}a}$ and the corresponding statechart for $rule_{7.2\text{-}a}$ (presented in Figure 7.3.4-1) include $rule_{7.2\text{-}b}$. The definition of $rule_{7.2\text{-}b}$ is:

$$rule_{7.2\text{-}b} \triangleq ((atm\_non\_empty \wedge \circ\text{wait\_customer} ; \text{read\_card}; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b}) \\ \vee (\neg atm\_non\_empty \wedge \text{empty})$$

The statechart depicting the state sequences satisfying $rule_{7.2\text{-}b}$ is presented in Figure 7.3.4-2. The statechart for $rule_{7.2\text{-}b}$ is based on the generic visual formalism for the 'while' structure as previously presented in Section 7.3.3.2. In $rule_{7.2\text{-}b}$, branching between two alternative state sequences is based on the rule condition *atm_non_empty*. If *atm_non_empty* is satisfied, a state sequence described by wait_customer, then read_card, and then $rule_{7.2\text{-}c}$ follows, and then $rule_{7.2\text{-}b}$ is repeated. With this series of state sequences, the formula $(atm\_non\_empty \wedge \circ\text{wait\_customer} ; \text{read\_card} ; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b}$ is satisfied. If *atm_non_empty* is not satisfied, the state sequence satisfying $rule_{7.2\text{-}b}$ ends, and the formula $\neg atm\_non\_empty \wedge \text{empty}$ is satisfied. In this statechart, this termination is depicted with the stubbed arrow. Given that $\sigma_{rule_{7.2\text{-}b}}$ is part of $\sigma_{rule_{7.2\text{-}a}}$, with the termination of $rule_{7.2\text{-}b}$, the state sequence described by $rule_{7.2\text{-}a}$ continues with refill_atm (as previously depicted in Figure 7.3.4-1).

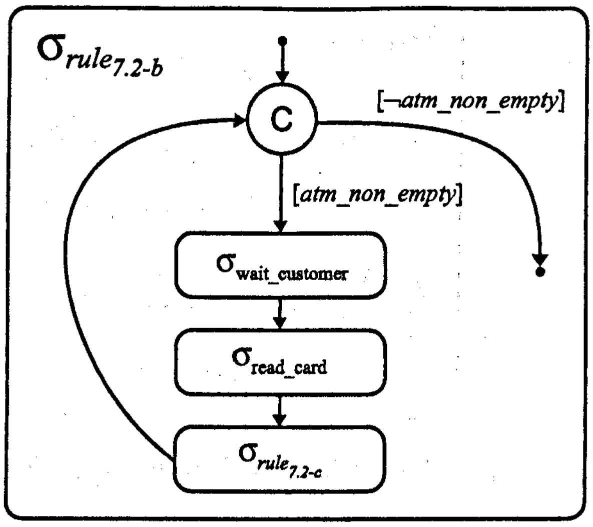

$\sigma_{rule_{7.2\text{-}b}}$
C
[¬atm_non_empty]
[atm_non_empty]
$\sigma_{\text{wait\_customer}}$
$\sigma_{\text{read\_card}}$
$\sigma_{rule_{7.2\text{-}c}}$

Figure 7.3.4-2: Statechart for $rule_{7.2\text{-}b}$

$rule_{7.2\text{-}b}$ and the corresponding statechart for $rule_{7.2\text{-}b}$ (presented in Figure 7.3.4-2) include $rule_{7.2\text{-}c}$. The definition of $rule_{7.2\text{-}c}$ is:

$$rule_{7.2\text{-}c} \triangleq (\mathit{card\_disabled} \wedge \circ\text{take\_disabled\_card}) \\ \vee (\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} \,;\, rule_{7.2\text{-}d})$$

The statechart depicting the state sequences satisfying $rule_{7.2\text{-}c}$ is presented in Figure 7.3.4-3. The statechart for $rule_{7.2\text{-}c}$ is based on the generic visual formalism for the 'if-then-else' structure as previously presented in Section 7.3.3.1. In $rule_{7.2\text{-}c}$, branching between two alternative state sequences is based on the rule condition *card_disabled.* If *card_disabled* is not satisfied (that is, if ¬*card_disabled* is true), a state sequence described by get_pin and then $rule_{7.2\text{-}d}$ follows. Thus, the formula $\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} \,;\, rule_{7.2\text{-}d}$ is satisfied. If *card_disabled* is satisfied, a state sequence described by take_disabled_card follows and the formula $\mathit{card\_disabled} \wedge \circ\text{take\_disabled\_card}$ is satisfied.

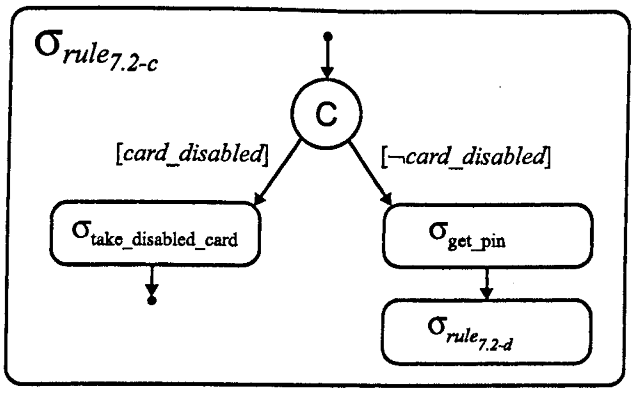

$\sigma_{rule_{7.2\text{-}c}}$
C
[card_disabled]
[¬card_disabled]
$\sigma_{\text{take\_disabled\_card}}$
$\sigma_{\text{get\_pin}}$
$\sigma_{rule_{7.2\text{-}d}}$

Figure 7.3.4-3: Statechart for $rule_{7.2\text{-}c}$

$rule_{7.2\text{-}c}$ and the corresponding statechart for $rule_{7.2\text{-}c}$ (presented in Figure 7.3.4-3) include $rule_{7.2\text{-}d}$. The definition of $rule_{7.2\text{-}d}$ is:

$$rule_{7.2\text{-}d} \triangleq (max\_pin \wedge \circ\text{disable\_card} \,;\, \text{take\_disabled\_card}) \vee (\neg max\_pin \wedge \circ rule_{7.2\text{-}e})$$

The statechart depicting the state sequences satisfying $rule_{7.2\text{-}d}$ is presented in Figure 7.3.4-4. The statechart for $rule_{7.2\text{-}c}$ is based on the generic visual formalism for the 'if-then-else' structure as previously presented in Section 7.3.3.1. In $rule_{7.2\text{-}d}$, branching between two alternative state sequences is based on the rule condition *max_pin*. If *max_pin* is not satisfied, a state sequence described by $rule_{7.2\text{-}e}$ follows. Thus, the formula $\neg max\_pin \wedge \circ rule_{7.2\text{-}e}$ is satisfied. If *max_pin* is satisfied, a state sequence described by disable_card follows and the formula $card\_disabled \wedge \circ\text{disable\_card}$ is satisfied.

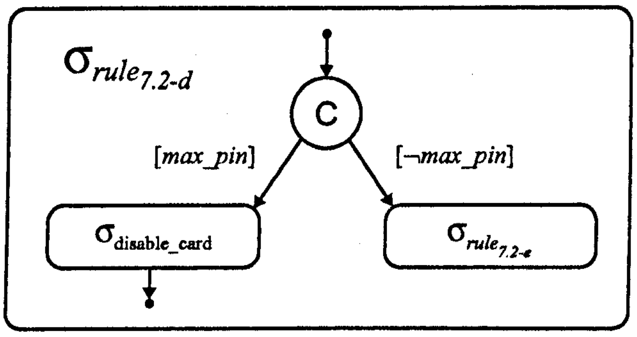
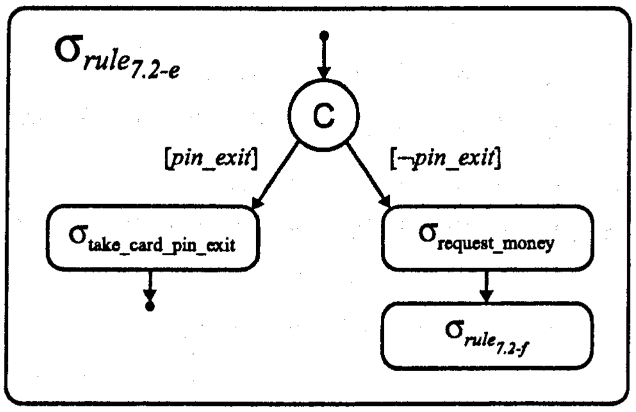

$\sigma_{rule_{7.2\text{-}d}}$
C
[max_pin]
[¬max_pin]
$\sigma_{\text{disable\_card}}$
$\sigma_{rule_{7.2\text{-}e}}$

Figure 7.3.4-4: Statechart for $rule_{7.2\text{-}d}$

$rule_{7.2\text{-}d}$ and the corresponding statechart for $rule_{7.2\text{-}d}$ (presented in Figure 7.3.4-4) include $rule_{7.2\text{-}e}$. The definition of $rule_{7.2\text{-}e}$ is:

$$rule_{7.2\text{-}e} \triangleq (pin\_exit \wedge \circ \mathrm{take\_card\_pin\_exit}) \\ \vee (\neg pin\_exit \wedge \circ \mathrm{request\_money} \,;\, rule_{7.2\text{-}f})$$

The statechart depicting the state sequences satisfying $rule_{7.2\text{-}e}$ is presented in Figure 7.3.4-5. The statechart for $rule_{7.2\text{-}e}$ is based on the generic visual formalism for the 'if-then-else' structure as previously presented in Section 7.3.3.1. In $rule_{7.2\text{-}e}$, branching between two alternative state sequences is based on the rule condition *pin_exit.* If *pin_exit* is not satisfied, a state sequence described by request_money and then $rule_{7.2\text{-}f}$ follows. Thus, the formula $\neg pin_exit \wedge \circ$request_money ; $rule_{7.2\text{-}f}$ is satisfied. If *pin_exit* is satisfied, a state sequence described by take_card_pin_exit follows and the formula $pin_exit \wedge \circ$take_card_pin_exit is satisfied.



$\sigma_{rule_{7.2\text{-}e}}$
C
[pin_exit]
[¬pin_exit]
$\sigma_{\mathrm{take\_card\_pin\_exit}}$
$\sigma_{\mathrm{request\_money}}$
$\sigma_{rule_{7.2\text{-}f}}$

Figure 7.3.4-5: Statechart for $rule_{7.2\text{-}e}$

$rule_{7.2\text{-}e}$ and the corresponding statechart for $rule_{7.2\text{-}e}$ (presented in Figure 7.3.4-5) include $rule_{7.2\text{-}f}$. The definition of $rule_{7.2\text{-}f}$ is:

$$rule_{7.2\text{-}f} \triangleq (money\_exit \wedge \circ \mathrm{take\_card\_money\_exit}) \\ \vee (\neg money\_exit \wedge \circ \mathrm{debit\_account} \,;\, \mathrm{take\_card\_money})$$

The statechart depicting the state sequences satisfying $rule_{7.2\text{-}f}$ is presented in Figure 7.3.4-6. The statechart for $rule_{7.2\text{-}f}$ is based on the generic visual formalism for the 'if-then-else' structure as previously presented in Section 7.3.3.1. In *$rule_{7.2\text{-}f}$*, branching between two alternative state sequences is based on the rule condition *money_exit*. If *money_exit* is not satisfied, a state sequence described by debit_account and then take_card_money follows. Thus, the formula $\neg$*money_exit* $\wedge$ $\circ$debit_account ; take_card_money is satisfied. If *money_exit* is satisfied, a state sequence described by take_card_money_exit follows and the formula *money_exit* $\wedge$ $\circ$take_card_money_exit is satisfied.

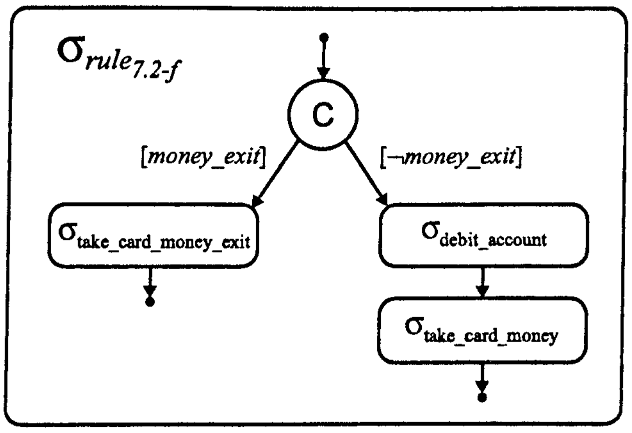
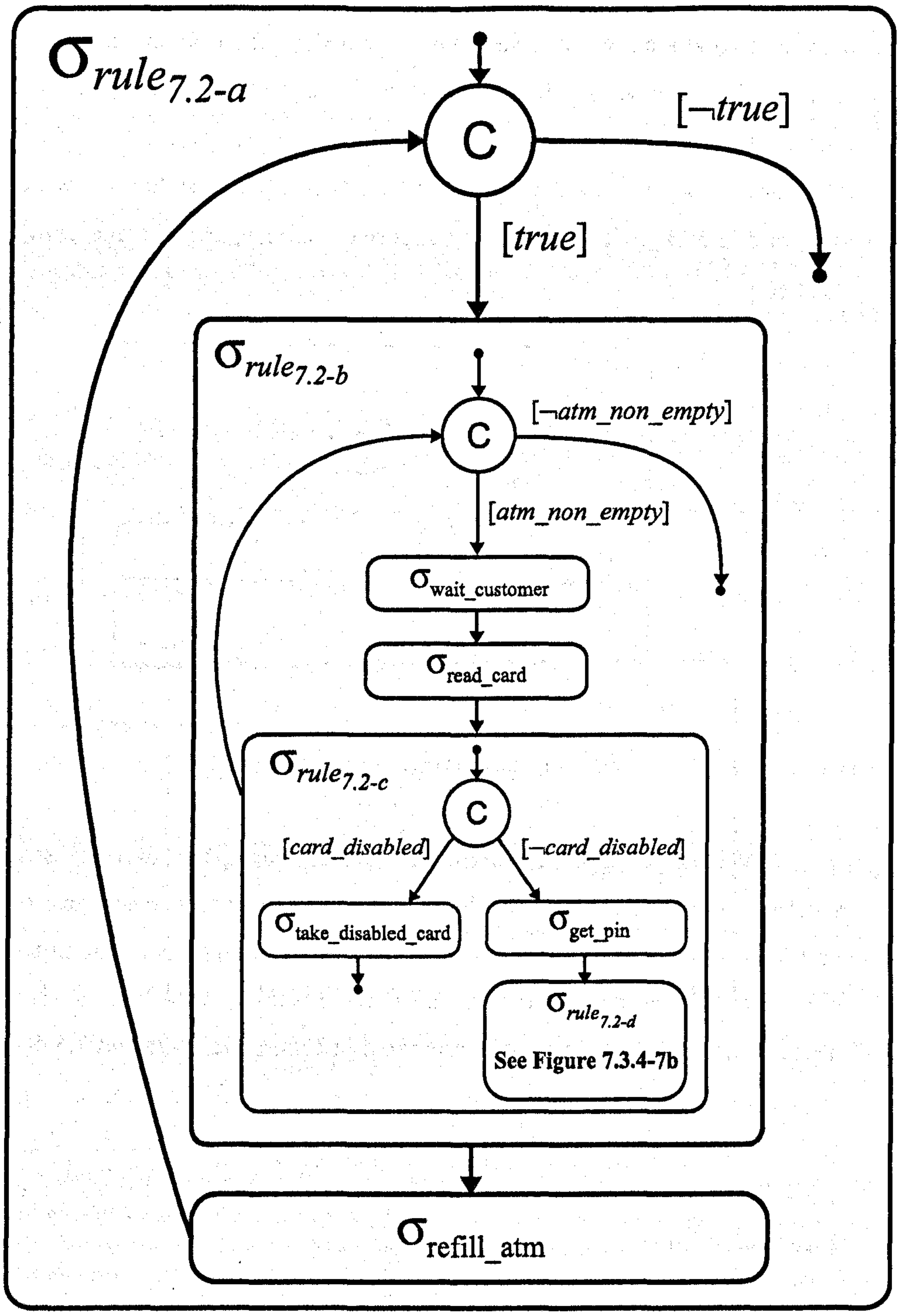

$\sigma_{rule_{7.2\text{-}f}}$
C
[money_exit]
[¬money_exit]
$\sigma_{\text{take\_card\_money\_exit}}$
$\sigma_{\text{debit\_account}}$
$\sigma_{\text{take\_card\_money}}$

Figure 7.3.4-6: Statechart for *$rule_{7.2\text{-}f}$*

Whereas each of these individual statecharts describes the state sequences satisfying each individual rule, the power and value of statecharts can be understood best by looking at these statecharts and the associated rules as a unified whole. As this rule system is composed of six rules, the resulting statechart is six layers deep. Therefore, the composite chart is presented in Figures 7.3.4-7a and 7.3.4-7b. The state sequences satisfying *$rule_{7.2\text{-}a}$*, *$rule_{7.2\text{-}b}$*, and *$rule_{7.2\text{-}c}$* are presented in Figure 7.3.4-7a, which includes a minimal depiction of *$rule_{7.2\text{-}d}$* and a corresponding reference to Figure 7.3.4-7b. The state sequences satisfying *$rule_{7.2\text{-}d}$*, *$rule_{7.2\text{-}e}$*, and *$rule_{7.2\text{-}f}$* are presented in Figure 7.3.4-7b.



$\sigma_{rule_{7.2\text{-}a}}$
C
[¬true]
[true]
$\sigma_{rule_{7.2\text{-}b}}$
C
[¬atm_non_empty]
[atm_non_empty]
$\sigma_{\text{wait\_customer}}$
$\sigma_{\text{read\_card}}$
$\sigma_{rule_{7.2\text{-}c}}$
C
[card_disabled]
[¬card_disabled]
$\sigma_{\text{take\_disabled\_card}}$
$\sigma_{\text{get\_pin}}$
$\sigma_{rule_{7.2\text{-}d}}$
See Figure 7.3.4-7b
$\sigma_{\text{refill\_atm}}$

Figure 7.3.4-7a: Statechart for $rule_{7.2\text{-}a}$, $rule_{7.2\text{-}b}$, and $rule_{7.2\text{-}c}$

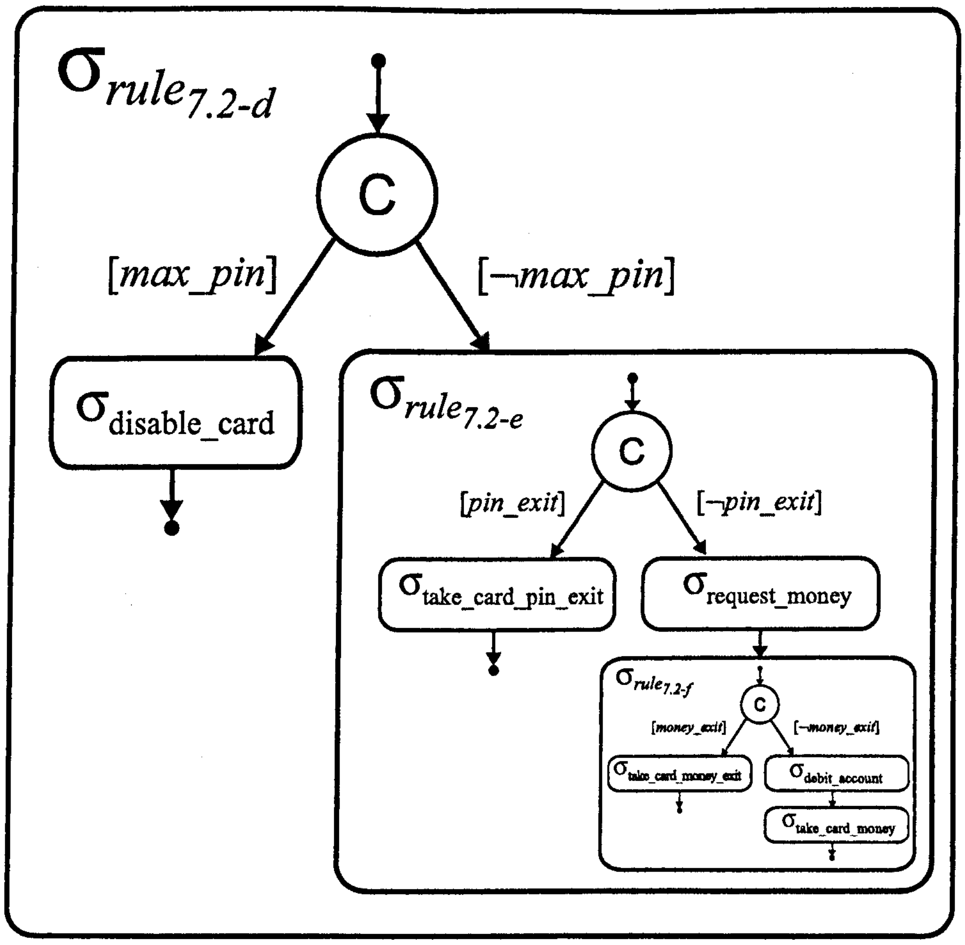


Figure 7.3.4-7b: Statechart for $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$

Quite literally, $rule_{7.2\text{-}a}$ is described in Figures 7.3.4-7a and 7.3.4-7b as a statechart inside of a statechart inside of a statechart, etc., just as the six rules $rule_{7.2\text{-}a}$ through $rule_{7.2\text{-}f}$ are, quite literally, a rule within a rule within a rule, etc. In statechart form, the logical connections, sequencing, and nested relationships of the six rules $rule_{7.2\text{-}a}$ through $rule_{7.2\text{-}f}$ are clearly depicted graphically. Just as the general rule form $f_i \wedge \circ f_j$ allows the encapsulation of the logical relationships between state sequences by using rules within rules, statecharts allow the same encapsulation by imbedding statecharts within statecharts. In both cases, with this encapsulation comes the ability to represent depth, and limit or focus interest to a specific depth as necessary or appropriate.

For example, ignore the reference to Figure 7.3.4-7b and consider Figure 7.3.4-7a as an autonomous statechart. With the suspension of Figure 7.3.4-7b, the three rules depicted in the statechart presented in Figure 7.3.4-7a are:

$$\begin{aligned}
rule_{7.2\text{-}a} &\triangleq ((true \wedge \circ rule_{7.2\text{-}b}\,;\ \text{refill\_atm})\,;\ rule_{7.2\text{-}a}) \vee (\neg true \wedge \text{empty}) \\
rule_{7.2\text{-}b} &\triangleq ((atm\_non\_empty \wedge \circ\text{wait\_customer}\,;\ \text{read\_card};\ rule_{7.2\text{-}c})\,;\ rule_{7.2\text{-}b}) \\
&\qquad \vee (\neg atm\_non\_empty \wedge \text{empty}) \\
rule_{7.2\text{-}c} &\triangleq (card\_disabled \wedge \circ\text{take\_disabled\_card}) \\
&\qquad \vee (\neg card\_disabled \wedge \circ\text{get\_pin}\,;\ rule_{7.2\text{-}d})
\end{aligned}$$

Without additional details regarding $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}d}$ is just another minimally-defined state sequence. Like wait_customer or read_card, no details are available regarding the state sequence that $rule_{7.2\text{-}d}$ represents. If the label $rule_{7.2\text{-}d}$ were replaced with the label rule_to_take_money, the similarities would be even more dramatic. However, with the addition of Figure 7.3.4-7b to Figure 7.3.4-7a, $rule_{7.2\text{-}d}$ is expanded, and additional depth and details are added regarding the state sequence that $rule_{7.2\text{-}d}$ represents. Similarly, with the additional description of $rule_{7.2\text{-}d}$ in terms of the general form rule $f_i \wedge \circ f_j$, including definitions and references to $rule_{7.2\text{-}d}$ and $rule_{7.2\text{-}e}$, additional depth and details are added to the rule-based description of the system.

This comparison is made to highlight a critical issue – that general form rules (i.e., $f_i \wedge \circ f_j$) and statecharts as presented here are different representations of the same information, specifically the conditioned relationships between state sequences. If the visual formalisms of the various legacy code structures presented in Section 7.3.3 are accepted as accurate representations of the underlying logical concepts, and the implied correspondences between the statechart elements and the rule elements are accepted, then the statecharts of Figures 7.3.4-7a and 7.3.4-7b and the extracted six rules are equivalent. And with that, these presentations differ not in content, but only in how they can be used in future analysis and understanding. Whereas the statechart approach allows a visual presentation that is readily understandable by a wider audience, the formulaic approach of representing the extracted rules as ITL formulas is readily adaptable to computer analysis techniques.

### 7.3.5 The Value of Statecharts in Legacy Code Analysis

Several important issues regarding rule extraction, legacy code analysis, and statecharts merit special note. Common legacy code concepts, previously expressed in Chapter 6 in terms of the general form rule $f_i \wedge \circ f_j$, have been expressed in terms of statechart visual formalisms. These visual formalisms provide a graphical representation of the system state changes that occur with each legacy structure. These visual formalisms depict the location of program rules and the resulting state changes. These various visual formalisms demonstrate the similarities and differences between various code structures, again facilitating both understanding and analysis. This statechart approach is consistent with the rule model presented in Chapter 4 and the associated rule algebra presented in Chapters 5 and 6. Using these four legacy code formalisms developed here, more complex logical and programming structures can be built using the rule algebra presented in Chapters 5 and 6, either by linking these concepts together or by nesting structures within structures. With such an expanded approach, sophisticated and complex legacy codes can be graphically represented for both understanding and analysis. Similarly, with the corresponding ability to encapsulate or hide states within states, the 'state explosion' problem associated with the application of the formal framework presented in Chapter 3 can be managed. Considering these factors, statecharts, in concert with the rule model and rule algebra presented in this research, provide a robust tool for legacy code analysis.

# Chapter 8

# Analysis of Rules in Legacy Code

In this chapter, the formal rule extraction framework of Chapter 3, the formal temporal rule model of Chapter 4, and the rule algebra of Chapters 5 and 6 are applied to the extraction and analysis of rules from legacy code. In Section 8.1, the formal rule model and the corresponding rule algebra are applied to the extraction and analysis of the rules contained in a small but relatively complicated block of legacy code; using the rule model and rule algebra, a corresponding database is developed to describe the rule and non-rule elements of this legacy code. A statechart is developed to assist in code analysis and understanding, and the extracted rules are assessed based on specific variables of interest. In Section 8.2, the FermaT tool is used to slice an example WSL program, and rules are extracted from the associated program slice(s).

## 8.1 Using Rules to Build a Database for Legacy Code Analysis

In this section, the concepts developed in this research are applied to a small but relatively complicated block of legacy code. Using the rule extraction framework presented in Chapter 3, the rule model presented in Chapter 4, and the rule algebra presented in Chapters 5 and 6, rules are extracted and a simple rule-analysis database is developed to describe the rules and non-rule elements in the legacy code. To supplement this rule extraction and the associated database, a statechart of the target legacy code is developed using the statechart concepts presented in Section 7.3.

As demonstrated in the section, the rule algebra is applied and the legacy code is transformed into a series of rules and formulas. Then, the properties of these rules and formulas are recorded in the associated database. Within this analysis paradigm, the application of this rule algebra provides a formal context for the identification of a wide range of rule and formula properties that may be of specific interest to the user relative to the user's analysis objectives. Therefore, the design of this database can and will vary substantially depending on how the database will be used, including specific project needs, database analysis techniques, and other anticipated applications of the database information. Also, the design of the database depends on whether the database is an adjunct to the transformed code or a replacement for the original code. Therefore, the database that is developed using this analysis paradigm can be as simple or as complicated as desired. For this demonstration, a relatively small set of properties have been selected. The following fields are included in the rule database:

- Rule or formula label
- ITL formula
- *W* (frame variable set)
- *V* (used variable set)
- Primary membership of the rule or formula

Based on the extraction and analysis presented below, this completed database is presented at the end of this section in Table 8.1-1.

The legacy code used in this example has been the subject of previous formal abstraction analysis (Cau and Zedan, 2006). The legacy code example analyzed here is a procedure from a published lexical scanner package written in Pascal. The total package, the overall package structure, and related procedures are discussed in detail in Cau and Zedan (2006). The target of this rule analysis, the procedure printerrorline, is presented as follows:

```
procedure printerrorline(var Lbuf: linebufrec);
var
   Column,I,J,Num: integer;
begin
   Column:= 0;
   with Lbuf do
   begin
      printline(Lbuf);
      write('*****': 6, ' ');
      for I:=1 to length +1 do
         if eline[I] < > errnone then
         begin
            errorset:= errorset+[eline[I]];
            Num:= ord(eline[I]);
            if I > Column then
            begin
               for J:= Column + 2 to I do write(' ');
               write('↑');Column:= I
            end
            else begin write(',');Column:= Column + 1 end
```

```
          write(Num:1);
          Column:= Column + l;
          if Num > 9 then Column:= Column + l;
          eline[I]:= errnone
        end; {of if and for}
      writeln;
      lineerror:= false;
      fileerror:= true
    end {of with}
  end; {of procedure printerrorline}
```

Within this legacy code, rules are identified based on the if-then-else, while, and indexed for-loop code structures. These rule structures reflect locations in the legacy code where alternative state sequences may be created based on the satisfaction or non-satisfaction of the associated rule conditions. To facilitate the incremental analysis of this legacy code, mixed formulas are allowed. Consistent with the *Spec* representation used in Cau and Zedan (2006), mixed formulas used to represent the associated legacy code may contain concrete code structures, ITL formulas, and other abstract specifications, as needed and as appropriate.

Rules and non-rule formulas are extracted from the target code using the procedure described below:

1. Consistent with the general framework outlined in Chapter 3, the legacy code is analyzed and broken into individual units based on the syntax of the target language, in this case, Pascal.
2. Working from the top down, these individual units are analyzed iteratively to identify structures that represent rules and structures that specify states such as assignment statements. Based on the legacy code forms analyzed in Section 6.6 and considering the language being analyzed, rule structures are if-then-else, while, and indexed for-loop code structures. Assignment statements are identified. Other structures (i.e., structures that are not rules and not assignments) are identified but left unclassified. Within the context of the *Spec* concept, these unclassified structures are left unmodified for later assessment if necessary and as appropriate.

3. The start and end of each rule, assignment, and unclassified structure are determined. Each rule is labeled and the code associated with that rule marked for further assessment on a subsequent iteration. Assignment statements are converted into temporal formulas and labeled. As appropriate, unclassified structures are labeled.

4. For each formula, the frame *W* and the variable set *V* (i.e., variables used to calculate those variables in the frame, as described in Chapter 6) are identified. *W* and *V* for each rule are calculated later, after the analyses of all contributing formulas are completed.

5. A single sequence of labels is created, identifying and ordering the formulas, rules, and other unclassified structures visible at the current level.

6. Adjacent formulas are assessed to determine if any other reductions are possible or appropriate. Unclassified structures are assessed, and aggregated, deleted, processed, and/or left unchanged, as appropriate.

7. With the next iterative pass, the code associated with the first rule of the above sequence is assessed. This code is analyzed to identify the elements used to specify the rule condition and the elements used to specify the rule state, including new rules.

8. The code representing the rule state is processed, as described above starting at (3) above, and a sequence of formulas and rules is generated reflecting the code structures visible at that level.

9. Each element of this sequence is processed iteratively until all rules have been reduced to their component formulas.

10. The next rule in the original sequence at (2) above is processed using this procedure.

11. This process is repeated until all code has been processed and transformed to a sequence of formulas, rules, and unclassified structures.

Using this process and with the first iterative analysis of the target code, the following sequence of formulas, rules, and unclassified structures are identified.

```
procedure printerrorline(var Lbuf: linebufrec);
var
   Column,I,J,Num: integer;
begin
   $f_{0a}$;
   with Lbuf do
   begin
      $f_{pl}$;
      $f_{0b}$;
      $rule_1$;
      $f_{0c}$;$f_{0d}$;$f_{0e}$;
   end {of with}
end; {of procedure printerrorline}
```

where:

$f_{0a} \triangleq \circ\text{Column} = 0$

$f_{pl} \triangleq \text{printline(Lbuf)}$

$f_{0b} \triangleq \circ\text{write('*****', ' ')}$

$f_{0c} \triangleq \circ\text{writeln}$

$f_{0d} \triangleq \circ\text{lineerror} = \textit{false}$

$f_{0e} \triangleq \circ\text{fileerror} = \textit{true}$

With these definitions, the operative portions of the procedure printerrorline can be described as follows:

$$f_0 \equiv f_{0a}\,;f_{pl}\,;f_{0b}\,;\,rule_1\,;f_{0c}\,;f_{0d}\,;f_{0e} \qquad (8.1\text{-}1)$$

In this expression of the legacy code, $f_{pl}$, representing the procedure printline(Lbuf), is unclassified with regard to rules and formulas. The specific code associated with $rule_1$ is described later and is transformed into the component formulas and/or rules in a subsequent iteration.

For $f_{0a}, f_{0b}, f_{0c}, f_{0d}$, and $f_{0e}$, the frame set $W$ for each formula is:

$W_{0a} = \{\text{Column}\}$

$W_{0b} = \{I/O_{\text{write}}\}$

$W_{0c} = \{I/O_{\text{write}}\}$

$W_{0d} = \{\text{lineerror}\}$

$W_{0e}$ = {fileerror}

For this analysis, a variable $I/O_{\text{write}}$ is imposed to describe the system service that is updated by the PASCAL 'write,' 'writeln,' and similar commands. For $f_{0a}, f_{0b}, f_{0c}, f_{0d}$, and $f_{0e}$, the variable set $V$ for each formula is:

$$V_{0a} = \varnothing$$
$$V_{0b} = \varnothing$$
$$V_{0c} = \varnothing$$
$$V_{0d} = \varnothing$$
$$V_{0e} = \varnothing$$

In the preceding analysis, *rule₁* represents the following code:

```
for I:=1 to length +1 do
   if eline[I] < > errnone then
   begin
      errorset:= errorset+[eline[I]];
      Num:= ord(eline[I]);
      if I > Column then
      begin
         for J:= Column + 2 to I do write(' ');
         write('↑');Column:= I
      end
      else begin write(',');Column:= Column + 1 end
      write(Num:1);
      Column:= Column + l;
      if Num > 9 then Column:= Column + l;
      eline[I]:= errnone
      end; {of if and for}
```

Based on the next iterative analysis of this code, $rule_1$ is an indexed for-loop. Using the indexed for-loop rule-form presented in Section 6.6.3, $rule_1$ is described as:

$$rule_1 \triangleq \text{for I:=1 to length +1 do } rule_2 \qquad (8.1\text{-}2)$$

The specific code associated with $rule_2$ is described later and is transformed into the component formulas and/or rules in a subsequent iteration.

As an indexed for-loop and consistent with Section 6.6.3, $rule_1$ is transformed to a while structure as:

$$rule_1 \equiv f_{1a} \,;\, rule_{1'} \tag{8.1-3}$$

where:

$$\begin{aligned} f_{1a} &\triangleq (\circ I = 1) \\ rule_{1'} &\triangleq \text{while } (I \leq \text{length} + 1) \text{ do } (rule_2 \,;\, \circ I = I + 1) \end{aligned}$$

Using the definitions presented in Section 6.6.2, the while structure $rule_{1'}$ is transformed to:

$$\begin{aligned} rule_{1'} \equiv\ & (((I \leq \text{length} + 1) \wedge \circ rule_2 \,;\, \circ I = I + 1) \,;\, rule_{1'}) \\ & \vee (\neg(I \leq \text{length} + 1) \wedge \text{empty}) \end{aligned} \tag{8.1-4}$$

With these transformations, $rule_1$ can be described as:

$$rule_1 \equiv f_{1a} \,;\, rule_{1'} \tag{8.1-5}$$

where:

$$\begin{aligned} f_{1a} &\triangleq \circ I = 1 \\ rule_{1'} &\triangleq ((w_{C1'} \wedge \circ rule_2 \,; f_{1b}) \,;\, rule_{1'}) \vee (\neg w_{C1'} \wedge \text{empty})) \\ w_{C1'} &\triangleq I \leq \text{length} + 1 \\ f_{1b} &\triangleq \circ I = I + 1 \end{aligned}$$

For each of the above non-rule formulas, the sets $V$ and $W$ are determined based on their respective definitions, and the database is updated accordingly. The determination of $V$ and $W$ for $rule_{1'}$ is deferred until all contributory formulas are identified.

In $rule_{1'}$, the rule condition $w_{C1'}$ is a state formula. Therefore, to transform $rule_{1'}$ to a simpler form, StateAndNextChop is applied to $rule_{1'}$ to yield:

$$rule_{1'} \triangleq (w_{C1'} \wedge \circ rule_2 \,; f_{1b} \,;\, rule_{1'}) \vee (\neg w_{C1'} \wedge \text{empty}) \tag{8.1-6}$$

In the specification of $rule_{1'}$, $rule_2$ represents the following code:

```
if eline[I] < > errnone then
begin
   errorset:= errorset+[eline[I]];
   Num:= ord(eline[I]);
   if I > Column then
   begin
      for J:= Column + 2 to I do write(' ');
      write('↑');Column:= I
   end
   else begin write(',');Column:= Column + 1 end
   write(Num:1);
   Column:= Column + l;
   if Num > 9 then Column:= Column + l;
   eline[I]:= errnone
end; {of if and for}
```

Based on the next iterative analysis of this code, *rule₂* is an if-then-else rule structure. Consistent with the definition presented in Section 6.6.1, *rule₂* is described as follows:

$$rule_2 \equiv (w_{C2} \wedge \circ f_3) \vee (\neg w_{C2} \wedge \text{empty}) \qquad (8.1\text{-}7)$$

where:

$w_{C2} \quad \triangleq$ eline(I) $\neq$ errnone

For each of the above non-rule formulas, the sets *V* and *W* are determined based on their respective definitions, and the database is updated accordingly. The determination of *V* and *W* for *rule₂* is deferred until all contributory formulas are identified.

In the specification of $rule_2$, $f_3$ represents the following code:

```
errorset:= errorset+[eline[I]];
Num:= ord(eline[I]);
if I > Column then
begin
   for J:= Column + 2 to I do write(' ');
   write('↑');Column:= I
end
else begin write(',');Column:= Column + 1 end
write(Num:1);
Column:= Column + l;
if Num > 9 then Column:= Column + l;
eline[I]:= errnone
```

Based on the next iterative analysis of this code, $f_3$ is described by the following sequence:

$$f_3 \equiv f_{3a} ; f_{3b} ; rule_3 ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \tag{8.1-8}$$

where:

$f_{3a} \triangleq \circ\text{errorset} = \text{errorset}+(\text{eline}(I))$

$f_{3b} \triangleq \circ\text{Num} = \text{ord}(\text{eline}(I))$

$f_{3c} \triangleq \circ\text{write}(\text{Num})$

$f_{3d} \triangleq \circ\text{Column} = \text{Column} + 1$

$f_{3e} \triangleq \circ\text{eline}(I) = \text{errnone}$

For each of the above non-rule formulas, the sets $V$ and $W$ are determined based on their respective definitions, and the database is updated accordingly. The determination of $V$ and $W$ for $rule_3$ and $rule_4$ is deferred until all contributory formulas are identified.

With the above expansion of $f_3$, $rule_2$ is restated as:

$$\begin{aligned} rule_2 \equiv & (w_{C2} \wedge \circ f_{3a} ; f_{3b} ; rule_3 ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}) \\ & \vee (\neg w_{C2} \wedge \text{empty}) \end{aligned} \tag{8.1-9}$$

In the specification of $rule_2$, $rule_3$ represents the following code:

```
if I > Column then
begin
   for J:= Column + 2 to I do write(' ');
   write('↑');Column:= I
end
else begin write(',');Column:= Column + 1 end
```

Based on the next iterative analysis of this code, $rule_3$ is an if-then-else rule structure. Consistent with the definition presented in Section 6.6.1, $rule_3$ is described as follows:

$$rule_3 \equiv (w_{C3} \wedge \circ rule_5 ; f_{4a} ; f_{4b}) \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d}) \tag{8.1-10}$$

where:

$w_{C3} \quad \triangleq I > \text{Column}$

$f_{4a} \quad \triangleq \circ\text{write}('\uparrow')$

$f_{4b} \quad \triangleq \circ\text{Column} = \text{I}$

$f_{4c} \quad \triangleq \circ\text{write}(',')$

$f_{4d} \quad \triangleq \circ\text{Column} = \text{Column} + \text{I}$

For each of the above non-rule formulas, the sets *V* and *W* are determined based on their respective definitions, and the database is updated accordingly. The determination of *V* and *W* for $rule_5$ is deferred until all contributory formulas are identified.

In the specification of $rule_3$, $rule_5$ represents the following code:

```
for J:= Column + 2 to I do write(' ');
```

Based on the next iterative analysis of this code, $rule_5$ is an indexed for-loop. Using the indexed for-loop rule-form presented in Section 6.6.3, $rule_5$ is described as:

$$rule_5 \equiv f_{6a} \,;\, rule_{5'} \tag{8.1-11}$$

where:

$f_{6a} \quad \triangleq \circ\text{J} = \text{Column} + 2$

$rule_{5'} \quad \triangleq \text{while } w_{C5'} \text{ do } f_{6b}$

$w_{C5'} \quad \triangleq \text{J} \leq \text{I}$

$f_{6b} \quad \triangleq \circ\text{write}(' \ ')$

Using the definitions presented in Section 6.6.2, the while structure $rule_{5'}$ is transformed to:

$$rule_{5'} \equiv ((w_{C5'} \wedge \circ f_{6b} \,; f_{6c}) \,; rule_{5'}) \vee (\neg w_{C5'} \wedge \text{empty}) \tag{8.1-12}$$

where:

$f_{6c} \quad \triangleq \circ\text{J} = \text{J} + 1$

For each of the above non-rule formulas, the sets *V* and *W* are determined based on their respective definitions, and the database is updated accordingly. The determination of *V* and *W* for $rule_{5'}$ is deferred until all contributory formulas are identified.

In $rules_{5'}$, the rule condition $w_{C5'}$ is a state formula. Therefore, to transform $rules_{5'}$ to a simpler form, StateAndNextChop is applied to $rules_{5'}$ to yield:

$$rules_{5'} \equiv (w_{C5'} \wedge \circ f_{6b} \,; f_{6c} \,; rules_{5'}) \vee (\neg w_{C5'} \wedge \text{empty}) \qquad (8.1\text{-}13)$$

Returning to and completing the specification of $rule_2$, $rule_4$ represents the following code:

if Num > 9 then Column:= Column + 1;

Based on the next iterative analysis of this code, $rule_4$ is an if-then-else rule structure. Consistent with the definition presented in Section 6.6.1, $rule_4$ is described as follows:

$$rule_4 \equiv (w_{C4} \wedge \circ f_{5a}) \vee (\neg w_{C4} \wedge \text{empty}) \qquad (8.1\text{-}14)$$

where:

$w_{C4} \quad \triangleq \text{Num} > 9$

$f_{5a} \quad \triangleq \circ\text{Column} = \text{Column} + 1$

For each of the above non-rule formulas, the sets $V$ and $W$ are determined based on their respective definitions, and the database is updated accordingly.

With the identification of all rules and all formulas that compose these rules, the frames associated with each rule can be determined. For a given rule, the frame of that rule is the set of variables that are modified by that rule, and is the union of all frames for the formulas and other rules that are part of that rule.

For example, $rules_{5'}$ has been previously defined as:

$$rules_{5'} \equiv ((w_{C5'} \wedge \circ f_{6b} \,; f_{6c}) \,; rules_{5'}) \vee (\neg w_{C5'} \wedge \text{empty}) \qquad (8.1\text{-}15)$$

Therefore, the frame for $rules_{5'}$ is:

$$W_{rules_{5'}} = W_{C5'} \cup W_{6b} \cup W_{6c} \qquad (8.1\text{-}16)$$

Substituting the values for the various frames yields:

$$W_{rule_{5'}} = \varnothing \cup \{I/O_{\text{write}}\} \cup \{\text{J}\} \qquad (8.1\text{-}17a)$$
$$= \{I/O_{\text{write}}, \text{J}\} \qquad (8.1\text{-}17b)$$

The frames for all rules are determined using this approach and the database is updated accordingly. The *V* set is determined for each rule in a similar manner.

Summarizing these analyses, the following rules have been extracted from the legacy code:

$$f_0 \equiv f_{0a} ; f_{pl} ; f_{0b} ; rule_1 ; f_{0c} ; f_{0d} ; f_{0e}$$
$$rule_1 \equiv f_{1a} ; rule_{1'}$$
$$rule_{1'} \equiv (w_{C1'} \wedge \circ rule_2 ; f_{1b} ; rule_{1'}) \vee (\neg w_{C1'} \wedge \text{empty})$$
$$rule_2 \equiv (w_{C2} \wedge \circ f_{3a} ; f_{3b} ; rule_3 ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}) \vee (\neg w_{C2} \wedge \text{empty})$$
$$rule_3 \equiv (w_{C3} \wedge \circ rule_5 ; f_{4a} ; f_{4b}) \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d})$$
$$rule_4 \equiv (w_{C4} \wedge \circ f_{5a}) \vee (\neg w_{C4} \wedge \text{empty})$$
$$rule_5 \equiv f_{6a} ; rule_{5'}$$
$$rule_{5'} \equiv ((w_{C5'} \wedge \circ f_{6b} ; f_{6c}) ; rule_{5'}) \vee (\neg w_{C5'} \wedge \text{empty})$$

The database associated with these extracted rules is presented in Table 8.1-1. Using the concepts described in Section 7.3, a statechart, based on and representing these extracted rules, is presented in Figure 8.1-1. Because this statechart is based on these extracted rules, the database presented in Table 8.1-1 is applicable to the statechart in Figure 8.1-1.

As a demonstration of the coordinated use of these extracted rules, the associated database, and the rule algebra presented in this thesis, the rules extracted from this legacy code are analyzed for those formulas that result in the writing to an output device. As previously discussed, the variable $I/O_{write}$ is used as a frame variable to describe the system service that is updated by the PASCAL 'write,' 'writeln,' and similar commands. Therefore, the database is searched for formulas that have only $I/O_{write}$ as the frame.

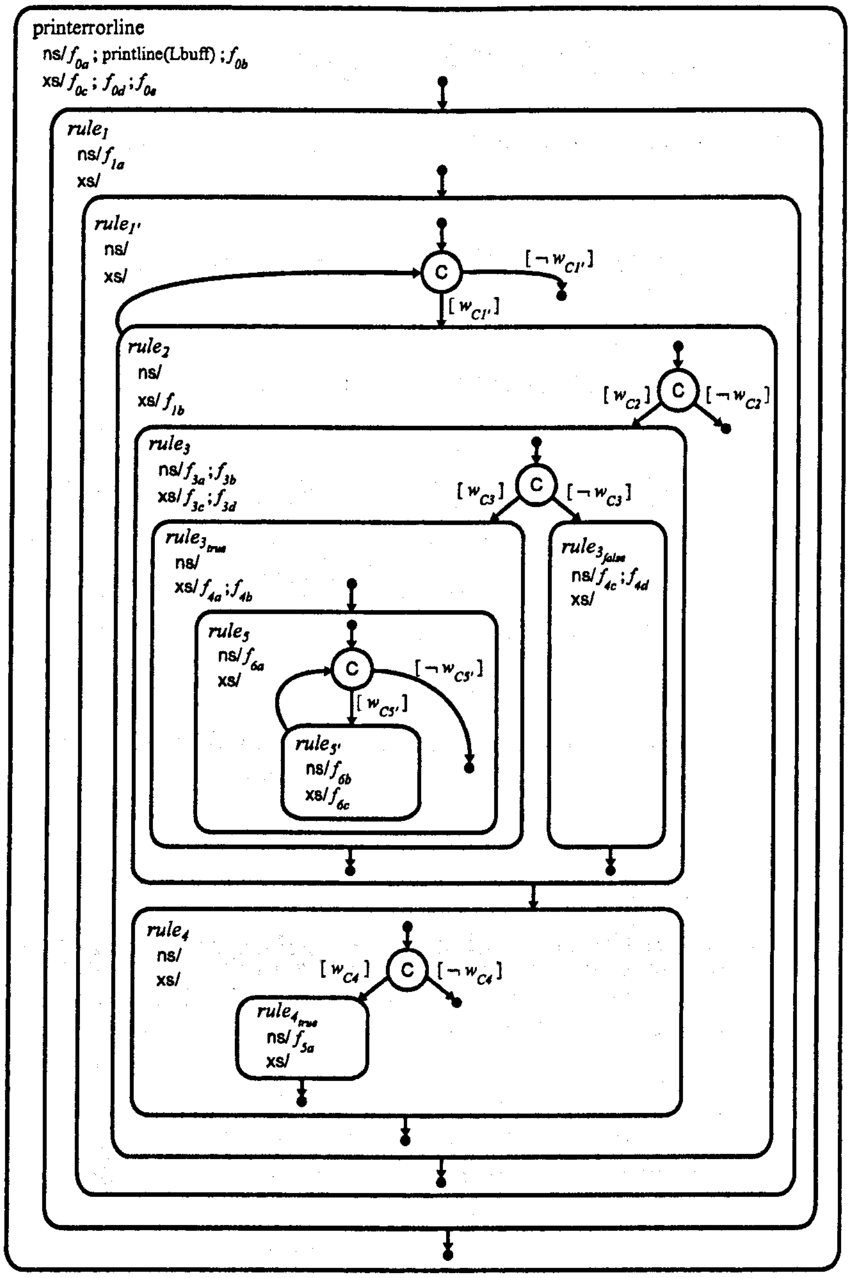


Figure 8.1-1: Statechart for Procedure printerrorline Legacy Code

Based on this search of the database, six formulas meet this criterion – $f_{0b}$, $f_{0c}$, $f_{3c}$, $f_{4a}$, $f_{4c}$, and $f_{6a}$. Using the database, these formulas have primary membership in $f_0$, *rule*$_2$, *rule*$_3$, and *rule*$_{5'}$. Therefore, $f_0$, *rule*$_2$, *rule*$_3$, and *rule*$_{5'}$ must be analyzed. However, these four rules and formulas are not directly connected. Referencing the database, *rule*$_{5'}$ is not a primary member of *rule*$_3$. Instead, *rule*$_{5'}$ is a member of *rule*$_5$

and $rule_5$ is a member of $rule_3$. Similarly, $rule_2$ is not a primary member of $rule_3$. Instead, $rule_2$ is a member of $rule_{1'}$, $rule_{1'}$ is a member of $rule_1$, and $rule_1$ is a member of $f_0$. Therefore, $rule_5$, $rule_{1'}$, and $rule_1$ must be included in the analysis. Summarizing, seven rules – $f_0$, $rule_1$, $rule_{1'}$, $rule_2$, $rule_3$, $rule_5$, and $rule_{5'}$ – are analyzed regarding formulas that result in the writing to an output device. Because $rule_4$ does not include any I/O activities, as demonstrated in the database by the absence of $I/O_{write}$ in the frame of any formula with a primary membership to $rule_4$, $rule_4$ is not considered in this analysis.

In this analysis, these seven rules are transformed to create a single rule structure, and this rule structure is used to assess the specific rule conditions that are associated with specific I/O activities. To implement these rule transformations, an additional lemma is introduced – TwoChopRulesImp3. TwoChopRulesImp3 is a continuation of the series TwoChopRulesImp and TwoChopRulesImp2 introduced in Section 7.2.

LEMMA: TwoChopRulesImp3

$\vdash f_0 ; (f_1 \wedge f_2) ; (f_3 \wedge f_4) ; f_5$ implies $\vdash f_0 ; ((f_1 ; f_3) \wedge (f_2 ; f_4 ; f_5))$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 ; (f_1 \wedge f_2) ; (f_3 \wedge f_4) ; f_5$ | premise |
| 2 | $(f_1 \wedge f_2) ; (f_3 \wedge f_4) ; f_5$ | CP assumption |
| 3 | $(f_1 ; f_3) \wedge (f_2 ; f_4 ; f_5)$ | 2, TwoChopRulesImp2 |
| 4 | $(f_1 \wedge f_2) ; (f_3 \wedge f_4) ; f_5 \supset (f_1 ; f_3 \wedge f_2 ; f_4 ; f_5)$ | 2-3, $\supset$ introduction |
| 5 | $f_0 ; (f_1 \wedge f_2) ; (f_3 \wedge f_4) ; f_5$ $\supset f_0 ; ((f_1 ; f_3) \wedge (f_2 ; f_4 ; f_5))$ | 4, ITL (RightChopImpChop) |
| 6 | $f_0 ; ((f_1 ; f_3) \wedge (f_2 ; f_4 ; f_5))$ | 1, 5, MP |

The general transformation strategy for the analysis of this set of extracted legacy code rules is similar to that implemented in the transformation of the specification in Section 7.2. Each contributory rule is separated into the component rule condition and rule state, and then the components are added in order into the aggregate description of the possible system behaviors. Because this transformation considers multiple rules and therefore multiple behaviors, the resulting alternative behaviors are expressed disjunctively. The target rules are processed in reverse order, that is, from the deepest rule upwards. In this way, behaviors are transformed systematically, and each subsequent behavior associated with a specific rule rests on the behavior(s) defined by that rule's component rules.

This transformation rests on seven premises that reflect the rules extracted from legacy code that directly or indirectly include the variable $I/O_{write}$ – $f_0$, $rule_1$, $rule_{1'}$, $rule_2$, $rule_3$, $rule_5$, and $rule_{5'}$. Because the deepest rule, $rule_5$ (including the subrule $rule_{5'}$) includes no other rules and therefore, by definition, totally describes all behaviors associated with $rule_5$, $rule_5$ needs no transformation. Therefore, $rule_3$ is transformed first and incorporates the behaviors associated with $rule_5$. Then, $rule_2$ is transformed and incorporates the behaviors derived from $rule_3$ and $rule_5$. Then, $rule_1$ (including the subrule $rule_{1'}$) is transformed and incorporates the behaviors derived from $rule_2$, $rule_3$, and $rule_5$. Finally, $f_0$ is transformed and incorporates the behaviors derived from $rule_1$, $rule_2$, $rule_3$, and $rule_5$. This formal transformation is presented in Appendix C.

With this transformation and based on the premises $f_0$, $rule_1$, $rule_{1'}$, $rule_2$, $rule_3$, $rule_5$, and $rule_{5'}$ as extracted from the legacy code, the following disjunctive rule structure is concluded:

$$f_{0a} \,;f_{pl} \,;f_{0b} \,;f_{1a} \,;( \tag{8.1-18a}$$

$$\begin{aligned}&(w_{C1'} \,; w_{C2} \,; w_{C3} \,; w_{C5'}\\ &\quad \wedge \circ\circ f_{3a} \,;f_{3b} \,; \circ f_{6a} \,; \circ f_{6b} \,;f_{6c} \,; rule_{5'} \,;f_{4a} \,;f_{4b} \,;\\ &\quad f_{3c} \,;f_{3d} \,; rule_4 \,;f_{3e} \,;f_{1b} \,; rule_{1'} \,;f_{0c} \,;f_{0d} \,;f_{0e})\end{aligned} \tag{8.1-18b}$$

$$\begin{aligned}&\vee (w_{C1'} \,; w_{C2} \,; w_{C3} \,; \neg w_{C5'}\\ &\quad \wedge \circ\circ f_{3a} \,;f_{3b} \,; \circ f_{6a} \,;f_{4a} \,;f_{4b} \,;f_{3c} \,;f_{3d} \,; rule_4 \,;\\ &\quad f_{3e} \,;f_{1b} \,; rule_{1'} \,;f_{0c} \,;f_{0d} \,;f_{0e})\end{aligned} \tag{8.1-18c}$$

$$\begin{aligned}&\vee (w_{C1'} \,; w_{C2} \,; \neg w_{C3}\\ &\quad \wedge \circ\circ f_{3a} \,;f_{3b} \,; \circ f_{4c} \,;f_{4d} \,;f_{3c} \,;f_{3d} \,; rule_4 \,;\\ &\quad f_{3e} \,;f_{1b} \,; rule_{1'} \,;f_{0c} \,;f_{0d} \,;f_{0e})\end{aligned} \tag{8.1-18d}$$

$$\vee (w_{C1'} \,; \neg w_{C2} \wedge \circ f_{1b} \,; rule_{1'} \,;f_{0c} \,;f_{0d} \,;f_{0e}) \tag{8.1-18e}$$

$$\vee (\neg w_{C1'} \wedge f_{0c} \,;f_{0d} \,;f_{0e})) \tag{8.1-18f}$$

Although (8.1-18) is a single structure, each component is numbered individually to facilitate discussion.

With the transformation presented in (8.1-18), the behavior associated with the legacy code is described as a sequence of chopped formulas at (8.1-18a) and then one of the five disjunctively connected general-form rules presented at (8.1-18b) through (8.1-18f). Specifically which of these five disjunctively-connected general-form rules describes the specific behavior in a given circumstance depends on the verity of the rule conditions $w_{C1'}$, $w_{C2}$, $w_{C3}$, and $w_{C5'}$ under that given circumstance. The transformation (8.1-18) provides an orderly basis for assessing and understanding the specific behavior associated with the verities for each condition. For example, $\neg w_{C1'}$ results in the behavior specified by (8.1-18f), whereas $w_{C1'}$ is associated with the behaviors specified by (8.1-18b) through (8.1-18e). Similarly, $\neg w_{C2}$ results in the behavior specified in (8.1-18f), whereas $w_{C2}$ is associated with the behaviors specified in (8.1-18b) through (8.1-18e), etc.

This analysis and understanding of this transformation can be facilitated by restoring specific formulas of interest. Referencing the various substitutions performed earlier in this section, the rule conditions in (8.1-18) are represented by one or more of the following state formulas:

$$\begin{aligned} w_{C1'} &\equiv (\mathrm{I} \leq \mathrm{length} + 1) \\ w_{C2} &\equiv (\mathrm{eline(I)} \neq \mathrm{ernone}) \\ w_{C3} &\equiv (\mathrm{I} > \mathrm{Column}) \\ w_{C5'} &\equiv (\mathrm{J} \leq \mathrm{I}) \end{aligned}$$

Based on an analysis of the database developed for this legacy code using the rule algebra, the following formulas result in the writing to an I/O device:

$$\begin{aligned} f_{0b} &\equiv \text{owrite('*****', ' ')} \\ f_{0c} &\equiv \text{owriteln} \\ f_{3c} &\equiv \text{owrite(Num)} \\ f_{4a} &\equiv \text{owrite('}\uparrow\text{')} \\ f_{4c} &\equiv \text{owrite(',')} \\ f_{6b} &\equiv \text{owrite(' ')} \end{aligned}$$

Substituting the above rule conditions and I/O-related formulas into (8.1-18) yields:

$$f_{0a} \,; f_{pl} \,; \text{owrite('*****', ' ')} \,; f_{1a} \,; \{ \qquad \text{(8.1-19a)}$$

$$((\mathrm{I} \leq \mathrm{length} + 1) \,; (\mathrm{eline(I)} \neq \mathrm{ernone}) \,; (\mathrm{I} > \mathrm{Column}) \,; (\mathrm{J} \leq \mathrm{I})$$

$$\begin{aligned}&\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ\circ \text{write}(\text{' '}) ; f_{6c} ; \mathit{rule}_{5'} ; \circ \text{write}(\text{'}\uparrow\text{'}) ; f_{4b} ;\\ &\quad \circ \text{write}(\text{Num}) ; f_{3d} ; \mathit{rule}_4 ; f_{3e} ;\\ &\quad f_{1b} ; \mathit{rule}_{1'} ; \circ \text{writeln} ; f_{0d} ; f_{0e})\end{aligned} \qquad (8.1\text{-}19b)$$

$$\begin{aligned}&\vee ((I \leq \text{length} + 1) ; (\text{eline}(I) \neq \text{errnone}) ; (I > \text{Column}) ; \neg(J \leq I)\\ &\quad \wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ \text{write}(\text{'}\uparrow\text{'}) ; f_{4b} ; \circ \text{write}(\text{Num}) ; f_{3d} ; \mathit{rule}_4 ;\\ &\quad f_{3e} ; f_{1b} ; \mathit{rule}_{1'} ; \circ \text{writeln} ; f_{0d} ; f_{0e})\end{aligned} \qquad (8.1\text{-}19c)$$

$$\begin{aligned}&\vee ((I \leq \text{length} + 1) ; (\text{eline}(I) \neq \text{errnone}) ; \neg(I > \text{Column})\\ &\quad \wedge \circ\circ f_{3a} ; f_{3b} ; \circ\circ \text{write}(\text{','}) ; f_{4d} ; \circ \text{write}(\text{Num}) ; f_{3d} ; \mathit{rule}_4 ;\\ &\quad f_{3e} ; f_{1b} ; \mathit{rule}_{1'} ; \circ \text{writeln} ; f_{0d} ; f_{0e})\end{aligned} \qquad (8.1\text{-}19d)$$

$$\begin{aligned}&\vee ((I \leq \text{length} + 1) ; \neg(\text{eline}(I) \neq \text{errnone})\\ &\quad \wedge \circ f_{1b} ; \mathit{rule}_{1'} ; \circ \text{writeln} ; f_{0d} ; f_{0e})\end{aligned} \qquad (8.1\text{-}19e)$$

$$\vee (\neg(I \leq \text{length} + 1) \wedge \circ \text{writeln} ; f_{0d} ; f_{0e})\} \qquad (8.1\text{-}19f)$$

With these substitutions, the value of this transformation is demonstrated. I/O operations are identified in the order they occur relative to the satisfaction of the various rule conditions. For example, as described in (8.1-19b), the I/O operation write(' ') occurs only when the rule condition $(I \leq \text{length} + 1)$ ; $(\text{eline}(I) \neq \text{errnone})$ ; $(I > \text{Column})$ ; $(J \leq I)$ is satisfied. As another example, the I/O operations write('↑') and write(Num) only occur together and in that order in (8.1-19b) and (8.1-19c), and require the satisfaction of the rule condition $(I \leq \text{length} + 1)$ ; $(\text{eline}(I) \neq \text{errnone})$ ; $(I > \text{Column})$. Inspection of the rule conditions associated with (8.1-19b) and (8.1-19c) reveals that the verity of the rule condition $(J \leq I)$ does not affect the occurrence of I/O operations write('↑') and write(Num). As a final example, the I/O operation writeln is associated with all disjuncts and therefore is not dependent on the satisfaction of a specific set of rule conditions.

Whereas numerous other transformations and analyses are possible using these extracted rules, the rule extraction and analysis presented in this section demonstrates the use and applicability of this rule model and rule algebra in the assessment of legacy code.

Table 8.1-1 Legacy code analysis database

| Formula | Description | *W* | *V* | Primary Membership |
|---|---|---|---|---|
| $f_0$ | $f_{0a}$ ; $f_{pl}$ ; $f_{0b}$ ; $rule_1$ ; $f_{0c}$ ; $f_{0d}$ ; $f_0$ | Column, eline(I), errorset, fileerror, I, $I/O_{\text{write}}$, J, lineerror, Num | Column, errnone, eline(I), errorset, I, J, length, Num | – |
| $f_{0a}$ | ∘Column = 0 | Column | ∅ | $f_0$ |
| $f_{pl}$ | printline(Lbuf) | – unclassified – | – unclassified – | $f_0$ |
| $f_{0b}$ | ∘write('*****', ' ') | $I/O_{\text{write}}$ | ∅ | $f_0$ |
| $f_{0c}$ | ∘writeln | $I/O_{\text{write}}$ | ∅ | $f_0$ |
| $f_{0d}$ | ∘lineerror = *false* | lineerror | ∅ | $f_0$ |
| $f_{0e}$ | ∘fileerror = *true* | fileerror | ∅ | $f_0$ |
| $rule_1$ | $f_{1a}$ ; $rule_{1'}$ | Column, eline(I), errorset, I, $I/O_{\text{write}}$, J, Num | Column, errnone, eline(I), errorset, I, J, length, Num | $f_0$ |
| $f_{1a}$ | ∘I = 1 | I | ∅ | $rule_1$ |
| $rule_{1'}$ | $rule_{1'\text{-true}} \vee rule_{1'\text{-false}}$ | Column, eline(I), errorset, I, $I/O_{\text{write}}$, J, Num | Column, errnone, eline(I), errorset, I, J, length, Num | $rule_1$ |
| $rule_{1'\text{-true}}$ | $w_{C1'} \wedge$ ∘$rule_2$ ; $f_{1b}$ ; $rule_{1'}$ | Column, eline(I), errorset, I, $I/O_{\text{write}}$, J, Num | Column, errnone, eline(I), errorset, I, J, length, Num | $rule_{1'}$ |
| $rule_{1'\text{-false}}$ | $\neg w_{C1'} \wedge$ empty | ∅ | I, length | $rule_{1'}$ |
| $w_{C1'}$ | I $\leq$ length +1 | ∅ | I, length | $rule_{1'\text{-true}}$, $rule_{1'\text{-false}}$ |
| $f_{1b}$ | ∘I = I + 1 | I | I | $rule_{1'\text{-true}}$ |
| $rule_2$ | $rule_{2\text{-true}} \vee rule_{2\text{-false}}$ | Column, eline(I), errorset, $I/O_{\text{write}}$, J, Num | Column, errnone, eline(I), errorset, I, J, Num | $rule_{1'\text{-true}}$ |
| $rule_{2\text{-true}}$ | $w_{C2} \wedge$ ∘$f_{3a}$ ; $f_{3b}$ ; $rule_3$ ; $f_{3c}$ ; $f_{3d}$ ; $rule_4$ ; $f_{3e}$ | Column, eline(I), errorset, $I/O_{\text{write}}$, J, Num | Column, errnone, eline(I), errorset, I, J, Num | $rule_2$ |
| $rule_{2\text{-false}}$ | $\neg w_{C2} \wedge$ empty | ∅ | errnone, eline(I), I | $rule_2$ |
| $w_{C2}$ | eline(I) ≠ errnone | ∅ | errnone, eline(I), I | $rule_{2\text{-true}}$, $rule_{2\text{-false}}$ |
| $f_{3a}$ | ∘errorset = errorset+(eline(I)) | errorset | eline(I), errorset, I | $rule_{2\text{-true}}$ |
| $f_{3b}$ | ∘Num = ord(eline(I)) | Num | eline(I), I | $rule_{2\text{-true}}$ |
| $f_{3c}$ | ∘write(Num) | $I/O_{\text{write}}$ | Num | $rule_{2\text{-true}}$ |
| $f_{3d}$ | ∘Column = Column + 1 | Column | Column | $rule_{2\text{-true}}$ |
| $f_{3e}$ | ∘eline(I) = errnone | eline(I) | errnone | $rule_{2\text{-true}}$ |

Table 8.1-1 (continued) Legacy code analysis database

| Formula | Description | $W$ | $V$ | Primary Membership |
|---|---|---|---|---|
| $rule_3$ | $rule_{3\text{-}true} \vee rule_{3\text{-}false}$ | Column, $I/O_{\text{write}}$, J | Column, I, J | $rule_{2\text{-}true}$ |
| $rule_{3\text{-}true}$ | $w_{C3} \wedge \circ rule_5 \,; f_{4a} \,; f_{4b}$ | Column, $I/O_{\text{write}}$, J | Column, I, J | $rule_3$ |
| $rule_{3\text{-}false}$ | $\neg w_{C3} \wedge \circ f_{4c} \,; f_{4d}$ | Column, $I/O_{\text{write}}$ | Column, I | $rule_3$ |
| $w_{C3}$ | I > Column | $\varnothing$ | Column, I | $rule_{3\text{-}true}$, $rule_{3\text{-}false}$ |
| $f_{4a}$ | $\circ$write('↑') | $I/O_{\text{write}}$ | $\varnothing$ | $rule_{3\text{-}true}$ |
| $f_{4b}$ | $\circ$Column = I | Column | I | $rule_{3\text{-}true}$ |
| $f_{4c}$ | $\circ$write(',') | $I/O_{\text{write}}$ | $\varnothing$ | $rule_{3\text{-}false}$ |
| $f_{4d}$ | $\circ$Column = Column + I | Column | Column, I | $rule_{3\text{-}false}$ |
| $rule_4$ | $rule_{4\text{-}true} \vee rule_{4\text{-}false}$ | Column | Column, Num | $rule_{2\text{-}true}$ |
| $rule_{4\text{-}true}$ | $w_{C4} \wedge \circ f_{5a}$ | Column | Column, Num | $rule_4$ |
| $rule_{4\text{-}false}$ | $\neg w_{C4} \wedge$ empty | $\varnothing$ | Num | $rule_4$ |
| $w_{C4}$ | Num > 9 | $\varnothing$ | Num | $rule_{4\text{-}true}$, $rule_{4\text{-}false}$ |
| $f_{5a}$ | $\circ$Column = Column + 1 | Column | Column | $rule_{4\text{-}true}$ |
| $rule_5$ | $f_{6a} \,; rule_{5'}$ | $I/O_{\text{write}}$, J | Column, I, J | $rule_3$ |
| $rule_{5'}$ | $rule_{5'\text{-}true} \vee rule_{5'\text{-}false}$ | $I/O_{\text{write}}$, J | I, J | $rule_5$ |
| $rule_{5'\text{-}true}$ | $w_{C5'} \wedge \circ f_{6b} \,; f_{6c} \,; rule_{5'}$ | $I/O_{\text{write}}$, J | I, J | $rule_{5'}$ |
| $rule_{5'\text{-}false}$ | $\neg w_{C5'} \wedge$ empty | $\varnothing$ | I, J | $rule_{5'}$ |
| $w_{C5'}$ | $J \leq I$ | $\varnothing$ | I, J | $rule_{5'\text{-}true}$, $rule_{5'\text{-}false}$ |
| $f_{6a}$ | $\circ$J = Column + 2 | J | Column | $rule_5$ |
| $f_{6b}$ | $\circ$write(' ') | $I/O_{\text{write}}$ | $\varnothing$ | $rule_{5'\text{-}true}$ |
| $f_{6c}$ | $\circ$J = J + 1 | J | J | $rule_{5'\text{-}true}$ |

## 8.2 Representing WSL Program Slices as Rules

In this section, a Wide Spectrum Language (WSL) program is sliced, and rules are extracted from each slice and analyzed using the rule model and rule algebra developed in this research. Two different rules are extracted from this program in two separate slicing exercises. In the first slicing exercise, rules are extracted from the program slice and the rule algebra is applied to simply and clarify the extracted rule. In the second slicing exercise, rules are extracted from the program slice, and the rules are then conditioned and transformed using the rule algebra. The results of these rule transformations are compared with previous analyses of the same program.

The program analyzed herein is used to compute income tax and various tax-related amounts, including a non-taxable personal allowance, for a United Kingdom citizen for the tax year April 1998 to April 1999. The non-taxable personal allowance is dependent on specific attributes of a given citizen. Within this program, these attributes are represented by the variables 'age,' 'married,' 'widowed,' and 'blind.' This program, or an alternative language version, has been analyzed previously in Ward et al. (2005) and Fox et al. (2000). The WSL version of this program, as used in the research, is as follows:

```
IF age >= 75
THEN personal := 5980
  ELSE IF age >= 65
       THEN personal := 5720
       ELSE personal := 4335 FI FI;
IF age >= 65 AND income > 16800
   THEN VAR < t := personal - (income - 16800)/2 >:
       IF t > 4335
         THEN personal := t
         ELSE personal := 4335 FI ENDVAR FI;
IF blind = 1
  THEN personal := personal + 1380 FI;
IF married = 1 AND age >= 75
  THEN pc10 := 6692
  ELSE IF married = 1 AND age >= 65
       THEN pc10 := 6625
       ELSE IF married = 1 OR widow = 1
            THEN pc10 := 3470
            ELSE pc10 := 1500 FI FI FI;
IF married = 1 AND age >= 65 AND income > 16800
  THEN VAR < t := pc10 - ((income - 16800)/2) >:
        IF t > 3740
          THEN pc10 := t
          ELSE pc10 := 3740 FI ENDVAR FI;
IF income <= personal
  THEN tax := 0
  ELSE income := income - personal;
        IF income <= pc10
          THEN tax := income * rate10
          ELSE tax := pc10 * rate10;
               income := income - pc10;
               IF income <= 28000
                 THEN tax := tax + income * rate23
                 ELSE tax := tax + 28000 * rate23;
                      income := income - 28000;
                      tax := tax + income * rate40 FI FI FI
```

Slicing of this WSL program code was conducted using the FermaT transformation system. FermaT is an industrial-strength formal transformation system applicable to program comprehension and language migration. The FermaT transformation system is based on a comprehensive catalog of formal, proven program transformations that preserve or refine the semantics of a program while changing its form. By applying the appropriate program transformations to a program, the resulting transformed program is guaranteed to be equivalent to the original program logic. The FermaT transformation system, including theory and applications, is described in Ward (1999, 2000, 2004), and is available under the GNU General Public License (GPL) at http://www.cse.dmu.ac.uk/~mward/fermat.html.

Using the FermaT Syntactic_Slice transformation, the following slice was generated as a backward slice on the variable 'pc10':

```
IF married = 1 AND age >= 75
  THEN pc10 := 6692
  ELSE IF married = 1 AND age >= 65
         THEN pc10 := 6625
         ELSE IF married = 1 OR widow = 1
                THEN pc10 := 3470
                ELSE pc10 := 1500 FI FI FI;
IF married = 1 AND age >= 65 AND income > 16800
  THEN VAR < t := pc10 - (income - 16800) / 2 >:
       IF t > 3740 THEN pc10 := t ELSE pc10 := 3740 FI
       ENDVAR FI
```

Based on an inspection and analysis of the programming structures that comprise this slice, this slice on the variable 'pc10' can be represented as a sequence of chopped rules:

$$rule_{pc10} \equiv rule_{pc10\text{-}1} \; ; \; rule_{pc10\text{-}2} \qquad (8.2\text{-}1)$$

where:

```
rule_pc10-1 ≙   IF married = 1 AND age >= 75
                  THEN pc10 := 6692
                  ELSE IF married = 1 AND age >= 65
                      THEN pc10 := 6625
                      ELSE IF married = 1 OR widow = 1
                          THEN pc10 := 3470
                          ELSE pc10 := 1500 FI FI FI;

rule_pc10-2 ≙   IF married = 1 AND age >= 65 AND income > 16800
                  THEN VAR < t := pc10 - (income - 16800) / 2 >:
                     IF t > 3740 THEN pc10 := t ELSE pc10 := 3740 FI
                     ENDVAR FI
```

Applying the rule-form description of the if-then-else programming structure as presented in Section 6.6.1, $rule_{pc10\text{-}1}$ is described as:

$$\begin{aligned}&rule_{pc10\text{-}1} \equiv \\ &(\text{married} = 1 \wedge \text{age} \geq 75 \; \circ\text{pc10} = 6692) \\ &\vee (\neg(\text{married} = 1 \wedge \text{age} \geq 75) \wedge \circ rule_{pc10\text{-}1a})\end{aligned} \qquad (8.2\text{-}2)$$

In $rule_{pc10\text{-}1}$, $rule_{pc10\text{-}1a}$ is described as:

$$\begin{aligned}&rule_{pc10\text{-}1a} \equiv \\ &(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\text{pc10} = 6625) \\ &\vee (\neg(\text{married} = 1 \wedge \text{age} \geq 65) \wedge \circ rule_{pc10\text{-}1b})\end{aligned} \qquad (8.2\text{-}3)$$

In $rule_{pc10\text{-}1a}$, $rule_{pc10\text{-}1b}$ is described as:

$$\begin{aligned}&rule_{pc10\text{-}1b} \equiv \\ &((\text{married} = 1 \vee \text{widow} = 1) \wedge \circ\text{pc10} = 3470) \\ &\vee (\neg(\text{married} = 1 \vee \text{widow} = 1) \wedge \circ\text{pc10} = 1500)\end{aligned} \qquad (8.2\text{-}4)$$

Applying propositional logic and algebraic equivalence regarding negation and equality to $rule_{pc10\text{-}1b}$ yields:

$$\begin{aligned}
&rule_{pc10\text{-}1b} \equiv \\
&(\text{married} = 1 \wedge \circ\text{pc10} = 3470) \\
&\vee (\text{widow} = 1 \wedge \circ\text{pc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \circ\text{pc10} = 1500)
\end{aligned} \tag{8.2-5}$$

Regarding $rule_{pc10\text{-}1a}$ at (8.2-3), applying propositional logic and algebraic equivalences regarding the great-than-or-equal and negation operations yields:

$$\begin{aligned}
&rule_{pc10\text{-}1a} \equiv \\
&(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\text{pc10} = 6625) \\
&\vee (\text{married} \neq 1 \wedge \circ rule_{pc10\text{-}1b}) \\
&\vee (\text{age} < 65 \wedge \circ rule_{pc10\text{-}1b})
\end{aligned} \tag{8.2-6}$$

Substituting $rule_{pc10\text{-}1b}$ at (8.2-5) into $rule_{pc10\text{-}1a}$ at (8.2-6) yields:

$$\begin{aligned}
&rule_{pc10\text{-}1a} \equiv \\
&(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\text{pc10} = 6625) \\
&\vee (\text{married} \neq 1 \wedge \circ((\text{married} = 1 \wedge \circ\text{pc10} = 3470) \\
&\qquad\qquad \vee (\text{widow} = 1 \wedge \circ\text{pc10} = 3470) \\
&\qquad\qquad \vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \circ\text{pc10} = 1500))) \\
&\vee (\text{age} < 65 \wedge \circ((\text{married} = 1 \wedge \circ\text{pc10} = 3470) \\
&\qquad\qquad \vee (\text{widow} = 1 \wedge \circ\text{pc10} = 3470) \\
&\qquad\qquad \vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \circ\text{pc10} = 1500)))
\end{aligned} \tag{8.2-7}$$

Applying NextOrDistEqv, then NextAndDistEqv, and then propositional logic to $rule_{pc10\text{-}1a}$ at (8.2-7) yields:

$$\begin{aligned}
&rule_{pc10\text{-}1a} \equiv \\
&(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\text{pc10} = 6625) \\
&\vee (\text{married} \neq 1 \wedge \circ\text{married} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \circ\text{widow} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \circ\text{married} \neq 1 \wedge \circ\text{widow} \neq 1 \wedge \circ\circ\text{pc10} = 1500) \\
&\vee (\text{age} < 65 \wedge \circ\text{married} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\vee (\text{age} < 65 \wedge \circ\text{widow} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\vee (\text{age} < 65 \wedge \circ\text{married} \neq 1 \wedge \circ\text{widow} \neq 1 \wedge \circ\circ\text{pc10} = 1500)
\end{aligned} \tag{8.2-8}$$

Considering $rule_{pc10\text{-}1}$ as described at (8.2-2), applying propositional logic and algebraic equivalences regarding the great-than-or-equal and negation operators yields:

$$\begin{aligned}
&rule_{pc10\text{-}1} \equiv \\
&(\text{married} = 1 \wedge \text{age} \geq 75\ \circ\text{pc10} = 6692) \\
&\vee (\text{married} \neq 1 \wedge \circ rule_{pc10\text{-}1a}) \\
&\vee (\text{age} < 75 \wedge \circ rule_{pc10\text{-}1a})
\end{aligned} \qquad (8.2\text{-}9)$$

Substituting $rule_{pc10\text{-}1a}$ at (8.2-8) into $rule_{pc10\text{-}1}$ at (8.2-9) yields:

$$\begin{aligned}
&rule_{pc10\text{-}1} \equiv \\
&(\text{married} = 1 \wedge \text{age} \geq 75\ \circ\text{pc10} = 6692) \\
&\vee (\text{married} \neq 1 \wedge \circ((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\text{pc10} = 6625) \\
&\qquad \vee (\text{married} \neq 1 \wedge \circ\text{married} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{married} \neq 1 \wedge \circ\text{widow} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{married} \neq 1 \wedge \circ\text{married} \neq 1 \wedge \circ\text{widow} \neq 1 \\
&\qquad\quad \wedge \circ\circ\text{pc10} = 1500) \\
&\qquad \vee (\text{age} < 65 \wedge \circ\text{married} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{age} < 65 \wedge \circ\text{widow} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{age} < 65 \wedge \circ\text{married} \neq 1 \wedge \circ\text{widow} \neq 1 \\
&\qquad\quad \wedge \circ\circ\text{pc10} = 1500))) \\
&\vee (\text{age} < 75 \wedge \circ((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\text{pc10} = 6625) \\
&\qquad \vee (\text{married} \neq 1 \wedge \circ\text{married} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{married} \neq 1 \wedge \circ\text{widow} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{married} \neq 1 \wedge \circ\text{married} \neq 1 \wedge \circ\text{widow} \neq 1 \\
&\qquad\quad \wedge \circ\circ\text{pc10} = 1500) \\
&\qquad \vee (\text{age} < 65 \wedge \circ\text{married} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{age} < 65 \wedge \circ\text{widow} = 1 \wedge \circ\circ\text{pc10} = 3470) \\
&\qquad \vee (\text{age} < 65 \wedge \circ\text{married} \neq 1 \wedge \circ\text{widow} \neq 1 \\
&\qquad\quad \wedge \circ\circ\text{pc10} = 1500)))
\end{aligned} \qquad (8.2\text{-}10)$$

Applying NextOrDistEqv and then NextAndDistEqv yields:

$$
\begin{aligned}
&\mathit{rule}_{pc10\text{-}1} \equiv \\
&(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \text{opc10} = 6692) \\
&\vee (\text{married} \neq 1 \wedge ((\text{omarried} = 1 \wedge \text{oage} \geq 65 \wedge \text{oopc10} = 6625) \\
&\qquad\qquad \vee (\text{omarried} \neq 1 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{omarried} \neq 1 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{omarried} \neq 1 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad\qquad\quad \wedge \text{ooopc10} = 1500) \\
&\qquad\qquad \vee (\text{oage} < 65 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{oage} < 65 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{oage} < 65 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad\qquad\quad \wedge \text{ooopc10} = 1500))) \\
&\vee (\text{age} < 75 \wedge ((\text{omarried} = 1 \wedge \text{oage} \geq 65 \wedge \text{oopc10} = 6625) \\
&\qquad\qquad \vee (\text{omarried} \neq 1 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{omarried} \neq 1 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{omarried} \neq 1 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad\qquad\quad \wedge \text{ooopc10} = 1500) \\
&\qquad\qquad \vee (\text{oage} < 65 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{oage} < 65 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\qquad\qquad \vee (\text{oage} < 65 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad\qquad\quad \wedge \text{ooopc10} = 1500))) \qquad (8.2\text{-}11)
\end{aligned}
$$

Applying propositional logic to (8.2-11) yields:

$$
\begin{aligned}
&\mathit{rule}_{pc10\text{-}1} \equiv \\
&(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \text{opc10} = 6692) \\
&\vee (\text{married} \neq 1 \wedge \text{omarried} = 1 \wedge \text{oage} \geq 65 \wedge \text{oopc10} = 6625) \\
&\vee (\text{married} \neq 1 \wedge \text{omarried} \neq 1 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{omarried} \neq 1 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{omarried} \neq 1 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad \wedge \text{ooopc10} = 1500) \\
&\vee (\text{married} \neq 1 \wedge \text{oage} < 65 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{oage} < 65 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{oage} < 65 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad \wedge \text{ooopc10} = 1500) \\
&\vee (\text{age} < 75 \wedge \text{omarried} = 1 \wedge \text{oage} \geq 65 \wedge \text{oopc10} = 6625) \\
&\vee (\text{age} < 75 \wedge \text{omarried} \neq 1 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{omarried} \neq 1 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{omarried} \neq 1 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad \wedge \text{ooopc10} = 1500) \\
&\vee (\text{age} < 75 \wedge \text{oage} < 65 \wedge \text{oomarried} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{oage} < 65 \wedge \text{oowidow} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{oage} < 65 \wedge \text{oomarried} \neq 1 \wedge \text{oowidow} \neq 1 \\
&\qquad \wedge \text{ooopc10} = 1500) \qquad (8.2\text{-}12)
\end{aligned}
$$

Within the context of the strict linear nature of the system and the corresponding absence of any concurrent actions, and because the frame of $rule_{pc10\text{-}1}$ is limited to the variable 'pc10' and therefore does not interfere with any rule conditions in $rule_{pc10\text{-}1}$, the following implications are asserted:

$$(\circ\text{married} = 1) \supset (\text{married} = 1) \tag{8.2-13a}$$
$$(\circ\circ\text{married} = 1) \supset (\text{married} = 1) \tag{8.2-13b}$$
$$(\circ\text{married} \neq 1) \supset (\text{married} \neq 1) \tag{8.2-13c}$$
$$(\circ\circ\text{married} \neq 1) \supset (\text{married} \neq 1) \tag{8.2-13d}$$
$$(\circ\circ\text{widow} = 1) \supset (\text{widow} = 1) \tag{8.2-13e}$$
$$(\circ\circ\text{widow} \neq 1) \supset (\text{widow} \neq 1) \tag{8.2-13f}$$
$$(\circ\text{age} \geq 65) \supset (\text{age} \geq 65) \tag{8.2-13g}$$
$$(\circ\text{age} < 65) \supset (\text{age} < 65) \tag{8.2-13h}$$

All of these implications have the form $\circ w_0 \supset w_0$ or $\circ\circ w_0 \supset w_0$. They are applied to detemporalize a rule condition that what would otherwise be a simple state formula. Applying (8.2-13a) through (8.2-13h) to $rule_{pc10\text{-}1}$ as described at (8.2-12) using propositional logic (i.e., disjunction elimination) yields:

$$\begin{aligned}
&rule_{pc10\text{-}1} \supset \\
&(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \circ\text{pc10} = 6692) \\
&\vee (\text{married} \neq 1 \wedge \text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\circ\text{pc10} = 6625) \\
&\vee (\text{married} \neq 1 \wedge \text{married} \neq 1 \wedge \text{married} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{married} \neq 1 \wedge \text{widow} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{married} \neq 1 \wedge \text{married} \neq 1 \wedge \text{widow} \neq 1 \\
&\qquad \wedge \circ\circ\circ\text{pc10} = 1500) \\
&\vee (\text{married} \neq 1 \wedge \text{age} < 65 \wedge \text{married} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{age} < 65 \wedge \text{widow} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{age} < 65 \wedge \text{married} \neq 1 \wedge \text{widow} \neq 1 \\
&\qquad \wedge \circ\circ\circ\text{pc10} = 1500) \\
&\vee (\text{age} < 75 \wedge \text{married} = 1 \wedge \text{age} \geq 65 \wedge \circ\circ\text{pc10} = 6625) \\
&\vee (\text{age} < 75 \wedge \text{married} \neq 1 \wedge \text{married} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{married} \neq 1 \wedge \text{widow} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{married} \neq 1 \wedge \text{married} \neq 1 \wedge \text{widow} \neq 1 \\
&\qquad \wedge \circ\circ\circ\text{pc10} = 1500) \\
&\vee (\text{age} < 75 \wedge \text{age} < 65 \wedge \text{married} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{age} < 65 \wedge \text{widow} = 1 \wedge \circ\circ\circ\text{pc10} = 3470) \\
&\vee (\text{age} < 75 \wedge \text{age} < 65 \wedge \text{married} \neq 1 \wedge \text{widow} \neq 1 \\
&\qquad \wedge \circ\circ\circ\text{pc10} = 1500)
\end{aligned} \tag{8.2-14}$$

Applying propositional logic to (8.2-14) to eliminate contradictions and idempotent terms, and then reordering yields:

$$\begin{aligned}
&rule_{pc10\text{-}1} \supset \\
&(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \text{opc10} = 6692) \\
&\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{age} < 75 \wedge \text{oopc10} = 6625) \\
&\vee (\text{married} = 1 \wedge \text{age} < 65 \wedge \text{age} < 75 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{age} < 75 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{widow} = 1 \wedge \text{age} < 65 \wedge \text{age} < 75 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 1500) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 75 \wedge \text{ooopc10} = 1500) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 65 \wedge \text{age} < 75 \\
&\qquad \wedge \text{ooopc10} = 1500) \qquad (8.2\text{-}15)
\end{aligned}$$

Considering the overlapping rule conditions in (8.2-15) with regard to the variable 'age,' the following equivalence is noted:

$$\text{age} < 65 \equiv (\text{age} < 75 \wedge \text{age} < 65) \qquad (8.2\text{-}16)$$

Applying (8.2-16) to $rule_{pc10\text{-}1}$ as described at (8.2-15) and eliminating idempotent terms yields:

$$\begin{aligned}
&rule_{pc10\text{-}1} \supset \\
&(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \text{opc10} = 6692) \\
&\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{age} < 75 \wedge \text{oopc10} = 6625) \\
&\vee (\text{married} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{age} < 75 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{widow} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 1500) \\
&\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 75 \wedge \text{ooopc10} = 1500) \qquad (8.2\text{-}17)
\end{aligned}$$

Although substantial transformation and simplification has been achieved, various redundancies exist. Consider the following three disjunctively connected rules included in $rule_{pc10\text{-}1}$ as described at (8.2-17):

$$\begin{aligned}&(\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500)\\ &\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 1500)\\ &\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 75 \wedge \text{ooopc10} = 1500)\end{aligned} \tag{8.2-18}$$

Given that the rule condition variable 'age' is not included in the first rule, given that the rule conditions are otherwise identical, and given that each rule has the identical rule state, the inclusion of the rule condition variable 'age' in the second and third rule is irrelevant. Referencing the concept of transformational equivalence as previously presented Section 6.5, these three rules can be transformed into identical rules with the application of the appropriate logic. To support such transformations, the following implications are derived under propositional logic:

$$\begin{aligned}&\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 1500 \supset\\ &\quad \text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500\end{aligned} \tag{8.2-19a}$$

$$\begin{aligned}&\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{age} < 75 \wedge \text{ooopc10} = 1500 \supset\\ &\quad \text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500\end{aligned} \tag{8.2-19b}$$

$$\begin{aligned}&\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470 \supset\\ &\quad \text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{ooopc10} = 3470\end{aligned} \tag{8.2-19c}$$

$$\begin{aligned}&\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{age} < 75 \wedge \text{ooopc10} = 3470 \supset\\ &\quad \text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{ooopc10} = 3470\end{aligned} \tag{8.2-19d}$$

Applying (8.2-19a) through (8.2-19d) to $rule_{pc10\text{-}1}$ as described at (8.2-17) using propositional logic (i.e., disjunction elimination) and then eliminating the idempotent terms yields:

$$\begin{aligned}&rule_{pc10\text{-}1} \supset\\ &(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \text{opc10} = 6692)\\ &\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{age} < 75 \wedge \text{oopc10} = 6625)\\ &\vee (\text{married} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470)\\ &\vee (\text{married} \neq 1 \wedge \text{widow} = 1 \wedge \text{ooopc10} = 3470)\\ &\vee (\text{widow} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470)\\ &\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500)\end{aligned} \tag{8.2-20}$$

One final simplifying transformation is possible. Considering the domain of and relation between the rule condition variables 'married' and 'widow,' the following observation is made:

$$(\text{married} \neq 1 \wedge \text{widow} = 1) \supset (\text{widow} = 1) \qquad (8.2\text{-}21)$$

Informally, (8.2-21) describes the fact that a widow cannot be married. To complete this transformation, the following implication, similar to those previously presented at (8.2-19), is derived under propositional logic

$$\begin{aligned}&\text{widow} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470 \supset \\ &\quad \text{widow} = 1 \wedge \text{ooopc10} = 3470\end{aligned} \qquad (8.2\text{-}22)$$

Applying (8.2-21) and (8.2-22) to $rule_{pc10\text{-}1}$ as described at (8.2-20) using propositional logic (i.e., disjunction elimination) and then eliminating the idempotent term yields:

$$\begin{aligned}&rule_{pc10\text{-}1} \supset \\ &(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \text{opc10} = 6692) \\ &\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{age} < 75 \wedge \text{oopc10} = 6625) \\ &\vee (\text{married} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470) \\ &\vee (\text{widow} = 1 \wedge \text{ooopc10} = 3470) \\ &\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500)\end{aligned} \qquad (8.2\text{-}23)$$

To facilitate further analysis and rule representation, the consequent of (8.2-23) is defined as:

$$\begin{aligned}&rule_{pc10\text{-}1'} \triangleq \\ &(\text{married} = 1 \wedge \text{age} \geq 75 \wedge \text{opc10} = 6692) \\ &\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{age} < 75 \wedge \text{oopc10} = 6625) \\ &\vee (\text{married} = 1 \wedge \text{age} < 65 \wedge \text{ooopc10} = 3470) \\ &\vee (\text{widow} = 1 \wedge \text{ooopc10} = 3470) \\ &\vee (\text{married} \neq 1 \wedge \text{widow} \neq 1 \wedge \text{ooopc10} = 1500)\end{aligned} \qquad (8.2\text{-}24)$$

With the introduction of this definition, (8.2-23) can be restated as:

$$rule_{pc10\text{-}1} \supset rule_{pc10\text{-}1'} \qquad (8.2\text{-}25)$$

With (8.2-24) and (8.2-25), the transformation of $rule_{pc10\text{-}1}$ is complete. The original program code for $rule_{pc10\text{-}1}$, consisting of three nested if-then-else statements, has been transformed into five disjunctively connected rules as described by $rule_{pc10\text{-}1'}$. As a disjunctive structure, the component rules (i.e., disjuncts) can be reordered as necessary. As an ITL formula, $rule_{pc10\text{-}1'}$ can be used, as necessary, for additional reasoning about the overall system.

Based on an analysis of the programming structures associated with $rule_{pc10\text{-}2}$, and applying the rule-form description of the if-then and the if-then-else programming structures presented in Section 6.6.1 and the rule-form sequential composition presented in Section 5.6.1, $rule_{pc10\text{-}2}$ is described as:

$$rule_{pc10\text{-}2} \equiv (rule_{pc10\text{-}2a(true)} \wedge \circ rule_{pc10\text{-}2b}) \vee rule_{pc10\text{-}2a(false)} \qquad (8.2\text{-}26)$$

where:

$$\begin{aligned}
&rule_{pc10\text{-}2a(true)} \triangleq ((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ t)\\
&rule_{pc10\text{-}2a(false)} \triangleq (\neg(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \text{empty})\\
&rule_{pc10\text{-}2b} \triangleq \quad (t > 3740 \wedge \circ pc10 = t)\\
&\qquad\qquad\quad \vee (\neg(t > 3740) \wedge \circ pc10 = 3740)\\
&t \triangleq pc10 - (\text{income} - 16800) / 2
\end{aligned}$$

Applying the rule algebra and imposing some limited assumptions of the form $\circ w_0 \supset w_0$ regarding specific rule conditions variables, $rule_{pc10\text{-}2}$ is transformed, as described in Appendix D, such that:

$$\begin{aligned}
&rule_{pc10\text{-}2} \supset\\
&(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t > 3740\\
&\quad \wedge \circ\circ pc10 = t)\\
&\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t \leq 3740\\
&\quad \wedge \circ\circ pc10 = 3740)\\
&\vee (\text{married} \neq 1 \wedge \text{empty})\\
&\vee (\text{age} < 65 \wedge \text{empty})\\
&\vee (\text{income} \leq 16800 \wedge \text{empty}) \qquad (8.2\text{-}27)
\end{aligned}$$

where:

$$t \triangleq pc10 - (\text{income} - 16800) / 2$$

To facilitate further analysis and rule representation, the consequent of (8.2-27) is defined as:

$$
\begin{aligned}
&rule_{pc10\text{-}2'} \triangleq \\
&(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t > 3740 \\
&\quad \wedge \circ\circ \text{pc10} = t) \\
&\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t \leq 3740 \\
&\quad \wedge \circ\circ \text{pc10} = 3740) \\
&\vee (\text{married} \neq 1 \wedge \text{empty}) \\
&\vee (\text{age} < 65 \wedge \text{empty}) \\
&\vee (\text{income} \leq 16800 \wedge \text{empty})
\end{aligned}
\qquad (8.2\text{-}27)
$$

where:

$$t \triangleq \text{pc10} - (\text{income} - 16800) / 2$$

With the introduction of this definition, (8.2-26) can be restated as:

$$rule_{pc10\text{-}2} \supset rule_{pc10\text{-}2'} \qquad (8.2\text{-}28)$$

With (8.2-27) and (8.2-28), the transformation of $rule_{pc10\text{-}2}$ is complete. The original program code for $rule_{pc10\text{-}2}$, consisting of an if-then-else statement nested in an if-then statement, has been transformed into a set of disjunctively connected rules as described by $rule_{pc10\text{-}2'}$. In this set of rules, all conditions associated with each rule state are explicitly stated. As a disjunctive structure, the component rules (i.e., disjuncts) can be reordered as necessary. As an ITL formula, $rule_{pc10\text{-}2'}$ can be used as necessary for additional reasoning. For example, a simple manipulation allows the demonstration that the rule conditions of $rule_{pc10\text{-}2'}$ are of the form $w_0 \wedge w_1$, $w_0 \wedge \neg w_1$, or $\neg w_0$, supporting the observation that all possible conditions are considered under $rule_{pc10\text{-}2'}$.

Returning to $rule_{pc10}$, the rule-based representation of the slice on variable 'pc10,' $rule_{pc10}$ has been defined at (8.2-1) as:

$$rule_{pc10\text{-}1} \, ; \, rule_{pc10\text{-}2} \qquad (8.2\text{-}29)$$

Applying (8.2-25) and (8.2-28) to (8.2-29) with ChopSwapImp2 and ChopSwapImp, respectively, yields:

$rule_{pc10\text{-}1'} ; rule_{pc10\text{-}2'}$ (8.2-30)

With (8.2-30), the slice on variable 'pc10' is described as two chopped state sequences, where each component sequence is described by a set of disjunctively connected general-form rules. Unlike the original program code that includes nested if-then-else statements, these formulas reflect a substantial logical simplification. In each set of rules, the rule conditions associated with each rule state are explicitly stated. For specific values of the rule conditions, the applicable rule state can be easily identified by applying those values and assessing the verity of the conditions within each disjunct. This is particularly significant with regard to identifying the rules that apply to a limited set of conditions. For example, for an unmarried individual (i.e., married ≠ 1), it is immediately apparent that only one disjunct in $rule_{pc10\text{-}1'}$ and only one disjunct in $rule_{pc10\text{-}2'}$ applies. This is in contrast to the code slice, in which the code must be walked, that is, each line of code evaluated and the if-then-else statements followed, to determine what portion of the code applies to the specific condition. Although the relative order of the two component rules, $rule_{pc10\text{-}1'}$ and $rule_{pc10\text{-}2'}$, is fixed, the component disjuncts within each rule can be reordered as needed for presentation purposes. Continuing the previous example of the unmarried individual, the disjuncts of (8.2-24) and (8.2-27) can be reordered to list the disjuncts with the rule condition 'married ≠ 1' first. Finally, and as previously stated regarding the component rules, (8.2-30) is an ITL formula and can be used as needed for additional reasoning, either about the slice itself or as part of an analysis of the entire block of code.

As a second slicing exercise, the WSL tax program was backward sliced on the variable 'personal' using the FermaT Syntactic_Slice tranformation, and the following slice was generated:

```
IF age >= 75
  THEN personal := 5980
  ELSE IF age >= 65
       THEN personal := 5720
       ELSE personal := 4335 FI FI;
IF age >= 65 AND income > 16800
  THEN VAR < t := personal - (income - 16800) / 2 >:
     IF t > 4335
       THEN personal := t
```

```
        ELSE personal := 4335 FI ENDVAR FI;
IF blind = 1
  THEN personal := personal + 1380 FI
```

Based on an inspection and analysis of the programming structures that comprise this slice, this slice on the variable 'personal' can be represented as a sequence of three chopped rules:

$$rule_{personal} \equiv rule_{pers\text{-}1}\ ;\ rule_{pers\text{-}2}\ ;\ rule_{pers\text{-}3} \qquad (8.2\text{-}31)$$

where:

```
rule_pers-1 ≜   IF age >= 75
                  THEN personal := 5980
                  ELSE IF age >= 65
                         THEN personal := 5720
                         ELSE personal := 4335 FI FI;

rule_pers-2 ≜   IF age >= 65 AND income > 16800
                  THEN VAR < t := personal - (income - 16800) / 2 >:
                      IF t > 4335
                        THEN personal := t
                        ELSE personal := 4335 FI ENDVAR FI;

rule_pers-3 ≜   IF blind = 1
                  THEN personal := personal + 1380 FI
```

Applying the rule-form description of the if-then-else programming structure as presented in Section 6.6.1, $rule_{pers\text{-}1}$ is described as:

$$\begin{aligned} & rule_{pers\text{-}1} \equiv \\ & (age \geq 75 \land \circ personal = 5980) \\ & \lor (\neg(age \geq 75) \land \circ((age \geq 65 \land \circ personal = 5720) \\ & \qquad\qquad \lor (\neg(age \geq 65) \land \circ personal = 4335))) \end{aligned} \qquad (8.2\text{-}32)$$

Applying algebraic equivalences regarding the great-than-or-equal and negation operators yields:

$$\begin{aligned}&rule_{pers\text{-}1} \equiv \\ &(\text{age} \geq 75 \wedge \circ\text{personal} = 5980) \\ &\vee (\text{age} < 75 \wedge \circ((\text{age} \geq 65 \wedge \circ\text{personal} = 5720) \\ &\qquad\qquad \vee (\text{age} < 65 \wedge \circ\text{personal} = 4335)))\end{aligned} \qquad (8.2\text{-}33)$$

Applying NextOrDistEqv and then NextAndDistEqv to (8.2-33) yields:

$$\begin{aligned}&rule_{pers\text{-}1} \equiv \\ &(\text{age} \geq 75 \wedge \circ\text{personal} = 5980) \\ &\vee (\text{age} < 75 \wedge ((\circ\text{age} \geq 65 \wedge \circ\circ\text{personal} = 5720) \\ &\qquad\qquad \vee (\circ\text{age} < 65 \wedge \circ\circ\text{personal} = 4335)))\end{aligned} \qquad (8.2\text{-}34)$$

With the application of propositional logic to (8.2-34), $rule_{pers\text{-}1}$ is described as a disjunction of three general form rules:

$$\begin{aligned}&rule_{pers\text{-}1} \equiv \\ &(\text{age} \geq 75 \wedge \circ\text{personal} = 5980) \\ &\vee (\text{age} < 75 \wedge \circ\text{age} \geq 65 \wedge \circ\circ\text{personal} = 5720) \\ &\vee (\text{age} < 75 \wedge \circ\text{age} < 65 \wedge \circ\circ\text{personal} = 4335)\end{aligned} \qquad (8.2\text{-}35)$$

Within the context of the strict linear nature of the system and the corresponding absence of any concurrent actions, and because the frame of $rule_{pers\text{-}1}$ is limited to the variable 'personal' and therefore does not interfere with any rule conditions, the following implications of the form $\circ w_0 \supset w_0$ are asserted:

$$\circ\text{age} \geq 65 \supset \text{age} \geq 65 \qquad (8.2\text{-}36a)$$
$$\circ\text{age} < 65 \supset \text{age} < 65 \qquad (8.2\text{-}36b)$$

Applying these implications to (8.2-35) using propositional logic (i.e., disjunction elimination) yields:

$$\begin{aligned}&rule_{pers\text{-}1} \supset \\ &(\text{age} \geq 75 \wedge \circ\text{personal} = 5980) \\ &\vee (\text{age} < 75 \wedge \text{age} \geq 65 \wedge \circ\circ\text{personal} = 5720) \\ &\vee (\text{age} < 75 \wedge \text{age} < 65 \wedge \circ\circ\text{personal} = 4335)\end{aligned} \qquad (8.2\text{-}37)$$

The following equivalence regarding the variable 'age' has been asserted previously at (8.2-16):

$$\text{age} < 65 \equiv (\text{age} < 75 \wedge \text{age} < 65) \qquad (8.2\text{-}16)$$

Applying this equivalence to (8.2-37) yields:

$$\begin{aligned}&\mathit{rule}_{pers\text{-}1} \supset \\ &(\text{age} \geq 75 \wedge \circ\text{personal} = 5980) \\ &\vee (\text{age} < 75 \wedge \text{age} \geq 65 \wedge \circ\circ\text{personal} = 5720) \\ &\vee (\text{age} < 65 \wedge \circ\circ\text{personal} = 4335)\end{aligned} \qquad (8.2\text{-}38)$$

Based on an analysis of the programming structures associated with $\mathit{rule}_{pers\text{-}2}$, and applying the rule-form description of the if-then and the if-then-else programming structures presented in Section 6.6.1 and the rule-form sequential composition presented in Section 5.6.1, $\mathit{rule}_{pers\text{-}2}$ is described as:

$$\mathit{rule}_{pers\text{-}2} \equiv (\mathit{rule}_{pers\text{-}2a(true)} \wedge \circ\mathit{rule}_{pers\text{-}2b}) \vee \mathit{rule}_{pers\text{-}2a(false)} \qquad (8.2\text{-}39)$$

where:

$$\begin{aligned}&\mathit{rule}_{pers\text{-}2a(true)} \triangleq ((\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ\text{t}) \\ &\mathit{rule}_{pers\text{-}2a(false)} \triangleq (\neg(\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \text{empty}) \\ &\mathit{rule}_{pers\text{-}2b} \triangleq \quad (\text{t} > 4335 \wedge \circ\text{personal} = \text{t}) \\ &\qquad\qquad \vee (\neg(\text{t} > 4335) \wedge \circ\text{personal} = 4335) \\ &\text{t} \triangleq \text{personal} - (\text{income} - 16800) / 2\end{aligned}$$

Using these representations of $\mathit{rule}_{pers\text{-}2a}$ and $\mathit{rule}_{pers\text{-}2b}$, $\mathit{rule}_{pers\text{-}2}$ is transformed, as described in Appendix D, such that:

$$\begin{aligned}&\mathit{rule}_{pers\text{-}2} \supset \\ &(\text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ\text{t} > 4335 \wedge \circ\circ\text{personal} = \text{t}) \\ &\vee (\text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ\text{t} \leq 4335 \wedge \circ\circ\text{personal} = 4335) \\ &\vee (\text{age} < 65 \wedge \text{empty}) \\ &\vee (\text{income} \leq 16800 \wedge \text{empty})\end{aligned} \qquad (8.2\text{-}40)$$

where:

$$\text{t} \triangleq \text{personal} - (\text{income} - 16800) / 2$$

Applying the rule-form description of the if-then programming structure as presented in Section 6.6.1, $rule_{pers\text{-}3}$ is described as:

$$\begin{aligned} rule_{pers\text{-}3} \equiv\ & (\text{blind} = 1 \wedge \circ\text{personal} = \text{personal} + 1380) \\ & \vee (\neg(\text{blind} = 1) \wedge \text{empty}) \end{aligned} \qquad (8.2\text{-}41)$$

Applying algebraic equivalences regarding the equality and negation operators to (8.2-41) yields:

$$\begin{aligned} rule_{pers\text{-}3} \equiv\ & (\text{blind} = 1 \wedge \circ\text{personal} = \text{personal} + 1380) \\ & \vee (\text{blind} \neq 1 \wedge \text{empty}) \end{aligned} \qquad (8.2\text{-}42)$$

With (8.2-42), the transformation of the three rules that compose $rule_{personal}$ is complete. Summarizing, the slice on the variable 'personal' of WSL tax program code is described as a rule system with three sequential rules:

$$rule_{personal} \equiv rule_{pers\text{-}1}\ ;\ rule_{pers\text{-}2}\ ;\ rule_{pers\text{-}3} \qquad (8.2\text{-}43)$$

These three rules are described as rule systems of disjunctively connected general-form rules where:

$$\begin{aligned} & rule_{pers\text{-}1} \supset \\ & (\text{age} \geq 75 \wedge \circ\text{personal} = 5980) \\ & \vee (\text{age} < 75 \wedge \text{age} \geq 65 \wedge \circ\circ\text{personal} = 5720) \\ & \vee (\text{age} < 65 \wedge \circ\circ\text{personal} = 4335) \end{aligned} \qquad (8.2\text{-}44a)$$

$$\begin{aligned} & rule_{pers\text{-}2} \supset \\ & (\text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ\text{t} > 4335 \wedge \circ\circ\text{personal} = \text{t}) \\ & \vee (\text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ\text{t} \leq 4335 \wedge \circ\circ\text{personal} = 4335) \\ & \vee (\text{age} < 65 \wedge \text{empty}) \\ & \vee (\text{income} \leq 16800 \wedge \text{empty}) \end{aligned} \qquad (8.2\text{-}44b)$$

where:

$$\text{t} \triangleq \text{personal} - (\text{income} - 16800) / 2$$

$$\begin{aligned} rule_{pers\text{-}3} \equiv\ & (\text{blind} = 1 \wedge \circ\text{personal} = \text{personal} + 1380) \\ & \vee (\text{blind} \neq 1 \wedge \text{empty}) \end{aligned} \qquad (8.2\text{-}44c)$$

These extracted rules can be used to analyze the original WSL code. In previous evaluations of this code, Ward et al. (2005) and Fox et al. (2000) applied various forms of conditioned slicing to answer the question "What is the personal allowance calculation for a blind widow aged over 68?" These conditions are expressed in terms of the WSL program variables as:

$$\text{blind} = 1 \qquad (8.2\text{-}45a)$$
$$\text{married} = 0 \qquad (8.2\text{-}45b)$$
$$\text{widow} = 1 \qquad (8.2\text{-}45c)$$
$$\text{age} > 68 \qquad (8.2\text{-}45d)$$

In the following analysis, these same conditions are applied to the extracted rules to generate conditioned rules that reflect those specific conditions. These conditioned rules are then compared to the slices generated by others for the same conditions.

Referencing $rule_{pers\text{-}1}$ at (8.2-44a) and applying the specific conditions at (8.2-45) to access the satisfaction or non-satisfaction of the relevant rule conditions yields $rule_{pers\text{-}1\text{-}cond}$:

$$\begin{aligned}&rule_{pers\text{-}1\text{-}cond} \supset\\&(\text{age} \geq 75 \wedge \circ\text{personal} = 5980)\\&\vee (\text{age} < 75 \wedge \mathit{true} \wedge \circ\circ\text{personal} = 5720)\\&\vee (\mathit{false} \wedge \circ\circ\text{personal} = 4335)\end{aligned} \qquad (8.2\text{-}46)$$

Applying propositional logic to (8.2-46) yields:

$$\begin{aligned}&rule_{pers\text{-}1\text{-}cond} \supset\\&(\text{age} \geq 75 \wedge \circ\text{personal} = 5980)\\&\vee (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)\end{aligned} \qquad (8.2\text{-}47)$$

Referencing $rule_{pers\text{-}2}$ at (8.2-44b) and applying the specific conditions at (8.2-45) to access the satisfaction or non-satisfaction of the relevant rule conditions yields $rule_{pers\text{-}2\text{-}cond}$:

$$\begin{aligned}&rule_{pers\text{-}2\text{-}cond} \supset\\&(\mathit{true} \wedge \text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t)\\&\vee (\mathit{true} \wedge \text{income} > 16800 \wedge \circ t \leq 4335 \wedge \circ\circ\text{personal} = 4335)\\&\vee (\mathit{false} \wedge \text{empty})\\&\vee (\text{income} \leq 16800 \wedge \text{empty})\end{aligned} \qquad (8.2\text{-}48)$$

Applying propositional logic to (8.2-48) yields:

$$\begin{aligned}&rule_{pers\text{-}2\text{-}cond} \supset \\ &(\text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t) \\ &\vee (\text{income} > 16800 \wedge \circ t \leq 4335 \ \wedge \circ\circ\text{personal} = 4335) \\ &\vee (\text{income} \leq 16800 \wedge \text{empty})\end{aligned} \qquad (8.2\text{-}49)$$

Referencing $rule_{pers\text{-}3}$ at (8.2-44c) and applying the specific conditions at (8.2-45) to access the satisfaction or non-satisfaction of the relevant rule conditions yields $rule_{pers\text{-}3\text{-}cond}$:

$$\begin{aligned}rule_{pers\text{-}3\text{-}cond} \equiv\ &(\mathit{true} \wedge \circ\text{personal} = \text{personal} + 1380) \\ &\vee (\mathit{false} \wedge \text{empty})\end{aligned} \qquad (8.2\text{-}50)$$

Applying propositional logic to (8.2-50) yields:

$$rule_{pers\text{-}3\text{-}cond} \equiv (\circ\text{personal} = \text{personal} + 1380) \qquad (8.2\text{-}51)$$

Given the previous definition of $rule_{personal}$ at (8.2-43), the conditioned rule $rule_{personal\text{-}cond}$, conditioned based on the specific conditions presented at (8.2-45), is defined as the chopped sequence:

$$rule_{personal\text{-}cond} \equiv rule_{pers\text{-}1\text{-}cond} \, ; rule_{pers\text{-}2\text{-}cond} \, ; rule_{pers\text{-}3\text{-}cond} \qquad (8.2\text{-}52)$$

Using (8.2-52), applying $rule_{pers\text{-}1\text{-}cond}$ as described at (8.2-47) with ChopSwapImp1 and then applying OrChopEqv yields:

$$\begin{aligned}&rule_{personal\text{-}cond} \supset \\ &(\text{age} \geq 75 \wedge \circ\text{personal} = 5980) \, ; rule_{pers\text{-}2\text{-}cond} \, ; rule_{pers\text{-}3\text{-}cond} \\ &\vee (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720) \, ; rule_{pers\text{-}2\text{-}cond} \, ; rule_{pers\text{-}3\text{-}cond}\end{aligned} \qquad (8.2\text{-}53)$$

Using (8.2-53), applying propositional logic (i.e., disjunction elimination) and then applying $rule_{pers\text{-}2\text{-}cond}$ as described at (8.2-49) with ChopSwapImp3 yields:

$$\begin{array}{l}
\mathit{rule}_{\mathit{personal\text{-}cond}} \supset \\
(\text{age} \geq 75 \wedge \bigcirc\text{personal} = 5980) \,; \\
((\text{income} > 16800 \wedge \bigcirc t > 4335 \wedge \circ\bigcirc\text{personal} = t) \\
\quad \vee (\text{income} > 16800 \wedge \bigcirc t \leq 4335 \;\; \wedge \circ\bigcirc\text{personal} = 4335) \\
\quad \vee (\text{income} \leq 16800 \wedge \text{empty})) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}} \\
\vee (\text{age} < 75 \wedge \circ\bigcirc\text{personal} = 5720) \,; \\
((\text{income} > 16800 \wedge \bigcirc t > 4335 \wedge \circ\bigcirc\text{personal} = t) \\
\quad \vee (\text{income} > 16800 \wedge \bigcirc t \leq 4335 \;\; \wedge \circ\bigcirc\text{personal} = 4335) \\
\quad \vee (\text{income} \leq 16800 \wedge \text{empty})) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}}
\end{array} \tag{8.2-54}$$

Applying OrChopEqv to (8.2-54) yields:

$$\begin{array}{l}
\mathit{rule}_{\mathit{personal\text{-}cond}} \supset \\
(\text{age} \geq 75 \wedge \bigcirc\text{personal} = 5980) \,; \\
((\text{income} > 16800 \wedge \bigcirc t > 4335 \wedge \circ\bigcirc\text{personal} = t) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}} \\
\quad \vee (\text{income} > 16800 \wedge \bigcirc t \leq 4335 \;\; \wedge \circ\bigcirc\text{personal} = 4335) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}} \\
\quad \vee (\text{income} \leq 16800 \wedge \text{empty}) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}} \,) \\
\vee (\text{age} < 75 \wedge \circ\bigcirc\text{personal} = 5720) \,; \\
((\text{income} > 16800 \wedge \bigcirc t > 4335 \wedge \circ\bigcirc\text{personal} = t) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}} \\
\quad \vee (\text{income} > 16800 \wedge \bigcirc t \leq 4335 \;\; \wedge \circ\bigcirc\text{personal} = 4335) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}} \\
\quad \vee (\text{income} \leq 16800 \wedge \text{empty}) \,; \mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}} \,)
\end{array} \tag{8.2-55}$$

Applying ChopOrEqv and substituting for $\mathit{rule}_{\mathit{pers\text{-}3\text{-}cond}}$ yields:

$$\begin{array}{l}
\mathit{rule}_{\mathit{personal\text{-}cond}} \supset \\
(\text{age} \geq 75 \wedge \bigcirc\text{personal} = 5980) \,; \\
(\text{income} > 16800 \wedge \bigcirc t > 4335 \wedge \circ\bigcirc\text{personal} = t) \,; \\
(\bigcirc\text{personal} = \text{personal} + 1380) \\
\vee (\text{age} \geq 75 \wedge \bigcirc\text{personal} = 5980) \,; \\
(\text{income} > 16800 \wedge \bigcirc t \leq 4335 \;\; \wedge \circ\bigcirc\text{personal} = 4335) \,; \\
(\bigcirc\text{personal} = \text{personal} + 1380) \\
\vee (\text{age} \geq 75 \wedge \bigcirc\text{personal} = 5980) \,; \\
(\text{income} \leq 16800 \wedge \text{empty}) \,; \\
(\bigcirc\text{personal} = \text{personal} + 1380) \\
\vee (\text{age} < 75 \wedge \circ\bigcirc\text{personal} = 5720) \,; \\
(\text{income} > 16800 \wedge \bigcirc t > 4335 \wedge \circ\bigcirc\text{personal} = t) \,; \\
(\bigcirc\text{personal} = \text{personal} + 1380)
\end{array}$$

$\vee$ (age < 75 $\wedge$ $\circ\circ$personal = 5720) ;
(income > 16800 $\wedge$ $\circ$t $\leq$ 4335 $\wedge$ $\circ\circ$personal = 4335) ;
($\circ$personal = personal + 1380)

$\vee$ (age < 75 $\wedge$ $\circ\circ$personal = 5720) ;
(income $\leq$ 16800 $\wedge$ empty) ;
($\circ$personal = personal + 1380) (8.2-56)

With (8.2-56), $rule_{personal\text{-}cond}$ is described as a disjunction of six alternative sequences, where each sequence is composed of two rules and a final assignment of a value to the variable 'personal.' Given the similar structure of these six sequences, the final value of the variable 'personal' in each sequence is dependant on the satisfaction of the specified (and complementary) rule conditions. This observation is supported by the final transformation of $rule_{personal\text{-}cond}$, presented in Appendix D, where:

$rule_{personal\text{-}cond} \supset$

(age $\geq$ 75 ; (income > 16800 $\wedge$ income < 20090)
$\wedge$ fin(personal = 15760 - income/2))

$\vee$ (age $\geq$ 75 ; (income > 16800 $\wedge$ income $\geq$ 20090)
$\wedge$ fin(personal = 5715))

$\vee$ (age $\geq$ 75 ; income $\leq$ 16800
$\wedge$ fin(personal = 7360))

$\vee$ (age < 75 ; (income > 16800 $\wedge$ income < 19570)
$\wedge$ fin(personal = 15500 - income/2))

$\vee$ (age < 75 ; (income > 16800 $\wedge$ income $\geq$ 19570)
$\wedge$ fin(personal = 5715))

$\vee$ (age < 75 ; income $\leq$ 16800
$\wedge$ fin(personal = 7100)) (8.2-57)

In (8.2-57), the ITL construct fin is used to denote that the specified formula is *true* on the final subinterval (in this case the final state) of the corresponding interval. Whereas the form of the disjuncts of (8.2-57) is a deviation from the general rule-form $f_i \wedge \circ f_j$ as developed and used in this research, some conceptual similarities are noted. Consistent with the temporal-relationship concepts developed in Chapter 4, the formula $f_i \wedge$ fin $f_j$ describes a conjunctive relationship between a set of conditions describable by $f_i$ and some set of properties describable by $f_j$ that hold in the final subinterval considered by the formula $f_i \wedge \text{fin}\, f_j$.

With (8.2-57), the relationships between the variables 'age' and 'income' and the variable 'personal' as specified in $rule_{personal\text{-}cond}$ are described in the six disjuncts. In each disjunct, the rule conditions are noted and the resulting final value of the variable 'personal' is specified. With regard to the use of the ITL operator chop in the expression of the conditions, given that the conditions based on the variables 'age' and 'income' are expressed in terms of state formulas, and given the semantics of chop, both condition formulas can hold for the same state.

For comparison, consider the following "conditioned slice," sliced by Fox et al. (2000) from the tax program code using the previously defined conditions and converted to WSL by Ward et al. (2005):

```
IF age >=75
 THEN personal := 5980
ELSE IF age >= 65
THEN personal := 5720 FI FI;
IF age >= 65 AND income > 16800
 THEN VAR < t := personal - (income-16800)/2 >:
IF t > 4335
 THEN personal := t
 ELSE personal := 4335 FI ENDVAR FI;
IF blind = 1
 THEN personal := personal + 1380 FI
```

Also for comparison, consider the following "semantic slice," sliced by Ward et al. (2005) from the tax program code using the previously defined conditions:

```
IF age < 75 AND income >= 19570
 THEN personal := 5715
ELSIF age < 75 AND income > 16800
 THEN personal := (16800 - income)/2 + 7100
ELSIF age < 75
 THEN personal := 7100
ELSIF income >= 20090
 THEN personal := 5715
ELSIF income > 16800
 THEN personal := (16800 - income)/2 + 7360
 ELSE personal := 7360 FI
```

Ward et al. (2005) describe semantic slices as business rules for a particular situation, in this case, the business rule for the personal allowance for a blind widow aged over 68.

The expression of this conditioned rule $rule_{personal\text{-}cond}$ as (8.2-56) and/or (8.2-57) has at least three distinct advantages over the above conditioned slice or semantic slice. Firstly, the conditioned rule $rule_{personal\text{-}cond}$ is potentially easier to understand. Whereas Ward et al. (2005) argues that the semantic slice is "clearly easier to understand" as compared to the conditioned slice, the conditioned rule is arguably easier to understand than the conditioned slice or semantic slice. This is because all conditions in $rule_{personal\text{-}cond}$ are explicitly listed and associated with each specific outcome. For example, in (8.2-57), the conditions age $< 75$ and income $\leq 16800$ are explicitly associated with the final outcome of personal $= 7100$. In the conditioned slice and semantic slice, the sliced code must be walked to determine the final outcome associated with the specific conditions. Secondly, because (8.2-56) and (8.2-57) are disjunctions, the six component structures can be presented in any order that is necessary for optimum rule presentation. Conversely, because the conditioned slice and semantic slice are expressed in a program language, the order of the program code and corresponding elements cannot be changed. Finally, as logical formulas, (8.2-56) and/or (8.2-57) can be used directly in further logical reasoning about the target system. Conversely, because the conditioned slice and semantic slice are expressed in a program language, neither support any further reasoning without substantial code-based transformations. Because of these three reasons, the advantages of this general-form rule approach to manipulating program slices are demonstrated, as compared with program code representation of slices.

In this section, a block of WSL program code is sliced and the rules extracted from the program slices. The rule algebra presented in this research is then used to analyze these extracted rules. In the first code slicing and rule extraction exercise, the rule algebra is applied to simply and clarify the extracted rule. Unlike the original program code that includes multiple if-then-else statements that must be traced to determine the specific conditions associated with a given outcome, with the transformed rules, the rule conditions associated with each rule state are explicitly identified and bound to that rule state. Therefore, these transformed rules reflect a substantial simplification. In the second code slicing and rule extraction exercise, the rules extracted from the program slices are conditioned, and these conditioned rules are compared to conditioned slices and semantic slices on the same variable. For program understanding and analysis, these conditioned rules are superior to the conditioned slices or semantic slices because they are more easily understood, can be more easily manipulated for presentation, and can be used directly in further reasoning about the slice or about the source program. As demonstrated in this section, the rule algebra presented in this research is a powerful and complementary addition to slicing for use in program understanding and analysis.

# Chapter 9

# Applying the Rule Algebra to Specify New Rules

In this chapter, the formal rule extraction framework of Chapter 3, the formal temporal rule model of Chapter 4, and the rule algebra of Chapters 5 and 6 are applied to the forward engineering of a rule-based system. This forward-engineering application of the rule model and rule algebra is presented to demonstrate the wide-ranging applicability of the concepts developed in this research.

## 9.1 Refining an Existing Rule with New Rules

Consider $rule_{7.2\text{-}c}$ (repeated below), extracted in Section 7.2 from the automated teller machine specification:

$$\begin{aligned} rule_{7.2\text{-}c} \triangleq\ & (card\_disabled \wedge \circ\text{take\_disabled\_card}) \\ & \vee (\neg card\_disabled \wedge \circ\text{get\_pin} ; rule_{7.2\text{-}d}) \end{aligned} \qquad (7.2\text{-}3)$$

This rule includes the state sequence get_pin. In this section, get_pin is refined using the general-form rule model presented in Chapter 4 and the rule algebra presented in Chapters 5 and 6.

For this analysis, the refinement relation $\sqsubseteq$ is defined as:

$$f_0 \sqsubseteq f_1 \triangleq f_1 \supset f_0 \qquad (9.1\text{-}1)$$

The refinement calculus was first described by Back (1988). Refinement rules expressed as ITL formulas are presented in Cau and Zedan (2000). For this analysis, refinement is achieved by instantiating $f_1$ in (9.1-1) as a sequence of component state sequences (e.g., $f_{1a} ; f_{1b}$), including state sequences described in terms of general-form rules (e.g., $f_{1a} ; (f_{2a} \wedge \circ f_{2b})$). As previously presented in Section 5.7, two forms of sequential composition are available under this rule algebra – using the general rule form and using the ITL operator chop. With regard to the target sequence get_pin, additional details are added by defining equivalent state sequences that split get_pin into component sequences, thereby adding new details and refining get_pin.

Based on an inspection of (7.2-3), the state sequence get_pin is chopped to the state sequence described by $rule_{7.2\text{-}d}$. Based on an inspection of $rule_{7.2\text{-}d}$ and subsequent rules $rule_{7.2\text{-}e}$ and $rule_{7.2\text{-}f}$, previously presented at (7.2-4), (7.2-5), and (7.2-6), respectively, get_pin represents the behavior in which a valid PIN either is or is not obtained within an maximum number of attempts. Within this context, the following informal specification for get_pin is used as the basis for the refinement of get_pin:

"Each ATM customer must enter a valid PIN within a limited number of tries"

Implicit in informal specification is the requirement that the PIN entry process must be initialized with each new customer. Therefore, state sequence get_pin can be defined as two sequentially-composed state sequences:

$$\text{get\_pin} \triangleq \text{init\_pin\_entry} \; ; \; rule_{\text{pin\_entry}} \qquad (9.1\text{-}2)$$

From the definition presented in (9.1-2):

$$\text{init\_pin\_entry} \; ; \; rule_{\text{pin\_entry}} \supset \text{get\_pin} \qquad (9.1\text{-}3)$$

Considering (9.1-3) and referencing the refinement relation $\sqsubset$ as defined in (9.1-1):

$$\text{get\_pin} \sqsubset \text{init\_pin\_entry} \; ; \; rule_{\text{pin\_entry}} \qquad (9.1\text{-}4)$$

Subsequent refinements of the state sequences composing get_pin are implemented in a similar manner.

In (9.1-2), init_pin_entry is the state sequence that results from resetting and initializing the various state variables necessary to accommodate a new customer. Although unspecified at this time, these various state variables include the various flags and counters used in subsequent rules that define the state sequence get_pin. Whereas init_pin_entry must eventually be refined prior to system implementation, this analysis will focus only on $rule_{\text{pin\_entry}}$.

In (9.1-2), the state sequence described by $rule_{\text{pin\_entry}}$ includes the behaviors specifically associated with the PIN entry and validation processes. Within the context of the informal specification for get_pin, as presented above, several distinct elements are required. Firstly, a valid PIN must be entered. Secondly, the customer has only a limited number of attempts to correctly enter a valid pin. Finally, given that there can be multiple (although limited) attempts to enter a valid PIN, a repetitive or looping construct is needed to express the underlying requirement of this specification. The first two elements are incorporated into the rule condition and the third element is used to define the rule form. Letting the rule condition *attempt_limit* be a state formula that is true in a state where the allowable number of entry tries has been exceeded and letting the rule condition *valid_pin* be a state formula that is true in a state where the entered PIN has been validated, $rule_{\text{pin\_entry}}$ is described as a recursive general-form rule as:

$$\begin{aligned} rule_{\text{pin\_entry}} \triangleq ((( & \neg attempt\_limit \land \neg valid\_pin) \\ & \land \circ\text{process\_pin}) ; rule_{\text{pin\_entry}})) \\ & \lor (\neg(\neg attempt\_limit \land \neg valid\_pin) \land \text{empty}) \end{aligned} \tag{9.1-5}$$

Applying propositional logic, (9.1-5) is expressed in an equivalent form as:

$$\begin{aligned} rule_{\text{pin\_entry}} \triangleq ((( & \neg attempt\_limit \land \neg valid\_pin) \\ & \land \circ\text{process\_pin}) ; rule_{\text{pin\_entry}})) \\ & \lor (valid\_pin \land \text{empty}) \\ & \lor (attempt\_limit \land \text{empty}) \end{aligned} \tag{9.1-6}$$

As a recursive rule, the state sequence defined by $rule_{\text{pin\_entry}}$ will end when either of the rule conditions *valid_pin* or *attempt_limit* is satisfied, thereby ending the recursion. Based on the above analysis and interpretation of the informal specification for get_pin, the refinement of get_pin to init_pin_entry ; $rule_{\text{pin\_entry}}$ is consistent with the informal specification for get_pin.

The definition of $rule_{\text{pin\_entry}}$ is a recursive rule that includes the state sequence process_pin. Within the context of the informal specification of get_pin, for this analysis, process_pin is defined as a sequence of state sequences such that:

$$\text{process\_pin} \triangleq \text{display\_pin\_screen} ; rule_{\text{read\_key\_pad}} ; rule_{\text{validate\_pin}} \tag{9.1-7}$$

With this partitioning of process_pin into three separate state sequences, each state sequence can be refined independently. As display_pin_screen is relatively straightforward, only two of the three separate state sequences in process_pin will be refined – $rule_{\text{read\_key\_pad}}$ and $rule_{\text{validate\_pin}}$.

The state sequence $rule_{\text{read\_key\_pad}}$ is a user-directed event. As a event-driven sequence, PIN entry is terminated with a specific key from the keypad – typically the enter key. Therefore, the rule defining the state sequence $rule_{\text{read\_key\_pad}}$ must incorporate this event-driven element. Letting the rule condition *enter_key* be a state formula that is true in a state where the enter key has been pressed, $rule_{\text{read\_key\_pad}}$ is described as a recursive general-form rule as:

$$\begin{aligned} rule_{\text{read\_key\_pad}} \triangleq\ & (\neg enter\_key \wedge \circ\text{key\_buffer})\ ;\ rule_{\text{read\_key\_pad}} \\ & \vee (enter\_key \wedge \circ\text{increment\_attempt}) \end{aligned} \qquad (9.1\text{-}8)$$

In this form, $rule_{\text{read\_key\_pad}}$ differs from previous recursive rules (i.e., the rule form of the while structure as previously discussed in Section 6.6.2) in that defined state sequences are associated with both the satisfaction and non-satisfaction of the rule conditions. In $rule_{\text{read\_key\_pad}}$, the state sequences key_buffer and increment_attempt must be refined (at some future time) to describe, respectively, how the keypad key entries are processed and how a counter is incremented with each PIN that is entered (where this counter can be used to assess the satisfaction of the rule condition *attempt_limit* in $rule_{\text{pin\_entry}}$).

As specified in (9.1-7), the state sequence described by $rule_{\text{read\_key\_pad}}$ is followed by $rule_{\text{validate\_pin}}$. The state sequence described by $rule_{\text{validate\_pin}}$ must consider at least two business rules. Firstly, the PIN must be the proper length – typically four digits, although this may vary based on the specific institution. Secondly, and only after a PIN of proper length is entered, the user-entered PIN must match the PIN on file with the institution for that card/account. Therefore, the general-form rule(s) defining the state sequence $rule_{\text{validate\_pin}}$ must incorporate these two business rules. That the PIN length can be assessed locally and the PIN must be matched at centralized location supports the decision that these activities are best described by two rules.

Letting the rule condition *pin_length* be a state formula that is true in a state where the PIN of proper length has been entered, $rule_{\text{validate\_pin}}$ is described as:

$$rule_{\text{validate\_pin}} \triangleq (pin\_length \wedge \circ rule_{\text{compare\_pin}}) \\ \vee (\neg pin\_length \wedge \text{empty}) \qquad (9.1\text{-}9)$$

In (9.1-9), the state sequence $rule_{\text{compare\_pin}}$ describes the behaviors resulting from the comparing of the user-entered PIN with the PIN on file with the institution for that card/account. Letting the rule condition *pin_match* be a state formula that is true in a state where the user-entered PIN matches the PIN on file, $rule_{\text{compare\_pin}}$ is described as:

$$rule_{\text{compare\_pin}} \triangleq (pin\_match \wedge \circ \text{pin\_valid}) \\ \vee (\neg pin\_length \wedge \text{empty}) \qquad (9.1\text{-}10)$$

Whereas not refined in this analysis, the state sequence pin_valid must satisfy the rule condition valid_pin in the rule $rule_{\text{pin\_entry}}$ at (9.1-6). With regard to the general refinement strategy, $rule_{\text{validate\_pin}}$ and $rule_{\text{compare\_pin}}$ reflect sequential association of two state sequences based on the general rule form, as previously presented in Section 5.6.1. This is in contrast to process_pin at (9.1-7), where rules are sequentially composed using the chop operator, as previously presented in Section 5.6.2.

In summary, the following rules and rule structures have been developed to refine the state sequence get_pin:

$$\text{get\_pin} \triangleq \text{init\_pin\_entry} \; ; \; rule_{\text{pin\_entry}} \qquad (9.1\text{-}2)$$

$$rule_{\text{pin\_entry}} \triangleq (((\neg attempt\_limit \wedge \neg valid\_pin) \\ \wedge \circ \text{process\_pin}) \; ; \; rule_{\text{pin\_entry}})) \\ \vee (valid\_pin \wedge \text{empty}) \\ \vee (attempt\_limit \wedge \text{empty}) \qquad (9.1\text{-}6)$$

$$\text{process\_pin} \triangleq \text{display\_pin\_screen} \; ; \; rule_{\text{read\_key\_pad}} \; ; \; rule_{\text{validate\_pin}} \qquad (9.1\text{-}7)$$

$$rule_{\text{read\_key\_pad}} \triangleq (\neg enter\_key \wedge \circ \text{key\_buffer}) \; ; \; rule_{\text{read\_key\_pad}} \\ \vee (enter\_key \wedge \text{empty}) \qquad (9.1\text{-}8)$$

$$rule_{\text{validate\_pin}} \triangleq (pin\_length \wedge \circ rule_{\text{compare\_pin}}) \\ \vee (\neg pin\_length \wedge \text{empty}) \qquad (9.1\text{-}9)$$

$$rule_{\text{compare\_pin}} \triangleq (pin\_match \wedge \circ\text{pin\_valid}) \vee (\neg pin\_match \wedge \text{empty}) \qquad (9.1\text{-}10)$$

Using the statechart concepts described in Section 7.3, a statechart representing these is presented in Figure 9.1-1.

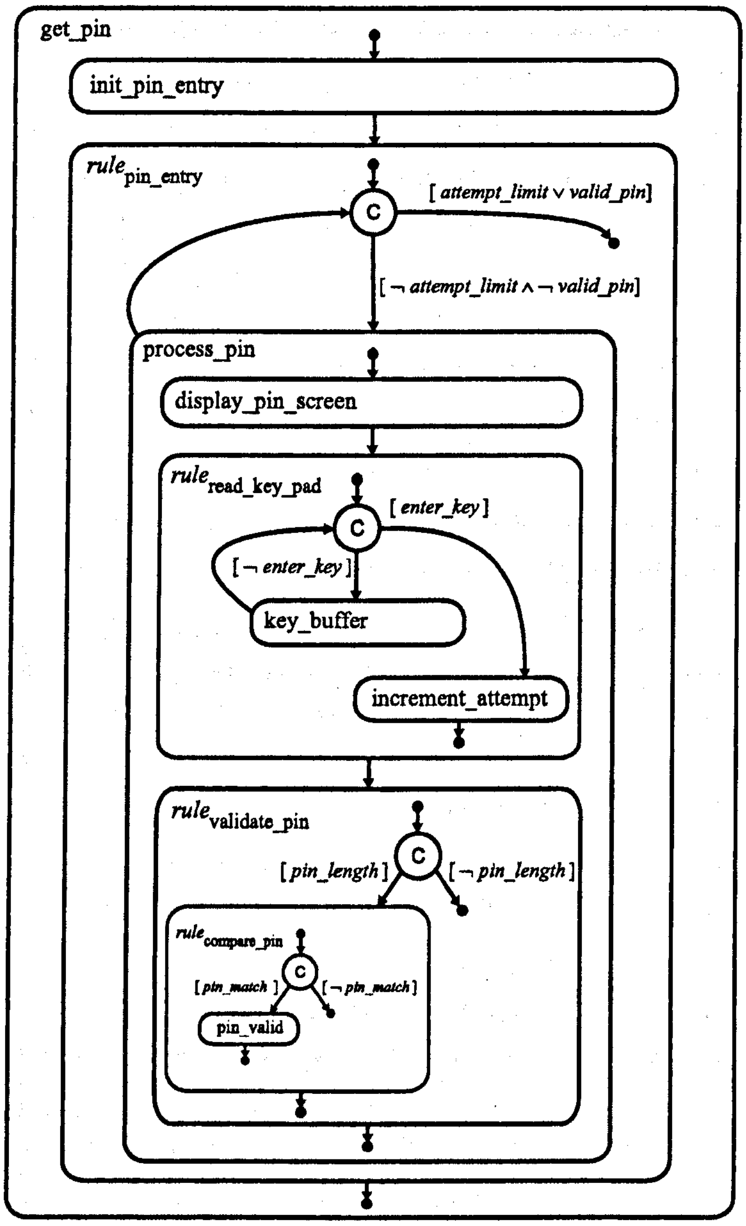


Figure 9.1-1: Statechart for Refined State Sequence get_pin

## 9.2 Analyzing the New Rules Using the Rule Algebra

To assess these rules, including the rule conditions and the associated rule states, the rule algebra presented in this research is applied to the analysis of $rule_{\text{pin\_entry}}$ (including process_pin), $rule_{\text{read\_key\_pad}}$, $rule_{\text{validate\_pin}}$, and $rule_{\text{compare\_pin}}$. Because the state sequence init_pin_entry is focused only on the initialization of various program flags and counters, and at this refinement level contains no rules, init_pin_entry is not considered in this analysis.

To implement these rule transformations, an additional lemma is introduced – TwoChopRulesImp4. TwoChopRulesImp4 is a continuation TwoChopRulesImp series previously presented in Sections 7.2 and 8.1, and is used to separate and collect the rule conditions and rule states of the two chopped rules and transform them into a single general form rule.

LEMMA: TwoChopRulesImp4

$\vdash f_0 ; (f_1 \wedge f_2) ; (f_3 \wedge f_4)$ implies $\vdash f_0 ; ((f_1 ; f_3) \wedge (f_2 ; f_4))$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 ; (f_1 \wedge f_2) ; (f_3 \wedge f_4)$ | premise |
| 2 | $(f_1 \wedge f_2) ; (f_3 \wedge f_4)$ | CP assumption |
| 3 | $(f_1 ; f_3) \wedge (f_2 ; f_4)$ | 2, TwoChopRulesImp |
| 4 | $(f_1 \wedge f_2) ; (f_3 \wedge f_4) \supset (f_1 ; f_3) \wedge (f_2 ; f_4)$ | 2-3, $\supset$ introduction |
| 5 | $f_0 ; (f_1 \wedge f_2) ; (f_3 \wedge f_4) \supset f_0 ; ((f_1 ; f_3) \wedge (f_2 ; f_4))$ | 4, ITL (RightChopImpChop) |
| 6 | $f_0 ; ((f_1 ; f_3) \wedge (f_2 ; f_4))$ | 1, 5, MP |

In this analysis, these four rules ($rule_{\text{pin\_entry}}$, $rule_{\text{read\_key\_pad}}$, $rule_{\text{validate\_pin}}$, and $rule_{\text{compare\_pin}}$) are used as premises. The general transformation strategy for this complete analysis of all state sequences or behaviors associated with $rule_{\text{pin\_entry}}$ (and $rule_{\text{read\_key\_pad}}$, $rule_{\text{validate\_pin}}$, and $rule_{\text{compare\_pin}}$ by inclusion) is identical to that used in the rule transformation of Section 7.2 – cleave each contributory rule into the component rule condition and rule state, and then add those components, in order, into the aggregate descriptions of rule conditions and corresponding system behaviors. This disassembly and subsequent reassembly is performed using ITL and the rule algebra presented in this research. Because this is an assessment of all possible behaviors associated with an entire set of rules, these alternative behaviors are expressed disjunctively. The target rules are processed in reverse order, that is, from the deepest rule upwards. In this way, behaviors are transformed systematically, and each subsequent behavior associated with a specific rule rests on the behavior defined by that rule's component rules. This transformation is presented in Appendix E.

The final result of this transformation, representing the various behaviors of $rule_{\text{pin\_entry}}$, is presented below:

$$\begin{aligned}
&(((\neg attempt\_limit \land \neg valid\_pin) \,;\, \neg enter\_key) \\
&\;\land (\circ\text{display\_pin\_screen} \,;\circ\text{key\_buffer} \,; \\
&\quad rule_{\text{read\_key\_pad}} \,;\, rule_{\text{pin\_entry}})) && (9.2\text{-}1a)
\end{aligned}$$

$$\begin{aligned}
&\lor (((\neg attempt\_limit \land \neg valid\_pin) \,;\, enter\_key \,; \\
&\quad (pin\_length \land \circ pin\_match)) \\
&\;\land (\circ\text{display\_pin\_screen} \,;\, \circ\text{increment\_attempt} \,; \\
&\quad \circ\circ\text{pin\_valid} \,;\, rule_{\text{pin\_entry}})) && (9.2\text{-}1b)
\end{aligned}$$

$$\begin{aligned}
&\lor (((\neg attempt\_limit \land \neg valid\_pin) \,;\, enter\_key \,; \\
&\quad (pin\_length \land \circ\neg pin\_match)) \\
&\;\land \circ\text{display\_pin\_screen} \,;\, \circ\text{increment\_attempt} \,; \\
&\quad \circ\circ\text{display\_invalid\_screen} \,;\, rule_{\text{pin\_entry}})) && (9.2\text{-}1c)
\end{aligned}$$

$$\begin{aligned}
&\lor (((\neg attempt\_limit \land \neg valid\_pin) \,;\, enter\_key \,;\, \neg pin\_length) \\
&\;\land (\circ\text{display\_pin\_screen} \,;\, \circ\text{increment\_attempt} \,; \\
&\quad \circ\text{display\_invalid\_screen} \,;\, rule_{\text{pin\_entry}})) && (9.2\text{-}1d)
\end{aligned}$$

$$\lor\, (valid\_pin \land \text{empty}) \qquad (9.2\text{-}1e)$$

$$\lor\, (attempt\_limit \land \text{empty})) \qquad (9.2\text{-}1f)$$

Although (9.2-1) is a single disjunctive statement, each component disjunct is numbered individually to facilitate discussion.

Using (9.2-1) and knowing the verity of the five rule conditions *attempt_limit, valid_pin, enter_key, pin_length,* and *pin_match* for a specific instance, the system behavior under $rule_{\text{pin\_entry}}$ for that instance can be determined. For example and as depicted in (9.2-1f), if *attempt_limit* is satisfied, then empty holds and the state sequence described by $rule_{\text{pin\_entry}}$ ends. Referencing (9.1-2), when $rule_{\text{pin\_entry}}$ ends, get_pin ends. Referencing $rule_{7.2\text{-}c}$ in Section 7.2 at (7.2-3), after get_pin, the system behavior is described by $rule_{7.2\text{-}d}$. A similar behavior is depicted in (9.2-1e) associated with the satisfaction of the rule condition *valid_pin.* Because disjuncts (9.2-1e) and (9.2-1f) are the only disjuncts that do not include a recursive reference to $rule_{\text{pin\_entry}}$, the satisfaction of either *attempt_limit* or *valid_pin* is the only way that $rule_{\text{pin\_entry}}$ and get_pin ends. As depicted in (9.2-1c) and (9.2-1d), if $\neg$*attempt_limit* and $\neg$*valid_pin* are satisfied, if *enter_key* is satisfied, and either $\neg$*pin_length* is satisfied or *pin_length* and $\neg$*pin_match* are satisfied, the resulting state sequence is described by display_invalid_screen, informing the user that an invalid PIN was entered. With the satisfaction of the rule conditions specified in (9.2-1b), the state sequence pin_valid results so that with the next recursive execution of $rule_{\text{pin\_entry}}$, the rule condition *valid_pin* will be satisfied. Finally, in (9.2-1a), if the rule condition $\neg$*enter_key* is satisfied, signaling that the enter key has not been pressed at the key pad, system behavior continues to be defined by the recursive reference to $rule_{\text{read\_key\_pad}}$, thereby accepting additional key pad input.

As demonstrated above, this transformation allows an alternative form for checking the formation of the original rules. In addition to the assessment of the rule verities and the associated final behaviors, this transformation allows the order of the rule conditions to be assessed. With each set of rule conditions, the associated order(s) of the intermediate behaviors leading to a specific final behavior can be assessed. Finally, because this transformation is a disjunctively connected sets of general-form rules, this transformation can be used for additional reasoning about the overall system of which $rule_{\text{read\_key\_pad}}$ is a part.

In this section, the rule model and rule algebra of this research are applied to the forward engineering of rules to refine a specification. Additional details regarding system behavior are achieved by dividing a previously specified state sequence into a composition of two or more state sequences. These new and more detailed state sequences can be expressed as a single state sequence (e.g., init_pin_entry) or they can be described as a system of two or more disjunctively connected rules, thereby describing two or more possible state sequences. For example, the state_sequence get_pin is initially refined into two state sequences, init_pin_entry and $rule_{\text{pin\_entry}}$, where init_pin_entry defines only one state sequence (subject to future refinement) and $rule_{\text{pin\_entry}}$ defines alternative multiple state sequences depending on the satisfaction of the associated rule conditions. This refinement process is repeated for selected state sequences until sufficient detail is introduced. Then, the resulting rules can be transformed using the rule algebra presented in the research. With these transformations, the rules can be assessed with regard to the rule conditions and the associated rule states. With this example, the rule model and rule algebra presented in this research are demonstrated to be a viable and useful basis for the orderly and stepwise development and refinement of rules and rule-based descriptions of specific system behaviors.

# Chapter 10

# Observations regarding the Rule Algebra and its Application

In this chapter, observations are presented regarding the basis and development of the rule algebra and regarding the application of the rule algebra to both the analysis and the development of rule-based models, specifications, and code. A brief discussion of this rule algebra and its application relative to rule analysis, relative to literature previously reviewed in Chapter 2, is presented.

## 10.1 On the Rule Algebra

In Chapters 5 and 6, a rule algebra is developed using the temporal rule model presented in Chapter 4. Given the underlying principle behind that rule model, that a rule is a conjunctive relationship between a state sequence and a future state sequence describable by the general-form rule $f_i \wedge \circ f_j$, this rule algebra is incrementally developed in Chapter 5 by considering fundamental systems and the corresponding relationships between the state sequences that compose those systems. Using the concept of a rule system – a collection of two or more related rules – more complicated state sequences are described. One extremely important rule system used extensively in this rule algebra is the total rule – a pair of disjunctively associated rules incorporating complementary rule conditions. With this inclusion of complementary rule conditions, it is assured that all the state sequences will satisfy one or the other of the rule conditions included in the total rule.

In Chapter 6, significant attention is given in this rule algebra to composing rules and rule systems in order to describe larger and more complex state sequences. Compositional paradigms that are demonstrated include: sequential composition using both the general rule form itself and the ITL operator chop; nesting; recursion; deterministic and non-deterministic guarded composition; and disjoint parallel composition. Using these compositional paradigms, rule-based representations of typical legacy code structures – the if-then-else structure, the while structure, and the indexed for-loop – are developed.

Although not easily quantified, a critical element of this rule algebra is the fundamental simplicity with which a diverse spectrum of rules are defined and manipulated. Forty-three lemmas are developed in this research as part of this rule algebra to describe allowable and desirable transformations of various rules and rule systems. With the expressiveness of ITL, the proofs necessary to support these lemmas are quite direct. In the development of this rule algebra, no problems or issues were encountered in the description of increasingly complicated state sequences or with the systematic development of the related lemmas. Whereas the rule algebra developed herein provides sufficient means to achieve the immediate goals of this research, the lemmas presented in this research form a core for the development of additional transformations as needed. Because this rule algebra is built on ITL, the richness of ITL is available, if and as needed, for additional development and future refinement of this rule algebra.

## 10.2 On the Application of the Rule Algebra

In Chapters 7, 8, and 9, the rule model of Chapter 4 and the rule algebra of Chapters 5 and 6 are applied to the extraction of rules from existing systems, to the analysis of those rules, and to the development of new rules. Rules are extracted from a variety of existing systems: a finite state machine, a detailed formal specification, a block of legacy Pascal code, and slices from a WSL program. The flexibility and adaptability of this rule algebra are demonstrated both with the diversity of systems from which rules are extracted and with the transformations and analyses that are achieved using the extracted rules. With these demonstrations, as least eight significant benefits are demonstrated regarding the value and applicability of this rule model and rule algebra.

Firstly, the rule algebra developed in this research is sufficiently expressive to allow the analysis of a range of existing models, specifications, and programs. No model, specification, and program structures are encountered that cannot be adequately represented with the rule model, rule algebra, and ITL. The general-form rule defined in Chapter 4, the fundamental structures explored in Chapter 5, and the compositional models presented in Chapter 6 are sufficient, either directly or indirectly, to describe all elements of the various systems considered in this research. By linking the rule algebra concepts presented in Chapters 5 and 6, either by composing rules sequentially or by nesting rules within rules, complex logical and programming structures can be addressed, as demonstrated with the diversity of systems analyzed.

Secondly, and closely related to the previously discussed expressiveness, the rule algebra is adaptable. Given the underlying formations of the rule algebra and the depth of ITL, additional lemmas can be developed to support and expand the rule algebra as needed, as demonstrated with the additional lemmas introduced in Sections 7.2 and 8.1 to achieve rule transformations. Similarly, the rule algebra is not overly restrictive with regard to new or allied concepts. In Section 7.2, the ITL sometimes operator $\Diamond$ is used to allow an alternative expression of the temporal ordering of the rule conditions while still maintaining the underlying general-rule form. In Section 8.2, the ITL fin construct is used to describe the properties of the final state in the state sequence described by the rule. Although the fin form is a deviation from the general rule-form used throughout this research, the conceptual similarities are noted.

Thirdly, the rule algebra supports different levels of analysis. As demonstrated with each of the rule analysis cases presented in Chapters 7 and 8, the application of the rule algebra can be tailored as needed to meet overall expectations and objectives of a specific rule extraction process. As demonstrated with each case considered herein, the rule algebra can be applied incrementally, and the resulting rule transformations can be used for additional reasoning about other rules and the overall system. This incremental approach is extremely important in the early phases of a legacy-system analysis when system-specific knowledge may be limited, and specific expectations and objectives may be vague and uncertain.

Fourthly, statecharts are used to represent legacy-code programming structures, and their use is consistent and compatible with the rule model and rule algebra presented in this research. Together, statecharts and this rule algebra provide a robust tool for legacy code analysis. Correspondences between the statechart elements and the rule elements are presented in Section 7.3 such that statecharts can be developed that are equivalent to extracted general-form rules. Therefore, these equivalent presentations of the same program structures differ not in content, but only in how they can be used in future analysis and understanding. The statechart approach allows a visual presentation that is readily understandable by a wider audience, and the formulaic approach of representing the extracted rules as ITL formulas is readily adaptable to computer analysis techniques. Coupled with the rule algebra, statecharts represent a robust approach to managing the 'state explosion' problem that may result in the analysis and extraction of rules in real-world legacy systems, as identified in Chapter 3.

Fifthly, this rule algebra is applicable within the context of other program analysis techniques such as those described in Chapter 2. In Section 7.2, with the transformation of the extracted rules, the specific sequence of rule conditions and associated rule states leading to a defined goal is identified. Borrowing from the nomenclature of other program understanding techniques, the resulting sequence is described as a state-sequence slice. In Section 8.1, the rule algebra is used as the basis for developing and populating a database usable for legacy code analysis. In Section 8.2, the rule algebra is applied in concert with traditional program slicing.

Sixthly, simplification is achieved with the transformation and representation of these systems using the rule algebra. At (7.1-11), a three-state, five-transition finite state machine is described with two general-form rules. At (7.2-27), a recursive while-form specification that includes four nested if-then-else specifications is transformed into a disjunction of six general-form rules. In each of these general-form rules, the rule conditions that must be met are identified and the corresponding system behavior is clearly presented as an ordered sequence of state sequences, including the recursive behavior of the original specification. At (8.2-25), three nested if-then-else statements are transformed into a disjunction of five easily understood rules, where all rule conditions associated with each rule state are explicitly identified.

Seventhly, the application of the rule algebra for the analysis of the rules from a given model, specification, or program allows the direct assessment of the behavior of that system with regard to specific conditions. In Section 7.1, the rules extracted from a finite state machine were used to model the state sequence response of that machine to a specific input. In Section 8.1, the extracted rules and the associated database were used to assess the specific rule conditions necessary for specific I/O writing operations in the original legacy code. In Section 8.2, conditioned rules – transformed and reduced rules reflecting the imposition of specific rule condition values – are demonstrated to be superior to conditioned slices or semantic slices with respect to program behavior, as well as rule presentation and further reasoning activities. With these assessments of specific system behaviors, substantial knowledge of the original system is obtained.

Finally, this rule algebra is not limited to rule extraction, but also can be applied to the forward engineering of new rules to describe new specifications/programs and their behaviors. Such forward engineering of rules is demonstrated in Chapter 9. The forward engineering of rules to describe a simple hardware system is presented in Appendix F. A significant advantage of using this rule algebra as the basis for forward engineering new rules is that these newly created rules can be then analyzed, reasoned about, and/or tested with the rule algebra, similar to the processes used to assess legacy code, to assure that these new rules meet all expectations associated with the new system. Whereas note explicitly explored in this research, the rule model presented in Chapter 4 is consistent with the inclusion and use of pre-condition and post-condition assertions in specification and program code development, including other language and programming paradigms that directly support such assertions.

Based on these eight observations, and as supported by the specific analyses presented in this research, the rule model and rule algebra developed in the research form the robust and adaptable basis for the extraction of rules from a spectrum of existing or legacy systems, the forward engineering of new systems, the formal transformation and analysis of rules, and specification/program comprehension.

## 10.3 Comparison with Existing Models and Approaches

This rule model and rule algebra differ substantially from rule models and rule analysis techniques presented in the literature as reviewed in Chapter 2. Unlike the informal, descriptive models or definitions of rules presented by Ulrich (1999), Perkins (2000), Odell (1995), Ross (1997), Sneed and Erdos (1996), and others, this rule model is formally defined under ITL and therefore incorporates ITL's well-defined semantics. Unlike the formal rule models presented by Alagar and Periyasamy (2001) and Ungureanu and Minsky (2000) that require identification or specification of an agent, this rule model and the application of the associated rule algebra require no such agent identification or specification.

Other research models or approaches have some specified limitations with regard to application. For example, in Fu et al. (2001), four types of constraints are supported by the Business Rule Language. With the expressiveness of ITL, no arbitrary limits are place on the number or type of constraints that can be expressed with the general form rule model developed as part of this research.

Numerous researchers attempt to partition rules into different and distinct categories and suggest these categories may influence how these various rules are modeled, analyzed, or represented. Theodoulidis et al. (1992) identified three categories of rules: constraint, derivation, and event-action. Shao and Pound (1999) classified business rules into three groups – structural rules, behavioral rules, and constraint rules. Leite and Leonardi (1998) propose classifying business rules as either functional or non-functional. Odell (1995) identified three types of constraint rules and two types of derivation rules. Unlike these approaches, the application of this rule model and associated rule algebra require no arbitrary partitioning or classification of the rules in the subject domain. Under the state-based model incorporated in this rule, any rule that is or can be implemented in a state-based architecture can be captured using this rule model. For example, structural changes can be modeled with the general form rule model of this research by adding or removing variables from the state space. Constraint rules can be modeled with the general form rule model of this research by adding additional conditions to the rule condition. Behavioral rules can be modeled with the general form rule model of this research by associating specific behaviors (i.e., sequences of states) with specific rule conditions.

Finally, few researchers identified or acknowledged the explicitly temporal nature of rules in their rule models, with Theodoulidis et al. (1992) being the rare exception. Considering that this rule model and the associated rule algebra are built on temporal logic, the temporal nature of rules are explicitly acknowledged and directly incorporated.

With respect to the application of the rule model and rule algebra for rule analysis, the rule algebra is applicable within the context of other rule analysis techniques such as program slicing. The rule model and rule algebra can be used in close association with program slicing to further reduce sliced code. As demonstrated in Chapter 7, the rule model and rule algebra can be applied to create state sequence slices, a type of logical slice heretofore not investigated nor applied in the other program slicing research reviewed.

With respect to the graphical analysis and representation of rules, the association of the rule model and rule algebra developed as part of this research with statecharts has been demonstrated. The ability to both nest and hide rules using statecharts is consistent with the objectives of techniques presented by Storey and Muller (1995). As statecharts have achieved a relatively widespread acceptance and understanding, the use of statecharts for graphical rule representation is preferable to the use of specialized graphical objects such as those used in Feijs and de Jong (1998).

In Section 2.9, six critical shortcomings are identified regarding existing rule analysis and extraction procedures. The rule model and rule algebra developed as part of this research and the associated rule analyses in both the reverse and forward engineering domains address the critical shortcomings. Firstly, the rule model presented in the research is explicit with regard to what is meant by the concept of a rule. Using ITL, the formal semantics of the general rule form presented in this research are well defined. Secondly, the rule model presented in this research is language independent. Therefore, this rule model and the associated rule algebra are ideal for application in heterogeneous environments. Thirdly, use of ITL as a formal notation eliminates the impact of alternative syntax in the analysis process and maintains focus on the semantic elements of the rule. Fourthly, use of ITL as a formal notation for the representation of rules minimizes the potential for variation in rule representation and interpretation by different practitioners in the analysis process. Fifthly, expressing the rule model and the rule algebra in ITL allows for formal and provable analyses. Finally, the rule model and rule algebra presented in the research support both the reverse and forward engineering analysis of rules.

# Chapter 11

# Conclusions and Recommendations for Future Research

In this chapter, the underlying vision that prompted this research is reviewed, the significant achievements associated with this research are enumerated, and some promising directions for future research are suggested.

## 11.1 Vision

As asserted in the introduction, rules give structure to knowledge. Within this context, knowledge-based business practices are structured by rules. Rules specify what is expected, what is preferred, what is a priority, what is allowable, and what is unacceptable. Within an organization, these rules are incorporated into computerized business/knowledge systems based on the organizational experiences and expectations so that all users of these systems are either guided or constrained (depending on the rule) with regard to their choice of behaviors. Over time, these rules are changed, refined, and/or updated to reflect acquired additional knowledge regarding successful and unsuccessful practices. Using this rule-based model of business practices, two different information systems, or more specifically two different program code elements, can be compared based on similarities and/or differences in their component rules. Should it be necessary to integrate these two systems or re-engineer a single replacement system, these rules can form the functional basis for the new system.

Therefore, this rule-based model forms a rational basis for the analysis of heterogeneous business systems. Within these systems, the component rules are used to express the knowledge-based business practices of the organization. If one identifies and extracts these rules, the refined knowledge expressed in the business system can be preserved, analyzed, and reused as desired.

Within the context of this rule-based model of knowledge-based business information systems, three fundamental questions emerged:

1. What is a rule?
2. Can rules be extracted from a diversity of different types of information systems?

3. Once extracted, can these extracted rules be manipulated and analyzed to yield information about the original system and/or to allow comparisons with other rules?

This rule-based model of knowledge-based business information systems and these three associated questions have driven the research presented herein.

## 11.2 Achievements

Within the context of the vision described above, the following eight achievements have been realized with the research:

1. A set-based formal framework is presented that allows the description and analysis of a program or information system as a set of structures that are describable as rules and non-rules. With this formal framework, the feasibility of representing information system as rules and extracting those rules is demonstrated, subject to the formalization of a sufficiently general definition of a rule.

2. A general formal model of a rule is developed, general in that it can be adapted to the variety of languages and programming paradigms that might be encountered in different legacy code applications. Using Interval Temporal Logic (ITL), a rule is defined formally as a conjunctive and temporal relationship between a state sequence and a future state sequence. Using the ITL next operator $\circ$, a general-form rule is defined as $f_i \wedge \circ f_j$, where $f_i$ describes the state sequence that satisfies the rule condition and $f_j$ describes the future (i.e., next) state sequence that satisfies the rule state. Informally, this ITL formula describes a rule as a conjunctive and temporal relationship between a state sequence satisfying the rule condition $f_i$, and a future state sequence satisfying the rule state $f_j$. Given the underlying simplicity of this definition – that a rule is a temporal relationship between two state sequences – no arbitrary limitations are introduced with this definition. Therefore, the general-form rule $f_i \wedge \circ f_j$ can be used for both reverse engineering of existing systems and forward engineering of new systems.

3. Using this general formal model, a rule algebra is developed that describes the set of operations that can be applied to compose, decompose, or transform rules. This rule algebra is developed incrementally by considering fundamental systems and presenting rules that describe the relationships between the state sequences that compose these fundamental systems. Given the underlying formations of the rule algebra and the depth of ITL, this rule algebra is adaptable. In addition to the 43 lemmas presented to describe this rule algebra, additional lemmas can be developed to support and expand the rule algebra as needed.

4. In developing this rule algebra, significant attention is given to composing rules and rule systems to describe larger and more complex state sequences. Compositional paradigms demonstrated with this rule algebra include sequential composition, nesting, recursion, deterministic and non-deterministic guarded composition, and disjoint parallel composition. Using these compositional paradigms, rule-based representations of typical legacy code structures – the if-then-else structure, the while structure, and the indexed for-loop – are developed.

5. Within the context of the formal rule model and the corresponding rule algebra, the use and the value of statecharts for legacy code analysis are demonstrated. Generic statecharts of different rule-based coding paradigms are developed. These generic statecharts are applied and various rule-based legacy code structures are presented as both ITL formulas and statecharts. The statechart approach allows a visual presentation that is readily understandable by a wider audience, and the formulaic approach of representing the extracted rules as ITL formulas is readily adaptable to computer analysis techniques. Coupled with the rule model and rule algebra, statecharts are demonstrated to be a robust approach to managing the 'state explosion' problem that may result in the analysis and extraction of rules in real-world legacy systems.

6. Using this rule algebra, rules are extracted from a range of rule-based systems, specifications, and legacy code: an existing finite state machine, a detailed formal specification, a small but relatively complicated block of Pascal legacy code, and a block of code from a tax calculation program. The flexibility and adaptability of this rule algebra are demonstrated both with the types of systems from which rules are extracted and with the transformations and analyses that are achieved using the extracted rules. In these rule extraction exercises, the application of this rule algebra is demonstrated to be compatible with other traditional approaches to legacy code analysis including traditional slicing, conditioned and semantic slicing, program simplification, program transformation, and database approaches.

7. To demonstrate the applicability of this rule algebra with respect to forward engineering, rules are developed using this rule model and rule algebra to describe two systems – specification of a new business process and a simple hardware system.

8. With the reverse and forward engineering applications described herein, the rule model and rule algebra, as developed in the research, are demonstrated to be a robust, flexible, and expressive approach for the extraction of rules from a spectrum of existing or legacy systems, the forward engineering of new rule-based systems, the formal transformation and analysis of rules, and system comprehension.

## 11.3 Future Research Directions

In the course of this research, as with any journey, numerous interesting avenues were observed but left unexplored. In this section, some possible research directions are discussed.

### 11.3.1 Equivalence and Isomorphism

Two of the most important concepts that merit future research are equivalence and isomorphism. Informally, the two concepts relate to the fundamental questions "Are these rules the same?" and "If they are not the same, then are they similar?" Three forms of equivalence – strong equivalence (or strong bisimulation), transformational equivalence, and non-temporal equivalence – are discussed in Section 6.5. Given the knowledge-basis for this rule approach to assessing legacy systems (as articulated in the vision described in Section 11.1), demonstrating the extent of equivalence between two rules is critical. Therefore, formalization of the three equivalence models presented in Section 6.5 and the development of alternative equivalence models is an important research direction. Within the context that there are multiple models of equivalence and that the assessment of equivalence is not strictly binary, the formalization of equivalence models for rules is critical for rendering domain-specific judgments that two rules are sufficiently equivalent for a given domain-specific application.

The root of the word isomorphism is derived from two Greek words – *iso* meaning the 'same' and *morphe* meaning 'form.' Unlike rule equivalence, which is concerned with whether two rules are the same with respect to rule states, rule conditions, input/output, and observable changes, isomorphism considers whether two rules have the same structural form. With respect to legacy code analysis, isomorphic rules may suggest multiple implementations of similar rules.

At a minimum, for two rules to be isomorphic there must exist a bijective function such that each state sequence in the domain of the first rule maps to a state sequence in the domain in the second rule, and a second bijective function such that each state sequence in the codomain of the first rule maps to a sequence in the codomain in the second rule. As these two functions are bijective, two inverse bijective functions must exist mapping the state sequences of the domain and codomain of the second rule to state sequences in the domain and codomain, respectively, of the first rule. Whereas these bijective functions are minimum requirements and additional properties may be necessary to prove an isomorphism between the two rules, these minimum requirements contribute to the following demonstration.

Consider these two rules:

$$f_0 \wedge \circ f_1 \qquad (11.3.1\text{-}1)$$

$$\circ f_0 \wedge \circ\circ f_1 \qquad (11.3.1\text{-}2)$$

Are these two rules the same? If not the same, are they similar? And if so, how similar? Given that $f_0 \equiv f_0$ and $f_1 \equiv f_1$, the existence of the two bijective functions and their inverses is assured. Therefore, an isomorphism between (11.3.1-1) and (11.3.1-2) may exist (subject to any additional requirements that are added in a formal and complete definition of rule isomorphism). By inspection, (11.3.1-1) and (11.3.1-2) differ only by the presence of an additional ITL next operator $\circ$ in (11.3.1-2). Therefore and informally, (11.3.1-1) and (11.3.1-2) can be described as non-temporally equivalent. However, in the absence of a domain-specific assertion of the form $f \supset \circ f$, neither non-temporal equivalence nor transformational equivalence can be proven formally at this time and without additional research. This simple example illustrates the need for additional research regarding equivalence and isomorphism within the context of rules.

## 11.3.2 Alternative Rule Forms

This research has centered on the general rule form $f_i \wedge \circ f_j$. As developed in Chapter 4, this rule form describes a temporal relationship between the state sequence $\sigma_i$ where $\sigma_i \models f_i$ and the future state sequence $\sigma_j$ where $\sigma_j \models f_j$. However, various alternatives can be created using ITL to describe similar temporal relationships. In this section, several of these alternatives are discussed to highlight additional research directions.

As demonstrated in Section 8.2, the ITL construct $\mathsf{fin}\, f$ is a powerful technique for describing or specifying a set of properties, describable by $f$, that must hold in the final subinterval of a given state sequence. Therefore, the general rule form used in this research can be extended such that a rule is defined as:

$$f_i \wedge \circ f_j \wedge \mathsf{fin}(w_k) \tag{11.3.2-1}$$

In (11.3.2-1), $f_i$ specifies the rule conditions that must be met, $f_j$ describes the rule state, and $\mathsf{fin}(w_k)$ describes the state properties of the final state sequence of the state sequence specified by the rule. Within the context of the stated objectives associated with this alternative form, the following form is tempting:

$$f_i \wedge \circ f_j \wedge (\circ f_j \supset \mathsf{fin}(w_k)) \tag{11.3.2-2}$$

However, representing the implication in (11.3.2-1) as $\neg \circ f_j \vee \mathsf{fin}(w_k)$ and with the application of propositional logic, (11.3.2-1) and (11.3.2-2) can be demonstrated to be equivalent. Therefore, either (11.3.2-1) or (11.3.2-2) capture the fundamental notion of this alternative approach. Finally, applying propositional logic to (11.3.2-1), the following can be concluded:

$$f_i \wedge \mathsf{fin}(w_k) \qquad (11.3.2\text{-}3)$$

Note that the reduced form of (11.3.2-3) mimics the form of the consequents in (8.2-57). Therefore, the alternative form $f_i \wedge \circ f_j \wedge \mathsf{fin}(w_k)$ may hold distinct advantages in the analysis of legacy systems.

Extending the concepts embodied in (11.3.2-1) and loosely borrowing from the 'always-followed-by' construct proposed by Siewe et al. (2003) as discussed in Chapter 4, consider the following alternative representation of a rule:

$$f_i \wedge \circ f_j \,;\, w_k \qquad (11.3.2\text{-}4)$$

In (11.3.2-4), $f_i$ and $f_j$ are as previously described and $w_k$ describes the state properties of a state sequence that 'follows' $f_j$, subject to the semantics of the ITL chop operator.

The general rule form $f_i \wedge \circ f_j$ used in this research incorporates the ITL next operator. However, the use of the ITL next operator may cause problems in certain logical and programming constructs, including certain forms of parallelism. Therefore, temporal relationships described in Chapter 4 can be formalized using the ITL sometimes $\Diamond$ operator. Under this paradigm, an alternative rule definition is:

$$f_i \wedge \Diamond f_j \qquad (11.3.2\text{-}5)$$

This form has the advantage that the rule state satisfying $f_j$ need not hold in the next state sequence, but can hold instead in some state sequence including an eventual state sequence some time in the future. Using the concepts described in (11.3.2-1) and (11.3.2-4), other alternative rule forms based on the ITL sometimes $\Diamond$ operator include:

$$f_i \wedge \Diamond f_j \wedge \mathrm{fin}(w_k) \qquad (11.3.2\text{-}6)$$

$$f_i \wedge \Diamond f_j \,;\, w_k \qquad (11.3.2\text{-}7)$$

Whereas no representation is made at this time regarding the superiority of any of these forms relative to each other or to form $f_i \wedge \circ f_j$ as used in this research, these alternative forms may have some distinct advantages in certain circumstances. Therefore, these alternative forms, and any other related forms that support the temporal relationship concepts as developed in Chapter 4, merit additional investigation.

## 11.3.3 Interdependence, Independence, and Interference

Fundamental to the rule model presented in Chapter 4 is the concept of the state – a function mapping a set of variables to a set of values. Given the general form rule $f_i \wedge \circ f_j$, each rule considers at least two state sequences, one described by $f_i$ and another described by $\circ f_j$. Therefore, in the specification of the rule elements $f_i$ and $\circ f_j$, various sets of variables are used to describe the satisfying state sequences. These variable sets can be used as a basis for comparison and analysis. This concept can be applied in at least two ways – the comparison of the various elements within a single rule and the comparison of two or more rules with each other. This concept has been previously applied in Section 6.4 with regard to the assessment of the independence of any two ITL formulas.

However, a more sophisticated model is desirable as it may afford a significantly more detailed assessment of how rule elements and rules are similar or dissimilar. With regard to the variable sets used to formulate rules, at least three concepts merit additional research and formalization – interdependence, independence, and interference. Rule interdependence describes how the various elements of a single rule are or are not interrelated. Rule independence describes how two rules are or are not interrelated. Rule inference describes how two rules may conflict with regard to the assessment of the verity of the formulas describing the rule condition and the rule state.

To highlight how such additional research and formalization might be prosecuted, consider the following model. Let *rule* be a general-form rule defined as $f_i \wedge \circ f_j$, and let the sets $C$, $V$, and $W$ be sets of variables defined as follows:

$W \triangleq$ The set of state variables that can possibly change values under *rule* such that $f_i$ and $\circ f_j$ holds. This is typically referred to as the frame.

$V \triangleq$ The set of state variables used to specify, calculate, or otherwise define the new values of variables in the set $W$.

$C \triangleq$ The set of state variables used to specify the rule condition $f_i$.

Using this model, a variety of potentially useful concepts can be defined formally, as described below.

For a non-interdependent rule:

$$C \cap V \cap W = \varnothing \tag{11.3.3-1}$$

Informally, for a non-interdependent rule, the formula specifying the rule condition does not include any frame variables or any variables used in calculating the new values for the frame variables.

For a maximally interdependent rule:

$$C = V = W \tag{11.3.3-2}$$

Informally, for a maximally interdependent rule, all variables used in specifying the rule conditions are also in the frame and are also used to calculating the new values for the frame variables.

Regarding rule independence, consider two rules, $rule_1$ with $C_1$, $V_1$, and $W_1$, and $rule_2$ with $C_2$, $V_2$, and $W_2$. These two rules are totally independent if:

$$(C_1 \cup V_1 \cup W_1) \cap (C_2 \cup V_2 \cup W_2) = \varnothing \tag{11.3.3-3}$$

Informally, the two rules $rule_1$ and $rule_2$ are totally independent if they have no variables in common. Continuing with this concept of rule independence, these two rules are rule condition independent if:

$$C_1 \cap C_2 = \varnothing \tag{11.3.3-4}$$

Rule interference is a potential problem with regard to parallel composition. Informally, two rules in parallel may interfere with each other if a variable in the frame of one rule is used to specify the rule condition of the other rule or is used to calculate the value of a variable in the frame of the other rule. Formally, potential interference may exist between *rule*$_1$ and *rule*$_2$ if:

$$\begin{aligned}&(C_1 \cap W_2 \neq \emptyset) \vee (C_2 \cap W_1 \neq \emptyset)\\ &\vee (V_1 \cap W_2 \neq \emptyset) \vee (V_2 \cap W_1 \neq \emptyset)\end{aligned} \tag{11.3.3-5}$$

The above representations of rule interdependence, rule independence, and rule interference are very basic; other more appropriate and/or more detailed formalizations likely exist. However, these general formalizations do provide a solid basis for understanding the importance of these concepts, and the importance of additional research in these areas. Given the nature and scope of rule interdependence, rule independence, and rule interference, the adequate formalization of these concepts can be an important segue to other rule analysis and research issues.

### 11.3.4 Detemporalization

In Section 8.2, selected implications of the form $\circ w_i \supset w_i$ or $\circ\circ w_i \supset w_i$ were asserted to detemporalize specific rule conditions to facilitate rule simplification and analysis. Given the absence of parallelism in the target code, these assertions were supported by a code-specific analysis and assessment of the non-interdependence of the rule condition variables and the rule state variables. A formal definition of non-interdependence between the elements of a given rule is presented at (11.3.3-1). As demonstrated in Section 8.2, such detemporalization is a powerful pathway to rule simplification. Therefore, additional research into detemporalization, including a formal approach and basis for detemporalization, is very important. In the absence of such formalization, detemporalization can only be achieved by code-specific analysis and reasoning.

### 11.3.5 Formal Proof of Equivalence of Specific Statechart Constructs and Specific Rule Formulas

In Section 7.3, strong correspondences between specific statechart constructs and specific rule formulations were demonstrated. With these correspondences, the value of statecharts, when used in concert with the rule algebra presented herein, was demonstrated with regard to legacy code analysis. However, no formal proofs of equivalence between specific statechart constructs and specific rule formulas were presented. Given the demonstrated value of statecharts in legacy code analysis, and noting the general scarcity of other research regarding statecharts and legacy code, this could be a rich area for significant research.

### 11.3.6 Metadata About Rules

This research has focused on the analysis of legacy systems within the context of a formal rule definition and rule algebra. However, significant benefit can be realized through the analysis of the rules themselves. The database approach to legacy code analysis as presented in Section 8.1 incorporated with this concept in that the some of the database fields (i.e., *W*, *V*, and Primary Membership) were derived from the properties of the rules. Other rule properties that could be used to describe the rules themselves include the extent of nesting, the use of recursion, non-determinism, the *C* variable set as defined in Section 11.3.3, and the rule properties of interdependence, independence, and interference. As this area has not been investigated in this research, the depth and potential of such research cannot be quantified. However, given the formal basis of this rule algebra, its use to classify rules seems to be a reasonable and rational extension.

### 11.3.7 Automated Tool Using the Rule Algebra

As presented in this research, this rule algebra is demonstrated to be a robust, flexible, and expressive approach for the extraction of rules from a spectrum of existing or legacy systems. The development and implementation of an automated tool using this rule algebra approach would allow easier and faster analysis of a range of legacy systems, and could significantly speed the testing and expansion of the underlying rule algebra.

## References

Aiken, P., Muntz, A., & Richards, R. (1993). A framework for reverse engineering DoD legacy information systems. *Proceedings of the Working Conference on Reverse Engineering*, 180-191.

Alagar, V. S., & Periyasamy, K. (2001). BTOZ: A formal specification language for formalizing business transactions. *39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS 39),* 240-252.

Apt, K. R., & Olderog, E.-R. (1997). *Verification of sequential and concurrent programs.* New York: Springer-Verlag.

Arnold, A. (1994). *Finite transition systems.* Englewood Cliffs, NJ: Prentice-Hall

Back, R.-J. (1988). A calculus of refinements for program derivations. *Acta Informatica, 25*, 593-624.

Baeten, J. C. M. & Weijland, W. P. (1990). *Process algebra.* Cambridge, UK: Cambridge University Press.

Bennett, K. H., Bull, T., & Yang, H. (1992). A transformation system for maintenance – Turning theory into practice. *Proceedings of the International Conference on Software Maintenance (ICSM)*, 146-155.

Binkley, D., & Gallagher, K. (1996). A survey of program slicing. In M. Zelkowitz (Ed.), *Advances in Computers, 43*, 1-50. New York: Academic Press.

Binkley, D., Harman, M., Raszewski, L.R., & Smith, C. (2000). An empirical study of amorphous slicing as a program comprehension support tool. *Proceedings of the 8th International Workshop on Program Comprehension (IWPC 2000)*, 161-170.

Birkhoff, G. & MacLane, S. (1977). *A survey of modern algebra* (4th ed.). New York: Macmillan.

Blazy, S., & Facon, P. (1997). Application of formal methods to the development of a software maintenance tool. *Proceedings of the 12th IEEE International Conference Automated Software Engineering*, 162-171.

Bowen, J., Breuer, P., & Lano, K. (1993). A compendium of formal techniques for software maintenance. *IEE/BCS Software Engineering Journal*, *8*(5), 253-262.

Bratko, I. (2001). *Prolog programming for artificial intelligence* (3rd ed.). New York: Addison Wesley.

Britt, J.J. (1994). Case study: Applying formal methods to the Traffic Alert and Collision Avoidance System (TCAS) II. *Proceedings of the Ninth Annual Conference on Computer Assurance (COMPASS '94)*, 39-51.

Büchi, J. R. (1989). *Finite automata, their algebras and grammars: Towards a theory of formal expressions* (D. Siefkes, Ed.). New York: Springer-Verlag.

Bull, T. (1990). An introduction to the WSL program pransformer. *Proceedings of the International Conference on Software*, 242-250.

Burris, S. & Sankappanavar, H. P. (1981). *A course in universal algebra.* New York: Springer-Verlag.

Cau, A. & Moszkowski, B. (1996). *Using PVS for Interval Temporal Logic proofs. Part 1: The syntactic and semantic encoding.* (Technical Monograph 14). Leicester, UK: SERCentre, De Montfort University.

Cau, A. & Zedan, H. (1997). Refining Interval Temporal Logic specifications. In M. Bertran and T. Rus, (Eds.), *Transformation-Based Reactive Systems Development, Lecture Notes in Computer Science 1231,* 79-94. Springer Verlag.

Cau, A. & Zedan, H. (2000). Chapter 21: The systematic construction of information systems. In P. Henderson (Ed.), *Systems engineering for business process change* (pp. 264-278). Springer Verlag.

Cau, A. & Zedan, H. (2006). A practitioner's approach to reverse engineering through abstraction. Preprint submitted to Elsevier Science.

Corbett, J. C., Dwyer, M. B., Hatcliff, J., Laubach, S., Pasareanu, C.S., Robby, & Zheng, H. (2000). Bandera: Extracting finite-state models from Java source code. *Proceedings of the 2000 International Conference on Software Engineering*, 439-448.

Danicic, S., Daoudi, M., Fox, C., Harman, M., Hierons, R. M., Howroyd, J., Ourabya, L., & Ward, M. (2005). ConSUS: a light-weight program conditioner, *Journal of Systems and Software, 77*(3) 241-262.

De Giacomo, G., Lesperance, Y., & Levesque, H. J. (2000). ConGolog, a concurrent programming language based on the situation calculus. *Articifial Intelligence 121*(1-2) 109-169.

De Nicola, R. D. & Hennessy, M. C. B. (1984). Testing equivalences for processes. *Theoretical Computer Science, 34*(1-2) 83-133.

Denecke, K. & Wismath, S. L. (2002). *Universal algebra and applications in theoretical computer science.* Boca Raton: Chapman & Hall

Dijkstra, E. W. (1975). Guarded commands, nondeterminacy, and the formal derivation of programs. *Communications of the ACM 18*, 453-457.

Dijkstra, E. W. (1976). *A discipline of programming.* Englewood Cliffs, N.J.: Prentice-Hall.

Feijs, L., & de Jong, R. (1998). 3D visualization of software architectures. *Communications of the ACM, 41*(12) 73-78.

Feynman, R. P. (1996). *Feynman lectures on computation* (A. Hey & R. Allen. Eds.). Reading, Mass.: Addison-Wesley.

Fokkink, W. (2000). *Introduction to process algebra.* New York: Springer.

Fox, C., Harman, M., Hierons, R., & Danicic, S. (2000). ConSIT: A conditioned program slicer. *IEEE International Conference on Software Maintenance (ICSM'2000)*, 216-226

Francel, M.A., & Rugaber, S. (1999). The relationship of slicing and debugging to program understanding. *Proceedings of the Seventh International Workshop on Program Comprehension*, 106-113.

Fu, G., Shao, J., Embury, S. M., Gray, W. A., & Liu, X. (2001). A framework for business rule presentation. *Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, 922-926.

Gannod, G. C., & Cheng, B. H. C. (1996). Using informal and formal techniques for the reverse engineering of C programs. *Proceedings of the Third Working Conference on Reverse Engineering*, 249-258.

Gannod, G. C., & Cheng, B. H. C. (1999). A formal approach for reverse engineering: A case study. *Proceedings of the Sixth Working Conference on Reverse Engineering*, 100-111.

Gannod, G. C., & Cheng, B. H. C. (2001). A suite of tools for facilitating reverse engineering using formal methods. *Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001)*, 221-232.

Gill, A. (1976). *Applied algebra for the computer sciences.* Englewood Cliffs, N.J.: Prentice-Hall.

Giomi, J.C. (1995). Finite state machine extraction from hardware description languages. *Proceedings of the Eighth Annual IEEE International ASIC Conference and Exhibit*, 353-357.

Grosof, B. N., Labrou, Y., & Chan., H. Y. (1999). A declarative approach to business rules in contracts: Courteous logic programs in XML. *ACM Special Interest Group on E-Commerce (EC99)*, 68-77.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, 231-274.

Harel, D. & Naamad, A. (1996). The STATEMATE Semantics of Statecharts. *ACM Trans. on Software Engineering and Methodology*, *5*(4) 293-333.

Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtul-Trauring, A. & Trakhtenbrot, M. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Trans. on Software Engineering 16*(4) 403-414.

Hartmanis, J., & Stearns, R. E. (1966). *Algebraic structure theory of sequential machines.* Englewood Cliffs, N.J.: Prentice-Hall

He, J., Seidel, K., & McIver, A. (1997). Probabilistic Models for the Guarded Command Language. *Science of Computer Programming, 28*, 171-192.

Herbst, H. (1995). A meta-model for business rules in systems analysis. In J. Iivari, K. Lyytinen, & M. Rossi (Eds.), *Proceedings of the Seventh Conference on Advanced Information Systems Engineering (CAiSE '95),* 186-199. Berlin: Springer.

Herbst, H., Knolmayer, G., Myrach, T., & Schlesinger, M. (1994). The specification of business rules: A comparison af selected methodologies. In A.A. Verijn-Stuart & T.W. Olle (Eds.), *Methods and Associated Tools for the Information System Life Cycle*, 29-46. Amsterdam: Elsevier.

Hoare, C. A. R. (1975). Parallel programming: An axiomatic approach. In F. L. Bauer & K. Samelson (Eds.), *Language Hierarchies and Interfaces, Language Hierarchies and Interfaces, Lecture Notes in Computer Science 46*, 11-42.

Hoare, C. A. R. (1985). A couple of novelties in the propositional calculus. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, 31*(2), 173-178.

Huang, H., Tsai, W.T., Bhattacharya, S., Chen, X.P., Wang, Y., & Sun, J. (1996). Business rule extraction from legacy code. *Proceedings of the 20th International Computer Software and Applications Conference (COMPSAC '96)*, 162-167.

Huang, H., Tsai, W.T., Bhattacharya, S., Chen, X., Wang, Y., & Sun, J. (1998). Business rule extraction techniques for COBOL programs. *Journal of Software Maintenance: Research and Practice, 10*(1), 3-35.

Hungerford, T. W. (1974). *Algebra.* New York: Springer.

Koubarakis, M., & Plexousakis, D. (1999). Business process modelling and design – A formal model and methodology, *BT Technology Journal, 17*(4) 23-35.

Lamport, L. (1977). Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering SE-3*(2), 125-143.

Lamport, L. (1980). "Sometime" is sometimes "not never". *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages, ACM SIGACT-SIGPLAN,* 174-185.

Lamport, L. (1994). The temporal logic of actions. *ACM Transactions on Programming Languages and Systems, 16*(3), 872-923.

Lanubile, F., & Visaggio, G. (1997). Extracting reusable functions by flow graph-based program slicing. *IEEE Transactions on Software Engineering*, 23, 246-159.

Leite, J.C.S.d.P., & Leonardi, M.C. (1998). Business rules as organizational policies. *Proceedings of the Ninth International Workshop on Software Specification and Design*, 68-76.

Levy, L. S. (1980). *Discrete structures of computer science.* New York: Wiley.

Liu, X., Yang, H., & Zedan, H. (1997). Formal methods for the re-engineering of computing systems: A comparison. *Proceedings of the Twenty-First Annual International Computer Software and Applications Conference (COMPSAC '97)*, 409-414.

Mancoridis, S., Mitchell, B.S., Chen, Y., & Gansner, E.R. (1999). Bunch: A clustering tool for the recovery and maintenance of software system structures. *Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99)*, 50-59.

Manna, Z. & Pnueli, A. (1990). A hierarchy of temporal properties. *9th Symposium on Principles of Distributed Computing*, 377-408.

Manna, Z. & Pnueli, A. (1992). *The temporal logic of reactive and concurrent systems: Specification.* New York: Springer-Verlag.

Manna, Z. & Pnueli, A. (1995). *The temporal logic of reactive and concurrent systems: Saftey.* New York: Springer-Verlag.

*Merriam-Webster Dictionary.* (1998). Springfield, MA: Merriam-Webster.

Milner, R. (1980). A calculus for communicating systems. *Lecture Notes in Computer Science 92.*

Milner, R. (1989). *Communication and concurrency.* New York: Prentice Hall.

Morgan, C., & McIver, A. (1999). pGCL: Formal reasoning for random algorithms. *South African Computer Journal, 22*, 14-27.

Moszkowski, B. (1986). *Executing temporal logic programs.* Cambridge, UK: Cambridge University Press.

Moszkowski, B. (1994). Some very compositional temporal properties. In E.-R. Olderog, (Ed.), *Programming Concepts, Methods and Calculi, Vol. A-56 of IFIP Transactions*, 307-326. North-Holland: Elsevier Science B.V.

Moszkowski, B. (1996). Using temporal fixpoints to compositionally reason about liveness. In H. Jifeng, J. Cooke, & P. Wallis (Eds.) *BCS-FACS 7th Refinement Workshop, Electronic Workshops in Computing*, (1-28). Springer-Verlag and British Computer Society.

Moszkowski, B. (2000). A complete axiomatization of interval temporal logic with infinite time. *15th Annual IEEE Symposium on Logic in Computer Science*, 241-252.

Moszkowski, B. (2003). A hierarchical completeness proof for interval temporal logic with finite time. In V. Goranko and A. Montanari (Eds.), *Proceedings of the ESSLLI*

*Workshop on Interval Temporal Logics and Duration Calculi*, 41-65. Vienna: Technical University of Vienna.

Murphy, G.C., Notkin, D., & Lan, E.S.-C. (1996). An Empirical Study of Static Call Graph Extractors. *Proceedings of the 18th International Conference on Software Engineering*, 90-99.

Ning, J.Q., Engberts, A., & Kozaczynski, W. (1993). Recovering reusable components from legacy systems by program segmentation. *Proceedings of Working Conference on Reverse Engineering*, 64-72.

Odell, J.J. (1995). Business rules. *Journal of Object Oriented Program.* Reprinted in Odell, J.J. (1998). *Advanced Object-Oriented Analysis & Design Using UML.* (pp. 99-107). Cambridge, UK: Cambridge University Press.

Owicki, S. & Lamport, L. (1982). Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems, 4*(3), 455-495.

*Oxford English Dictionary.* (1971). Oxford, UK: Oxford University Press

Park, D. (1981). Concurrency and automata on infinite sequences. In P. Deussen, (Ed.) *Proceedings of the International Conference on Theorical Computer Science, Lecture Notes in Computer Science 104*, 167-183. Springer-Verlag.

Penteado, R., Masiero, P.C., & Cagnin, M. I. (1999). An experiment of legacy code segmentation to improve maintainability. *Proceedings of the Third European Conference on Software Maintenance and Reengineering*, 111-119.

Perkins, A. (2000). Business rules = Meta-data. *Proceedings of the 34th International Conference on Technology of Object-Oriented Languages and Systems*, 285-294.

Petry, B. L. (1996). Getting the most out of legacy code: The uses of hypercode within a typical IS organization. *Proceedings of the IEEE 1996 National Aerospace and Electronics Conference, 2*, 852-857.

Pitts, A. M. (1997). Operationally-based theories of program equivalence. In P. Dybjer & A. M. Pitts (Eds), *Semantics and Logics of Computation*, 241-298. Cambridge, UK: Cambridge University Press.

Plexousakis, D. (1995). Simulation and analysis of business processes using GOLOG. *Proceedings of the Conference on Organizational Computing Systems (COOCS'95)*, 311-323.

Popovic, M., Kovacevic, V., & Velikic, I. (2002). A formal software verification concept based on automated theorem proving and reverse engineering. *Proceedings of the Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'02)*, 59-66.

Presley, A., & Rogers, K. J. (1996). Process Modeling to Support Integration of Business Practices and Processes in Virtual Enterprises. *Proceedings of the International Conference on Engineering and Technology Management (IEMC 96)*, 475-479.

Ritsch, H. & Sneed, H. M. (1993). Reverse engineering programs via dynamic analysis. *Proceedings of the Working Conference on Reverse Engineering (WCRE 1993)*, 192-201.

Roscoe, A. W., & Hoare, C. A. R. (1986). *Laws of occam programming - Technical monograph PRG-53*, Oxford, UK: Oxford University Computing Laboratory - Programming Research Group.

Ross, R.G.. (1997). *The business rule book - Classifying, defining and modeling rules.* Houston: Business Rule Solutions LLC

Rouvellou, I., Degenaro, L., Rasmus, K., Ehnebuske, D., & McKee, B. (2000). Extending business objects with business rules. *International Conference on Technology of Object-Oriented Languages (TOOLS 33)*, 238-249.

Scott, M. L. (2000). *Programming language pragmatics.* San Francisco: Morgan Kaufmann.

Sebesta, R. W. (2002). *Concepts of programming languages* (5th ed.). Boston: Addison Wesley.

Sellink, A., Sneed, H., & Verhoef, C. (1999). Restructuring of COBOL/CICS legacy systems. *Proceedings of the Third European Conference on Software Maintenance and Reengineering*, 72-82.

Shao, J. & Pound, C. J. (1999). Extracting business rules from information systems, *BT Technology Journal, 17*(4), 179-186.

Siewe, F., Cau, A., & Zedan, H. (2003). A compositional framework for access control policies enforcement. In M. Backes, D. Basin, & M. Waidner (Eds.), *Proceedings of the ACM workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE'03)*, 32-42.

Sneed, H. M. (1998). Architecture and functions of a commercial software reengineering workbench. *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, 2-10.

Sneed, H. M. & Erdos, K. (1996). Extracting business rules from source code. *Proceedings of the Fourth Workshop on Program Comprehension*, 240-247.

Sneed, H. M. & Jandrasics, G. (1988). Inverse transformation of software from code to specification. *Proceedings of the IEEE Conference on Software Maintenance*, 102-109.

Song, Y. T., & Huynh, D. T. (1999). Forward dynamic object-oriented program slicing. *Proceedings of the IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET'99)*, 230-237

Stanat, D. F., & McAllister, D. F. (1977). *Discrete mathematics in computer science.* Englewood Cliffs, N.J.: Prentice-Hall.

Storey, M.-A. D., & Muller, H. A. (1995). Manipulating and documenting software structures using SHriMP views. *Proceedings of the International Conference on Software Maintenance*, 275-284.

STRL–Software Technology Research Laboratory. (2003). *Formal methods engineering or system modeling using finite-state machines.* Leicester, UK: Software Technology Research Laboratory, De Montfort University

STRL–Software Technology Research Laboratory. (2006). Interval temporal logic (ITL) homepage. http://www.cse.dmu.ac.uk/STRL/ITL//index.html.

Tan, H. B. K., & Kow, J. T. (2001) Extracting Code Fragment That Implements Functionality. *Journal of Software Maintenance and Evolution: Research and Practice, 13*, 53-75.

Theodoulidis, B., Alexakis, P., & Loucopoulos, P. (1992). Verification and validation of temporal business rules. *Proceeding of the 3rd International Workshop on the Deductive Approach to Information Systems and Databases*, 179-193.

Tip, F. (1995). A survey of program slicing techniques, *Journal of Programming Languages, 3*, 121-189.

Ulrich, W.M. (1999). Knowledge mining: Business rule extraction and reuse. *Cutter IT Journal*, 12(11), 21-26.

Ungureanu, V. & Minsky, N.H. (2000). Establishing business rules for inter-enterprise electronic commerce. *Proceedings of the 14th International Symposium on Distributed Computing (DISC2000), Lecture Notes in Computer Science 1914.*

van Gelder, A., Ross, K. A., & Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the Association for Computing Machinery, 38*(3), 620-650.

Villavicencio, G.& Oliveira, J. N. (2001). Reverse Program Calculation Supported by Code Slicing. *Proceedings of the Eighth Working Conference on Reverse Engineering*, 35-45.

von der Beeck, M. (2001). Formalization of UML-statecharts. In M. Gogolla & C. Kobryn (Eds.), *Proceedings of UML 2001, Lecture Notes in Computer Science 2185*, 406-442. Berlin: Springer.

Wang, T.H., & Edsall, T. (1998). Practical FSM analysis for Verilog. *Proceedings of the 1998 International Verilog HDL Conference and VHDL International Users Forum, (IVC/VIUF 1998)*, 52-58.

Ward, M. (1989). Proving program refinements and transformations. (D. Phil. Thesis, Oxford University, 1989).

Ward, M. (1999). Assembler to C migration using the FermaT transformation system. *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'99)*, 67-76.

Ward, M. (2000). Reverse engineering from assembler to formal specifications via program transformations. *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, 11-20.

Ward, M. (2001). The formal transformation approach to source code analysis and manipulation. *Proceedings of the First IEEE International Workshop on Source Code Analysis and Manipulation*, 185-193.

Ward, M. (2004). Pigs from sausages? Reengineering from assembler to C via FermaT transformations. *Science of Computer Programming, 52*, 213-255.

Ward, M., Zedan, H., & Hardcastle, T. (2005). Conditioned semantic slicing via abstraction and refinement in FermaT. *Proceedings of the 9th European Conference on Software Maintenance and Reengineering (CSMR 2005)*, 178-187.

Weiser, M. (1982) Programmers use slices when debugging. *Communications of the ACM, 25*(7), 446-452.

Yang, H. and Bennett, K. H. (1994). Extension of a transformation system for maintenance: Dealing with data-intensive programs. *Proceedings of the International Conference on Software Maintenance (ICSM 1994)*, 344-353

Yang, H., Liu, X., & Zedan, H. (2000). Abstraction: A key notion for reverse engineering in a system reengineering approach. *Journal of Software Maintenance: Research and Practice, 12*, 197-228.

Zedan, H. & Yang, H. (1998). A sound and practical approach to the re-engineering of time-critical systems. *2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR 1998)*, 220-223.

Zhao, J. (2000). A slicing-based approach to extracting reusable software architectures. *Proceedings of the Fourth European Software Maintenance and Reengineering*, 215-223.

Zhou, S., Zedan, H., & Cau, A. (1999). A framework for analysing the effect of 'change' in legacy code. *Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99)*, 411-420.

# Appendix A

## Supporting Lemmas for the Rule Algebra

The following is a presentation of several general proofs that have been used is support of other proofs that are presented elsewhere in this thesis. Proofs are presented for NextAndDistEqv, NextAndOrDistEqv, NextOrAndDistEqv, NextOrDistEqv, and TemporalContra.

LEMMA: NextAndDistEqv

$\vdash \circ(f_0 \wedge f_1) \equiv \circ f_0 \wedge \circ f_1$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 \wedge f_1 \equiv f_0 \wedge f_1$ | tautology |
| 2 | $f_0 \wedge f_1 \equiv (f_0 \wedge f_1)$ | 1, associativity of $\wedge$ |
| 3 | $\circ f_0 \wedge \circ f_1 \equiv \circ(f_0 \wedge f_1)$ | 2, ITL (NextAndNextEqvNextRule) |
| 4 | $\circ(f_0 \wedge f_1) \equiv \circ f_0 \wedge \circ f_1$ | 3, commutivity of $\equiv$ |

LEMMA: NextAndOrDistEqv

$\vdash \circ((f_0 \vee f_1) \wedge (f_2 \vee f_3)) \equiv (\circ f_0 \vee \circ f_1) \wedge (\circ f_2 \vee \circ f_3)$

Proof:

| | | |
|---|---|---|
| 1 | $\circ((f_0 \vee f_1) \wedge (f_2 \vee f_3)) \equiv \circ((f_0 \vee f_1) \wedge (f_2 \vee f_3))$ | tautology |
| 2 | $\equiv \circ(f_0 \vee f_1) \wedge \circ(f_2 \vee f_3)$ | 1, NextAndDistEqv |
| 3 | $\equiv (\circ f_0 \vee \circ f_1) \wedge \circ(f_2 \vee f_3)$ | 2, NextOrDistEqv |
| 4 | $\equiv (\circ f_0 \vee \circ f_1) \wedge (\circ f_2 \vee \circ f_3)$ | 3, NextOrDistEqv |

LEMMA: NextOrAndDistEqv

$\vdash \circ((f_0 \wedge f_1) \vee (f_2 \wedge f_3)) \equiv (\circ f_0 \wedge \circ f_1) \vee (\circ f_2 \wedge \circ f_3)$

Proof:

| | | |
|---|---|---|
| 1 | $\circ((f_0 \wedge f_1) \vee (f_2 \wedge f_3)) \equiv \circ((f_0 \wedge f_1) \vee (f_2 \wedge f_3))$ | tautology |
| 2 | $\equiv \circ(f_0 \wedge f_1) \vee \circ(f_2 \wedge f_3)$ | 1, NextOrDistEqv |
| 3 | $\equiv (\circ f_0 \wedge \circ f_1) \vee \circ(f_2 \wedge f_3)$ | 2, NextAndDistEqv |
| 4 | $\equiv (\circ f_0 \wedge \circ f_1) \vee (\circ f_2 \wedge \circ f_3)$ | 3, NextAndDistEqv |

LEMMA: NextOrDistEqv

$\vdash \circ(f_1 \vee f_2) \equiv \circ f_1 \vee \circ f_2$

Proof:

| | | |
|---|---|---|
| 1 | $f_0 ; (f_1 \vee f_2) \equiv (f_0 ; f_1) \vee (f_0 ; f_2)$ | ChopOrEqv (ITL) |
| 2 | $\mathsf{skip} ; (f_1 \vee f_2) \equiv (\mathsf{skip} ; f_1) \vee (\mathsf{skip} ; f_2)$ | 1, substitution of skip for $f_0$ |
| 3 | $\circ(f_1 \vee f_2) \equiv \circ f_1 \vee \circ f_2$ | 2, definition of $\circ$ (ITL) |

LEMMA: TemporalContra

$\vdash \circ f_0 \wedge \circ\neg f_0 \equiv \mathit{false}$

Proof:

| | | |
|---|---|---|
| 1 | $\circ f_0 \wedge \circ\neg f_0 \equiv \circ f_0 \wedge \circ\neg f_0$ | tautology |
| 2 | $\circ f_0 \wedge \circ\neg f_0 \equiv \circ(f_0 \wedge \neg f_0)$ | 1, NextAndDistEqv |
| 3 | $\circ f_0 \wedge \circ\neg f_0 \equiv \circ(\mathit{false})$ | 2, law of contradiction |
| 4 | $\circ f_0 \wedge \circ\neg f_0 \equiv \mathit{skip} ; \mathit{false}$ | 3, ITL (definition of *next*) |
| 5 | $\circ f_0 \wedge \circ\neg f_0 \equiv \mathit{false}$ | 4, ITL (semantics of *chop*) |

# Appendix B

## Formal Transformation of Rules

## Extracted from a Specification

In Section 7.2, rules are extracted from a concrete specification describing the operation of an automated teller machine. Five of the extracted total rules are considered in this formal transformation:

$$
\begin{aligned}
rule_{7.2\text{-}b} &\triangleq ((\mathit{atm\_non\_empty} \wedge \circ\text{wait\_customer} \,;\, \text{read\_card} \,;\, rule_{7.2\text{-}c}) \,;\, rule_{7.2\text{-}b}) \\
&\quad \vee (\neg \mathit{atm\_non\_empty} \wedge \text{empty}) \\
rule_{7.2\text{-}c} &\triangleq (\mathit{card\_disabled} \wedge \circ\text{take\_disabled\_card}) \\
&\quad \vee (\neg \mathit{card\_disabled} \wedge \circ\text{get\_pin} \,;\, rule_{7.2\text{-}d}) \\
rule_{7.2\text{-}d} &\triangleq (\mathit{max\_pin} \wedge \circ\text{disable\_card} \,;\, \text{take\_disabled\_card}) \\
&\quad \vee (\neg \mathit{max\_pin} \wedge \circ rule_{7.2\text{-}e}) \\
rule_{7.2\text{-}e} &\triangleq (\mathit{pin\_exit} \wedge \circ\text{take\_card\_pin\_exit}) \\
&\quad \vee (\neg \mathit{pin\_exit} \wedge \circ\text{request\_money} \,;\, rule_{7.2\text{-}f}) \\
rule_{7.2\text{-}f} &\triangleq (\mathit{money\_exit} \wedge \circ\text{take\_card\_money\_exit}) \\
&\quad \vee (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} \,;\, \text{take\_card\_money})
\end{aligned}
$$

To facilitate analysis, the following variable name substitutions are made:

| | |
|---|---|
| da $\triangleq$ | debit_account |
| dc $\triangleq$ | disable_card |
| gp $\triangleq$ | get_pin |
| rc $\triangleq$ | read_card |
| rm $\triangleq$ | request_money |
| tcm $\triangleq$ | take_card_money |
| tcme $\triangleq$ | take_card_money_exit |
| tcpe $\triangleq$ | take_card_pin_exit |
| tdc $\triangleq$ | take_disabled_card |
| wc $\triangleq$ | wait_customer |
| *xane* $\triangleq$ | *atm_non_empty* |
| *xcd* $\triangleq$ | *card_disabled* |
| *xme* $\triangleq$ | *money_exit* |
| *xmp* $\triangleq$ | *max_pin* |
| *xpe* $\triangleq$ | *pin_exit* |

Regarding these variable names, rule conditions variables begin with the letter $x$ and are depicted in italics. Using these rule conditions and rule state variable names, these five rules of interest are rewritten as:

$$rule_{7.2\text{-}b} \triangleq ((xane \land \mathrm{owc} ; \mathrm{rc} ; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b}) \lor (\neg xane \land \mathrm{empty})$$
$$rule_{7.2\text{-}c} \triangleq (xcd \land \mathrm{otdc}) \lor (\neg xcd \land \mathrm{ogp} ; rule_{7.2\text{-}d})$$
$$rule_{7.2\text{-}d} \triangleq (xmp \land \mathrm{odc} ; \mathrm{tdc}) \lor (\neg xmp \land \mathrm{o}rule_{7.2\text{-}e})$$
$$rule_{7.2\text{-}e} \triangleq (xpe \land \mathrm{otcpe}) \lor (\neg xpe \land \mathrm{orm} ; rule_{7.2\text{-}f})$$
$$rule_{7.2\text{-}f} \triangleq (xme \land \mathrm{otcme}) \lor (\neg xme \land \mathrm{oda} ; \mathrm{tcm})$$

Each of these rules is assumed as a premise. The formal transformation of these rules is as follows:

| | | |
|---|---|---|
| 1 | $rule_{7.2\text{-}b}$<br>where:<br>$rule_{7.2\text{-}b} \equiv ((xane \land \mathrm{owc} ; \mathrm{rc} ; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b})$<br>$\lor (\neg xane \land \mathrm{empty})$ | premise |
| 2 | $rule_{7.2\text{-}c}$<br>where:<br>$rule_{7.2\text{-}c} \equiv (xcd \land \mathrm{otdc}) \lor (\neg xcd \land \mathrm{ogp} ; rule_{7.2\text{-}d})$ | premise |
| 3 | $rule_{7.2\text{-}d}$<br>where:<br>$rule_{7.2\text{-}d} \equiv (xmp \land \mathrm{odc} ; \mathrm{tdc}) \lor (\neg xmp \land \mathrm{o}rule_{7.2\text{-}e})$ | premise |
| 4 | $rule_{7.2\text{-}e}$<br>where:<br>$rule_{7.2\text{-}e} \equiv (xpe \land \mathrm{otcpe}) \lor (\neg xpe \land \mathrm{orm} ; rule_{7.2\text{-}f})$ | premise |
| 5 | $rule_{7.2\text{-}f}$<br>where:<br>$rule_{7.2\text{-}f} \equiv (xme \land \mathrm{otcme}) \lor (\neg xme \land \mathrm{oda} ; \mathrm{tcm})$ | premise |
| 6 | $rule_{7.2\text{-}e}$ | 4, reiteration |
| 7 | $(xpe \land \mathrm{otcpe}) \lor (\neg xpe \land \mathrm{orm} ; rule_{7.2\text{-}f})$ | 4, 6, eqv. subst. |

| | | |
|---|---|---|
| 8 | $(xpe \land \mathrm{otcpe}) \lor (\neg xpe \land \mathrm{orm} ; ((xme \land \mathrm{otcme})$<br>$\lor (\neg xme \land \mathrm{oda} ; \mathrm{tcm})))$ | 5, 8, eqv. subst. |
| 9 | $(xpe \land \mathrm{otcpe}) \lor ((\neg xpe \land \mathrm{orm}) ; ((xme \land \mathrm{otcme})$<br>$\lor (\neg xme \land \mathrm{oda} ; \mathrm{tcm})))$ | 8, ITL<br>(StateAndChop) |
| 10 | $(\neg xpe \land \mathrm{orm}) ; ((xme \land \mathrm{otcme}) \lor (\neg xme \land \mathrm{oda} ; \mathrm{tcm}))$ | CP assumption |
| 11 | $(\neg xpe ; xme \land \mathrm{orm} ; \mathrm{otcme})$<br>$\lor (\neg xpe ; \neg xme \land \mathrm{orm} ; \mathrm{oda} ; \mathrm{tcm})$ | 10, RuleChop-<br>TwoRuleImp |
| 12 | $(\neg xpe ; xme \land \mathrm{orm} ; \mathrm{otcme})$<br>$\lor (\neg xpe ; \neg xme \land \mathrm{orm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\lor (xpe \land \mathrm{otcpe})$ | 11, $\lor$ introduction |
| 13 | $(xpe \land \mathrm{otcpe})$ | CP assumption |
| 14 | $(\neg xpe ; xme \land \mathrm{orm} ; \mathrm{otcme})$<br>$\lor (\neg xpe ; \neg xme \land \mathrm{orm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\lor (xpe \land \mathrm{otcpe})$ | 13, $\lor$ introduction<br>and comm. of $\lor$ |
| 15 | $(\neg xpe ; xme \land \mathrm{orm} ; \mathrm{otcme})$<br>$\lor (\neg xpe ; \neg xme \land \mathrm{orm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\lor (xpe \land \mathrm{otcpe})$ | 9, 10-12, 13-14,<br>$\lor$ elimination |
| 16 | $\neg rule_{7.2\text{-}e} \lor ((\neg xpe ; xme \land \mathrm{orm} ; \mathrm{otcme})$<br>$\lor (\neg xpe ; \neg xme \land \mathrm{orm} ; \mathrm{oda} ; \mathrm{tcm}) \lor (xpe \land \mathrm{otcpe}))$ | 15, $\lor$ introduction<br>and comm. of $\lor$ |
| 17 | $rule_{7.2\text{-}e} \supset ((\neg xpe ; xme \land \mathrm{orm} ; \mathrm{otcme})$<br>$\lor (\neg xpe ; \neg xme \land \mathrm{orm} ; \mathrm{oda} ; \mathrm{tcm}) \lor (xpe \land \mathrm{otcpe}))$ | 16, definition of $\supset$ |
| 18 | $rule_{7.2\text{-}d}$ | 3, reiteration |
| 19 | $(xmp \land \mathrm{odc} ; \mathrm{tdc}) \lor (\neg xmp \land \bigcirc rule_{7.2\text{-}e})$ | 3, 18, eqv. subst. |
| 20 | $(xmp \land \mathrm{odc} ; \mathrm{tdc}) \lor (\neg xmp \land \mathsf{skip} ; rule_{7.2\text{-}e})$ | 19, ITL (def. of $\bigcirc$) |
| 21 | $(xmp \land \mathrm{odc} ; \mathrm{tdc}) \lor ((\neg xmp \land \mathsf{skip}) ; rule_{7.2\text{-}e})$ | 20, ITL<br>(StateAndChop) |
| 22 | $(\neg xmp \land \mathsf{skip}) ; rule_{7.2\text{-}e}$ | CP assumption |
| 23 | $(\neg xmp \land \mathsf{skip}) ; ((\neg xpe ; xme \land \mathrm{orm} ; \mathrm{otcme})$<br>$\lor (\neg xpe ; \neg xme \land \mathrm{orm} ; \mathrm{oda} ; \mathrm{tcm}) \lor (xpe \land \mathrm{otcpe}))$ | 17, 22,<br>ChopSwapImp1 |

| | | |
|---|---|---|
| 24 | $(\neg xmp \wedge \mathrm{skip}) ; (\neg xpe ; xme \wedge \circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xmp \wedge \mathrm{skip}) ; (\neg xpe ; \neg xme \wedge \circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xmp \wedge \mathrm{skip}) ; (xpe \wedge \circ\mathrm{tcpe})$ | 23, ITL (ChopOr) |
| 25 | $(\neg xmp \wedge \mathrm{skip}) ; (\neg xpe ; xme \wedge \circ\mathrm{rm} ; \circ\mathrm{tcme})$ | CP assumption |
| 26 | $\neg xmp ; \neg xpe ; xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{tcme}$ | 25, TwoChop-RulesImp |
| 27 | $(\neg xmp ; \neg xpe ; xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xmp ; xpe \wedge \mathrm{skip} ; \circ\mathrm{tcpe})$ | 26, $\vee$ introduction |
| 28 | $(\neg xmp \wedge \mathrm{skip}) ; (\neg xpe ; \neg xme \wedge \circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$ | CP assumption |
| 29 | $\neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm}$ | 28, TwoChop-RulesImp |
| 30 | $(\neg xmp ; \neg xpe ; xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xmp ; xpe \wedge \mathrm{skip} ; \circ\mathrm{tcpe})$ | 29, $\vee$ introduction and comm. of $\vee$ |
| 31 | $(\neg xmp \wedge \mathrm{skip}) ; (xpe \wedge \circ\mathrm{tcpe})$ | CP assumption |
| 32 | $\neg xmp ; xpe \wedge \mathrm{skip} ; \circ\mathrm{tcpe}$ | 31, TwoChop-RulesImp |
| 33 | $(\neg xmp ; \neg xpe ; xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xmp ; xpe \wedge \mathrm{skip} ; \circ\mathrm{tcpe})$ | 32, $\vee$ introduction and comm. of $\vee$ |
| 34 | $(\neg xmp ; \neg xpe ; xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{skip} ; \circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xmp ; xpe \wedge \mathrm{skip} ; \circ\mathrm{tcpe})$ | 24, 25-27, 28-30, 31-33, $\vee$ elimination |
| 35 | $(\neg xmp ; \neg xpe ; xme \wedge \circ\circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xmp ; \neg xpe ; \neg xme \wedge \circ\circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xmp ; xpe \wedge \circ\circ\mathrm{tcpe})$ | 34, ITL (def. of $\circ$) |

| | | |
|---|---|---|
| 36 | (¬*xmp* ; ¬*xpe* ; *xme* ∧ oorm ; otcme) ∨ (¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ oorm ; oda ; tcm) ∨ (¬*xmp* ; *xpe* ∧ ootcpe) ∨ (*xmp* ∧ odc ; tdc) | 35, ∨ introduction |
| 37 | (*xmp* ∧ odc ; tdc) | CP assumption |
| 38 | (¬*xmp* ; ¬*xpe* ; *xme* ∧ oorm ; otcme) ∨ (¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ oorm ; oda ; tcm) ∨ (¬*xmp* ; *xpe* ∧ ootcpe) ∨ (*xmp* ∧ odc ; tdc) | 37, ∨ introduction and comm. of ∨ |
| 39 | (¬*xmp* ; ¬*xpe* ; *xme* ∧ oorm ; otcme) ∨ (¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ oorm ; oda ; tcm) ∨ (¬*xmp* ; *xpe* ∧ ootcpe) ∨ (*xmp* ∧ odc ; tdc) | 21, 22-36, 37-38, ∨ elimination |
| 40 | ¬$rule_{7.2\text{-}d}$ ∨ ((¬*xmp* ; ¬*xpe* ; *xme* ∧ oorm ; otcme) ∨ (¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ oorm ; oda ; tcm) ∨ (¬*xmp* ; *xpe* ∧ ootcpe) ∨ (*xmp* ∧ odc ; tdc)) | 39, ∨ introduction and comm. of ∨ |
| 41 | $rule_{7.2\text{-}d}$ ⊃ ((¬*xmp* ; ¬*xpe* ; *xme* ∧ oorm ; otcme) ∨ (¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ oorm ; oda ; tcm) ∨ (¬*xmp* ; *xpe* ∧ ootcpe) ∨ (*xmp* ∧ odc ; tdc)) | 40, definition of ⊃ |
| 42 | $rule_{7.2\text{-}c}$ | 2, reiteration |
| 43 | (*xcd* ∧ otdc) ∨ (¬*xcd* ∧ ogp ; $rule_{7.2\text{-}d}$) | 2, 42, eqv. subst. |
| 44 | (*xcd* ∧ otdc) ∨ ((¬*xcd* ∧ ogp) ; $rule_{7.2\text{-}d}$) | 43, ITL (StateAndChop) |
| 45 | (¬*xcd* ∧ ogp) ; $rule_{7.2\text{-}d}$ | CP assumption |
| 46 | (¬*xcd* ∧ ogp) ; ((¬*xmp* ; ¬*xpe* ; *xme* ∧ oorm ; otcme) ∨ (¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ oorm ; oda ; tcm) ∨ (¬*xmp* ; *xpe* ∧ ootcpe) ∨ (*xmp* ∧ odc ; tdc)) | 41, 45, ChopSwapImp1 |

| | | |
|---|---|---|
| 47 | $(\neg xcd \wedge \circ\mathrm{gp}) ; (\neg xmp ; \neg xpe ; xme \wedge \circ\circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xcd \wedge \circ\mathrm{gp}) ; (\neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \circ\circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xcd \wedge \circ\mathrm{gp}) ; (\neg xmp ; xpe \wedge \circ\circ\mathrm{tcpe})$<br>$\vee (\neg xcd \wedge \circ\mathrm{gp}) ; (xmp \wedge \circ\mathrm{dc} ; \mathrm{tdc})$ | 46, ITL (ChopOr) |
| 48 | $(\neg xcd \wedge \circ\mathrm{gp}) ; (\neg xmp ; \neg xpe ; xme \wedge \circ\circ\mathrm{rm} ; \circ\mathrm{tcme})$ | CP assumption |
| 49 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme) \wedge (\circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{tcme})$ | 48, TwoChop-RulesImp |
| 50 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \circ\mathrm{gp} ; \circ\circ\mathrm{tcpe})$<br>$\vee (\neg xcd ; xmp \wedge \circ\mathrm{gp} ; \circ\mathrm{dc} ; \mathrm{tdc})$ | 49, $\vee$ introduction and comm. of $\vee$ |
| 51 | $(\neg xcd \wedge \circ\mathrm{gp}) ; (\neg xmp ; \neg xpe ; \neg xme \wedge \circ\circ\mathrm{rm} ; \circ\mathrm{da} ;$ $\mathrm{tcm})$ | CP assumption |
| 52 | $(\neg xcd ; \neg xmp ; \neg xpe ; \neg xme)$<br>$\wedge (\circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$ | 51, TwoChop-RulesImp |
| 53 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \circ\mathrm{gp} ; \circ\circ\mathrm{tcpe})$<br>$\vee (\neg xcd ; xmp \wedge \circ\mathrm{gp} ; \circ\mathrm{dc} ; \mathrm{tdc})$ | 52, $\vee$ introduction and comm. of $\vee$ |
| 54 | $(\neg xcd \wedge \circ\mathrm{gp}) ; (\neg xmp ; xpe \wedge \circ\circ\mathrm{tcpe})$ | CP assumption |
| 55 | $(\neg xcd ; \neg xmp ; xpe) \wedge (\circ\mathrm{gp} ; \circ\circ\mathrm{tcpe})$ | 54, TwoChop-RulesImp |
| 56 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{tcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \circ\mathrm{gp} ; \circ\circ\mathrm{rm} ; \circ\mathrm{da} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \circ\mathrm{gp} ; \circ\circ\mathrm{tcpe})$<br>$\vee (\neg xcd ; xmp \wedge \circ\mathrm{gp} ; \circ\mathrm{dc} ; \mathrm{tdc})$ | 55, $\vee$ introduction and comm. of $\vee$ |
| 57 | $(\neg xcd \wedge \circ\mathrm{gp}) ; (xmp \wedge \circ\mathrm{dc} ; \mathrm{tdc})$ | CP assumption |

| | | |
|---|---|---|
| 58 | $(\neg xcd ; xmp) \wedge (\mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$ | 57, TwoChop-RulesImp |
| 59 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$<br>$\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$ | 58, $\vee$ introduction and comm. of $\vee$ |
| 60 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$<br>$\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$ | 47, 48-50, 51-53, 54-56, 57-59, $\vee$ elimination |
| 61 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$<br>$\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$<br>$\vee (xcd \wedge \mathrm{otdc})$ | 60, $\vee$ introduction |
| 62 | $(xcd \wedge \mathrm{otdc})$ | CP assumption |
| 63 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$<br>$\wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$<br>$\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$<br>$\vee (xcd \wedge \mathrm{otdc})$ | 62, $\vee$ introduction and comm. of $\vee$ |
| 64 | $(\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$<br>$\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$<br>$\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$<br>$\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$<br>$\vee (xcd \wedge \mathrm{otdc})$ | 44, 45-61, 62-63, $\vee$ elimination |

| | | |
|---|---|---|
| 65 | $\neg rule_{7.2\text{-}c}$ $((\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$ $\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$ $\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$ $\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$ $\vee (xcd \wedge \mathrm{otdc}))$ | 64, $\vee$ introduction and comm. of $\vee$ |
| 66 | $rule_{7.2\text{-}c} \supset$ $((\neg xcd ; \neg xmp ; \neg xpe ; xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$ $\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme \wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$ $\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$ $\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$ $\vee (xcd \wedge \mathrm{otdc}))$ | 65, definition of $\supset$ |
| 67 | $rule_{7.2\text{-}b}$ | 1, reiteration |
| 68 | $((xane \wedge \mathrm{owc} ; \mathrm{rc} ; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b}) \vee (\neg xane \wedge \mathrm{empty})$ | 1, 67, eqv. subst. |
| 69 | $((xane \wedge \mathrm{owc} ; \mathrm{rc} ; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b})$ | CP assumption |
| 70 | $(((xane \wedge \mathrm{owc} ; \mathrm{rc}) ; rule_{7.2\text{-}c}) ; rule_{7.2\text{-}b})$ | 69, ITL (StateAndChop) |
| 71 | $(xane \wedge \mathrm{owc} ; \mathrm{rc}) ; rule_{7.2\text{-}c} ; rule_{7.2\text{-}b}$ | 70, ChopAssoc |
| 72 | $(xane \wedge \mathrm{owc} ; \mathrm{rc}) ; ((\neg xcd ; \neg xmp ; \neg xpe ; xme$ $\wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{otcme})$ $\vee (\neg xcd ; \neg xmp ; \neg xpe ; \neg xme$ $\wedge \mathrm{ogp} ; \mathrm{oorm} ; \mathrm{oda} ; \mathrm{tcm})$ $\vee (\neg xcd ; \neg xmp ; xpe \wedge \mathrm{ogp} ; \mathrm{ootcpe})$ $\vee (\neg xcd ; xmp \wedge \mathrm{ogp} ; \mathrm{odc} ; \mathrm{tdc})$ $\vee (xcd \wedge \mathrm{otdc})) ; rule_{7.2\text{-}b}$ | 66, 71, ChopSwapImp3 |

| | | |
|---|---|---|
| 73 | ((*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme* ∧ ogp ; oorm ; otcme) ∨ (*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ ogp ; oorm ; oda ; tcm) ∨ (*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; *xpe* ∧ ogp ; ootcpe) ∨ (*xane* ∧ owc ; rc) ; (¬*xcd* ; *xmp* ∧ ogp ; odc ; tdc) ∨ (*xane* ∧ owc ; rc) ; (*xcd* ∧ otdc)) ; $\mathit{rule}_{7.2\text{-}b}$ | 72, ITL (ChopOr) |
| 74 | (*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme* ∧ ogp ; oorm ; otcme) ; $\mathit{rule}_{7.2\text{-}b}$ ∨ (*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ ogp ; oorm ; oda ; tcm) ; $\mathit{rule}_{7.2\text{-}b}$ ∨ (*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; *xpe* ∧ ogp ; ootcpe) ; $\mathit{rule}_{7.2\text{-}b}$ ∨ (*xane* ∧ owc ; rc) ; (¬*xcd* ; *xmp* ∧ ogp ; odc ; tdc) ; $\mathit{rule}_{7.2\text{-}b}$ ∨ (*xane* ∧ owc ; rc) ; (*xcd* ∧ otdc) ; $\mathit{rule}_{7.2\text{-}b}$ | 73, ITL (OrChop) |
| 75 | (*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme* ∧ ogp ; oorm ; otcme) ; $\mathit{rule}_{7.2\text{-}b}$ | CP assumption |
| 76 | (*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme*) ∧ (owc ; rc ; ogp ; oorm ; otcme ; $\mathit{rule}_{7.2\text{-}b}$) | 75, TwoChop-RulesImp2 |
| 77 | ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme*) ∧ (owc ; rc ; ogp ; oorm ; otcme ; $\mathit{rule}_{7.2\text{-}b}$)) ∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; ¬*xme*) ∧ (owc ; rc ; ogp ; oorm ; oda ; tcm ; $\mathit{rule}_{7.2\text{-}b}$)) ∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; *xpe*) ∧ (owc ; rc ; ogp ; ootcpe ; $\mathit{rule}_{7.2\text{-}b}$)) ∨ ((*xane* ; ¬*xcd* ; *xmp*) ∧ (owc ; rc ; ogp ; odc ; tdc; $\mathit{rule}_{7.2\text{-}b}$)) ∨ ((*xane* ; *xcd*) ∧ (owc ; rc ; otdc ; $\mathit{rule}_{7.2\text{-}b}$)) | 76, ∨ introduction |
| 78 | (*xane* ∧ owc ; rc) ; (¬*xcd* ; ¬*xmp* ; ¬*xpe* ; ¬*xme* ∧ ogp ; oorm ; oda ; tcm) ; $\mathit{rule}_{7.2\text{-}b}$ | CP assumption |

| | | |
|---|---|---|
| 79 | $(xane ; \neg xcd ; \neg xmp ; \neg xpe ; \neg xme)$ $\wedge$ (owc ; rc ; ogp ; oorm ; oda ; tcm ; $rule_{7.2\text{-}b}$) | 78, TwoChop-RulesImp2 |
| 80 | $((xane ; \neg xcd ; \neg xmp ; \neg xpe ; xme)$ $\wedge$ (owc ; rc ; ogp ; oorm ; otcme ; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; \neg xcd ; \neg xmp ; \neg xpe ; \neg xme)$ $\wedge$ (owc ; rc ; ogp ; oorm ; oda ; tcm ; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; \neg xcd ; \neg xmp ; xpe)$ $\wedge$ (owc ; rc ; ogp ; ootcpe ; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; \neg xcd ; xmp)$ $\wedge$ (owc ; rc ; ogp ; odc ; tdc; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; xcd)$ $\wedge$ (owc ; rc ; otdc ; $rule_{7.2\text{-}b}$)) | 79, $\vee$ introduction and comm. of $\vee$ |
| 81 | $(xane$ $\wedge$ owc ; rc) ; $(\neg xcd ; \neg xmp ; xpe$ $\wedge$ ogp ; ootcpe) ; $rule_{7.2\text{-}b}$ | CP assumption |
| 82 | $(xane ; \neg xcd ; \neg xmp ; xpe)$ $\wedge$ (owc ; rc ; ogp ; ootcpe ; $rule_{7.2\text{-}b}$) | 81, TwoChop-RulesImp2 |
| 83 | $((xane ; \neg xcd ; \neg xmp ; \neg xpe ; xme)$ $\wedge$ (owc ; rc ; ogp ; oorm ; otcme ; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; \neg xcd ; \neg xmp ; \neg xpe ; \neg xme)$ $\wedge$ (owc ; rc ; ogp ; oorm ; oda ; tcm ; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; \neg xcd ; \neg xmp ; xpe)$ $\wedge$ (owc ; rc ; ogp ; ootcpe ; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; \neg xcd ; xmp)$ $\wedge$ (owc ; rc ; ogp ; odc ; tdc; $rule_{7.2\text{-}b}$)) $\vee$ $((xane ; xcd)$ $\wedge$ (owc ; rc ; otdc ; $rule_{7.2\text{-}b}$)) | 82, $\vee$ introduction and comm. of $\vee$ |
| 84 | $(xane$ $\wedge$ owc ; rc) ; $(\neg xcd ; xmp$ $\wedge$ ogp ; odc ; tdc) ; $rule_{7.2\text{-}b}$ | CP assumption |
| 85 | $(xane ; \neg xcd ; xmp)$ $\wedge$ (owc ; rc ; ogp ; odc ; tdc; $rule_{7.2\text{-}b}$) | 84, TwoChop-RulesImp2 |

| | | |
|---|---|---|
| 86 | $((xane ; \neg xcd ; \neg xmp ; \neg xpe ; xme) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{oorm} ; \text{otcme} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; \neg xmp ; \neg xpe ; \neg xme) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{oorm} ; \text{oda} ; \text{tcm} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; \neg xmp ; xpe) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{ootcpe} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; xmp) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{odc} ; \text{tdc}; rule_{7.2\text{-}b})) \lor ((xane ; xcd) \land (\text{owc} ; \text{rc} ; \text{otdc} ; rule_{7.2\text{-}b}))$ | 85, ∨ introduction and comm. of ∨ |
| 87 | $(xane \land \text{owc} ; \text{rc}) ; (xcd \land \text{otdc}) ; rule_{7.2\text{-}b}$ | CP assumption |
| 88 | $(xane ; xcd) \land (\text{owc} ; \text{rc} ; \text{otdc} ; rule_{7.2\text{-}b})$ | 87, TwoChop-RulesImp2 |
| 89 | $((xane ; \neg xcd ; \neg xmp ; \neg xpe ; xme) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{oorm} ; \text{otcme} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; \neg xmp ; \neg xpe ; \neg xme) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{oorm} ; \text{oda} ; \text{tcm} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; \neg xmp ; xpe) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{ootcpe} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; xmp) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{odc} ; \text{tdc}; rule_{7.2\text{-}b})) \lor ((xane ; xcd) \land (\text{owc} ; \text{rc} ; \text{otdc} ; rule_{7.2\text{-}b}))$ | 88, ∨ introduction and comm. of ∨ |
| 90 | $((xane ; \neg xcd ; \neg xmp ; \neg xpe ; xme) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{oorm} ; \text{otcme} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; \neg xmp ; \neg xpe ; \neg xme) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{oorm} ; \text{oda} ; \text{tcm} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; \neg xmp ; xpe) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{ootcpe} ; rule_{7.2\text{-}b})) \lor ((xane ; \neg xcd ; xmp) \land (\text{owc} ; \text{rc} ; \text{ogp} ; \text{odc} ; \text{tdc}; rule_{7.2\text{-}b})) \lor ((xane ; xcd) \land (\text{owc} ; \text{rc} ; \text{otdc} ; rule_{7.2\text{-}b}))$ | 74, 75-77, 78-80, 81-83, 84-86, 87-89, ∨ elimination |

| | | |
|---|---|---|
| 91 | ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme*)<br>∧ (owc ; rc ; ogp ; oorm ; otcme ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; ¬*xme*)<br>∧ (owc ; rc ; ogp ; oorm ; oda ; tcm ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; *xpe*)<br>∧ (owc ; rc ; ogp ; ootcpe ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; *xmp*)<br>∧ (owc ; rc ; ogp ; odc ; tdc; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; *xcd*) ∧ (owc ; rc ; otdc ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ (¬*xane* ∧ empty) | ∨ introduction |
| 92 | (¬*xane* ∧ empty) | CP assumption |
| 93 | ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme*)<br>∧ (owc ; rc ; ogp ; oorm ; otcme ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; ¬*xme*)<br>∧ (owc ; rc ; ogp ; oorm ; oda ; tcm ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; *xpe*)<br>∧ (owc ; rc ; ogp ; ootcpe ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; *xmp*)<br>∧ (owc ; rc ; ogp ; odc ; tdc; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; *xcd*) ∧ (owc ; rc ; otdc ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ (¬*xane* ∧ empty) | 92, ∨ introduction and comm. of ∨ |
| 94 | ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; *xme*)<br>∧ (owc ; rc ; ogp ; oorm ; otcme ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; ¬*xpe* ; ¬*xme*)<br>∧ (owc ; rc ; ogp ; oorm ; oda ; tcm ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; ¬*xmp* ; *xpe*)<br>∧ (owc ; rc ; ogp ; ootcpe ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; ¬*xcd* ; *xmp*)<br>∧ (owc ; rc ; ogp ; odc ; tdc; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ ((*xane* ; *xcd*) ∧ (owc ; rc ; otdc ; $\mathit{rule}_{7.2\text{-}b}$))<br>∨ (¬*xane* ∧ empty) | 68, 69-91, 92-93, ∨ elimination |

Based on these transformations, the following observations are made. By definition, $rule_{7.2\text{-}f}$ describes the behaviors associated with $rule_{7.2\text{-}f}$ and no transformation is required. The two possible behaviors associated with $rule_{7.2\text{-}f}$ are:

$(\mathit{money\_exit} \wedge \circ\text{take\_card\_money\_exit})$

$\vee\ (\neg \mathit{money\_exit} \wedge \circ\text{debit\_account} ; \text{take\_card\_money})$

The transformation of $rule_{7.2\text{-}e}$ is complete at sequent 15 and incorporates the behaviors associated with $rule_{7.2\text{-}f}$. The three possible behaviors associated with $rule_{7.2\text{-}e}$ are:

$(\neg \mathit{pin\_exit} ; \mathit{money\_exit} \wedge \circ\text{request\_money} ; \circ\text{take\_card\_money\_exit})$

$\vee\ (\neg \mathit{pin\_exit} ; \neg \mathit{money\_exit} \wedge$
$\circ\text{request\_money} ; \circ\text{debit\_account} ; \text{take\_card\_money})$

$\vee\ (\mathit{pin\_exit} \wedge \circ\text{take\_card\_pin\_exit})$

The transformation of $rule_{7.2\text{-}d}$ is complete at sequent 39 and incorporates the behaviors associated with $rule_{7.2\text{-}e}$ and $rule_{7.2\text{-}f}$. The four possible behaviors associated with $rule_{7.2\text{-}e}$ are:

$(\neg \mathit{max\_pin} ; \neg \mathit{pin\_exit} ; \mathit{money\_exit}$
$\wedge\ \circ\circ\text{request\_money} ; \circ\text{take\_card\_money\_exit})$

$\vee\ (\neg \mathit{max\_pin} ; \neg \mathit{pin\_exit} ; \neg \mathit{money\_exit}$
$\wedge\ \circ\circ\text{request\_money} ; \circ\text{debit\_account} ; \text{take\_card\_money})$

$\vee\ (\neg \mathit{max\_pin} ; \mathit{pin\_exit} \wedge \circ\circ\text{take\_card\_pin\_exit})$

$\vee\ (\mathit{max\_pin} \wedge \circ\text{disable\_card} ; \text{take\_disabled\_card})$

The transformation of $rule_{7.2\text{-}c}$ is complete at sequent 64 and incorporates the behaviors associated with $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$. The five possible behaviors associated with $rule_{7.2\text{-}c}$ are:

$(\neg \mathit{card\_disabled} ; \neg \mathit{max\_pin} ; \neg \mathit{pin\_exit} ; \mathit{money\_exit}$
$\wedge\ \circ\text{get\_pin} ; \circ\circ\text{request\_money} ; \circ\text{take\_card\_money\_exit})$

$\vee\ (\neg \mathit{card\_disabled} ; \neg \mathit{max\_pin} ; \neg \mathit{pin\_exit} ; \neg \mathit{money\_exit}$
$\wedge\ \circ\text{get\_pin} ; \circ\circ\text{request\_money} ;$

○debit_account ; take_card_money)

∨ (¬*card_disabled* ; ¬*max_pin* ; *pin_exit*
  ∧ ○get_pin ; ○○take_card_pin_exit)

∨ (¬*card_disabled* ; *max_pin*
  ∧ ○get_pin ; ○disable_card ; take_disabled_card)

∨ (*card_disabled* ∧ ○take_disabled_card)

The transformation of $rule_{7.2\text{-}b}$ is complete at sequent 94 and incorporates the behaviors associated with $rule_{7.2\text{-}c}$, $rule_{7.2\text{-}d}$, $rule_{7.2\text{-}e}$, and $rule_{7.2\text{-}f}$. The six possible behaviors associated with $rule_{7.2\text{-}b}$ are:

(*atm_non_empty* ; ¬*card_disabled* ; ¬*max_pin* ; ¬*pin_exit* ; *money_exit*
  ∧ ○wait_customer ; read_card ; ○get_pin ; ○○request_money ;
    ○take_card_money_exit ; $rule_{7.2\text{-}b}$)

∨ (*atm_non_empty* ; ¬*card_disabled* ; ¬*max_pin* ; ¬*pin_exit* ; ¬*money_exit*
  ∧ ○wait_customer ; read_card ; ○get_pin ; ○○request_money ;
    ○debit_account ; take_card_money ; $rule_{7.2\text{-}b}$)

∨ (*atm_non_empty* ; ¬*card_disabled* ; ¬*max_pin* ; *pin_exit*
  ∧ ○wait_customer ; read_card ; ○get_pin ;
    ○○take_card_pin_exit ; $rule_{7.2\text{-}b}$)

∨ (*atm_non_empty* ; ¬*card_disabled* ; *max_pin*
  ∧ ○wait_customer ; read_card ; ○get_pin ;
    ○disable_card ; take_disabled_card ; $rule_{7.2\text{-}b}$)

∨ (*atm_non_empty* ; *card_disabled*
  ∧ ○wait_customer ; read_card ; take_disabled_card ; $rule_{7.2\text{-}b}$)

∨ (¬*atm_non_empty* ∧ empty)

# Appendix C

## Formal Transformation of I/O Rules in Legacy Code

In this formal transformation, the specific rules derived from the legacy code presented in Section 8.1 are transformed to create a single rule structure. The focus of this transformation are the rules associated with specific I/O activities. Because *rule*$_4$ does not include any I/O activities, *rule*$_4$ is not considered in this transformation.

This transformation rests on seven premises that reflect the rules extracted from legacy code that directly or indirectly include the variable $I/O_{write}$ – $f_0$, *rule*$_1$, *rule*$_{1'}$, *rule*$_2$, *rule*$_3$, *rule*$_5$, and *rule*$_{5'}$. Because the deepest rule, *rule*$_5$ (including the subrule *rule*$_{5'}$) includes no other rules and therefore, by definition, totally describes all behaviors associated with *rule*$_5$, *rule*$_5$ needs no transformation. Therefore, *rule*$_3$ is transformed first and incorporates the behaviors associated with *rule*$_5$. Then, *rule*$_2$ is transformed and incorporates the behaviors derived from *rule*$_3$ and *rule*$_5$. Then, *rule*$_1$ (including the subrule *rule*$_{1'}$) is transformed and incorporates the behaviors derived from *rule*$_2$, *rule*$_3$, and *rule*$_5$. Finally, $f_0$ is transformed and incorporates the behaviors derived from *rule*$_1$, *rule*$_2$, *rule*$_3$, and *rule*$_5$.

| | | |
|---|---|---|
| 1 | $f_0$<br>where: $f_0 \triangleq f_{0a} ; f_{pl} ; f_{0b} ; rule_1 ; f_{0c} ; f_{0d} ; f_{0e}$ | premise |
| 2 | $rule_1$<br>where: $rule_1 \triangleq f_{1a} ; rule_{1'}$ | premise |
| 3 | $rule_{1'}$<br>where: $rule_{1'} \triangleq (w_{C1'} \wedge \circ rule_2 ; f_{1b} ; rule_{1'})$<br>$\vee (\neg w_{C1'} \wedge \text{empty})$ | premise |
| 4 | $rule_2$<br>where: $rule_2 \triangleq (w_{C2} \wedge \circ f_{3a} ; f_{3b} ; rule_3 ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})$<br>$\vee (\neg w_{C2} \wedge \text{empty})$ | premise |
| 5 | $rule_3$<br>where: $rule_3 \triangleq (w_{C3} \wedge \circ rule_5 ; f_{4a} ; f_{4b})$<br>$\vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d})$ | premise |

| | | |
|---|---|---|
| 6 | $rules_5$<br>where: $rules_5 \triangleq f_{6a} \,; rules_{5'}$ | premise |
| 7 | $rules_{5'}$<br>where: $rules_{5'} \triangleq (w_{C5'} \wedge \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'})$<br>$\vee (\neg w_{C5'} \wedge \mathsf{empty})$ | premise |
| 8 | $rule_3$ | 5, reiteration |
| 9 | $(w_{C3} \wedge \bigcirc rules_5 \,; f_{4a} \,; f_{4b}) \vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | 5, 8, def. subst. |
| 10 | $rules_5 \equiv f_{6a} \,; rules_{5'}$ | 6, reiteration |
| 11 | $(w_{C3} \wedge (\bigcirc f_{6a} \,; rules_{5'}) \,; f_{4a} \,; f_{4b}) \vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | 9, 10, eqv. subst. |
| 12 | $((w_{C3} \wedge \bigcirc f_{6a}) \,; rules_{5'} \,; f_{4a} \,; f_{4b}) \vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | StateAndChop |
| 13 | $rules_{5'} \equiv (w_{C5'} \wedge \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'}) \vee (\neg w_{C5'} \wedge \mathsf{empty})$ | 7, reiteration |
| 14 | $((w_{C3} \wedge \bigcirc f_{6a}) \,; ((w_{C5'} \wedge \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'})$<br>$\vee (\neg w_{C5'} \wedge \mathsf{empty})) \,; f_{4a} \,; f_{4b}) \vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | 12, 13, def. subst. |
| 15 | $((w_{C3} \wedge \bigcirc f_{6a}) \,; ((w_{C5'} \wedge \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'}) \,; f_{4a} \,; f_{4b}$<br>$\vee (\neg w_{C5'} \wedge \mathsf{empty}) \,; f_{4a} \,; f_{4b})) \vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | 14, OrChop |
| 16 | $((w_{C3} \wedge \bigcirc f_{6a}) \,; ((w_{C5'} \wedge \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'} \,; f_{4a} \,; f_{4b})$<br>$\vee (\neg w_{C5'} \wedge \mathsf{empty} \,; f_{4a} \,; f_{4b}))) \vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | 15, StateAndChop |
| 17 | $((w_{C3} \wedge \bigcirc f_{6a}) \,; ((w_{C5'} \wedge \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'} \,; f_{4a} \,; f_{4b})$<br>$\vee (\neg w_{C5'} \wedge f_{4a} \,; f_{4b}))) \vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | 16, EmptyChop |
| 18 | $(w_{C3} \wedge \bigcirc f_{6a}) \,; ((w_{C5'} \wedge \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'} \,; f_{4a} \,; f_{4b})$<br>$\vee (\neg w_{C5'} \wedge f_{4a} \,; f_{4b}))$ | CP assumption |
| 19 | $((w_{C3} \,; w_{C5'}) \wedge (\bigcirc f_{6a} \,; \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'} \,; f_{4a} \,; f_{4b}))$<br>$\vee ((w_{C3} \,; \neg w_{C5'}) \wedge (\bigcirc f_{6a} \,; f_{4a} \,; f_{4b}))$ | 18, RuleChop-<br>TwoRuleImp |
| 20 | $((w_{C3} \,; w_{C5'}) \wedge (\bigcirc f_{6a} \,; \bigcirc f_{6b} \,; f_{6c} \,; rules_{5'} \,; f_{4a} \,; f_{4b}))$<br>$\vee ((w_{C3} \,; \neg w_{C5'}) \wedge (\bigcirc f_{6a} \,; f_{4a} \,; f_{4b}))$<br>$\vee (\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | 19, $\vee$ introduction |
| 21 | $(\neg w_{C3} \wedge \bigcirc f_{4c} \,; f_{4d})$ | CP assumption |

| | | |
|---|---|---|
| 22 | $((w_{C3} ; w_{C5'}) \wedge (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) \vee ((w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{6a} ; f_{4a} ; f_{4b})) \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d})$ | 21, $\vee$ introduction and comm. of $\vee$ |
| 23 | $((w_{C3} ; w_{C5'}) \wedge (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) \vee ((w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{6a} ; f_{4a} ; f_{4b})) \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d})$ | 18-20, 21-22, $\vee$ elimination |
| 24 | $((w_{C3} ; w_{C5'}) \wedge (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) \vee ((w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{6a} ; f_{4a} ; f_{4b})) \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d}) \vee \neg rule_3$ | 23, $\vee$ introduction |
| 25 | $rule_3 \supset (((w_{C3} ; w_{C5'}) \wedge (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) \vee ((w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{6a} ; f_{4a} ; f_{4b})) \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d}))$ | 24, comm. of $\vee$ and definition of $\supset$ |
| 26 | $rule_2$ | 4, reiteration |
| 27 | $(w_{C2} \wedge \circ f_{3a} ; f_{3b} ; rule_3 ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}) \vee (\neg w_{C2} \wedge \mathrm{empty})$ | 4, 26, def. subst. |
| 28 | $((w_{C2} \wedge \circ f_{3a} ; f_{3b}) ; rule_3 ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}) \vee (\neg w_{C2} \wedge \mathrm{empty})$ | 27, StateAndChop |
| 29 | $(w_{C2} \wedge \circ f_{3a} ; f_{3b}) ; rule_3 ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}$ | CP assumption |
| 30 | $(w_{C2} \wedge \circ f_{3a} ; f_{3b}) ; (((w_{C3} ; w_{C5'}) \wedge (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) \vee ((w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{6a} ; f_{4a} ; f_{4b})) \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}$ | 29, ChopSwapImp3 |
| 31 | $(w_{C2} \wedge \circ f_{3a} ; f_{3b}) ; (((w_{C3} ; w_{C5'}) \wedge (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \vee ((w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{6a} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d}) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} )$ | 30, OrChop |
| 32 | $(w_{C2} \wedge \circ f_{3a} ; f_{3b}) ; (((w_{C3} ; w_{C5'}) \wedge (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \vee ((w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{6a} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \vee (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d}) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ) \vee (\neg w_{C2} \wedge \mathrm{empty})$ | 31, $\vee$ introduction |

| | | |
|---|---|---|
| 33 | $(\neg w_{C2} \land \text{empty})$ | CP assumption |
| 34 | $(w_{C2} \land \circ f_{3a} ; f_{3b}) ; (((w_{C3} ; w_{C5'}) \land (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \lor ((w_{C3} ; \neg w_{C5'}) \land (\circ f_{6a} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \lor (\neg w_{C3} \land \circ f_{4c} ; f_{4d}) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}) \lor (\neg w_{C2} \land \text{empty})$ | 33, $\lor$ introduction and comm. of $\lor$ |
| 35 | $(w_{C2} \land \circ f_{3a} ; f_{3b}) ; (((w_{C3} ; w_{C5'}) \land (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \lor ((w_{C3} ; \neg w_{C5'}) \land (\circ f_{6a} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \lor (\neg w_{C3} \land \circ f_{4c} ; f_{4d}) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}) \lor (\neg w_{C2} \land \text{empty})$ | 34, $\lor$ elimination |
| 36 | $(w_{C2} \land \circ f_{3a} ; f_{3b}) ; ((w_{C3} ; w_{C5'}) \land (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \lor (w_{C2} \land \circ f_{3a} ; f_{3b}) ; ((w_{C3} ; \neg w_{C5'}) \land (\circ f_{6a} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \lor (w_{C2} \land \circ f_{3a} ; f_{3b}) ; (\neg w_{C3} \land \circ f_{4c} ; f_{4d}) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} \lor (\neg w_{C2} \land \text{empty})$ | 35, ChopOr |
| 37 | $(w_{C2} \land \circ f_{3a} ; f_{3b}) ; ((w_{C3} ; w_{C5'}) \land (\circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}$ | CP assumption |
| 38 | $(w_{C2} ; w_{C3} ; w_{C5'}) \land (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})$ | 37, TwoChop-RulesImp2 |
| 39 | $((w_{C2} ; w_{C3} ; w_{C5'}) \land (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \lor ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \land (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \lor ((w_{C2} ; \neg w_{C3}) \land (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \lor (\neg w_{C2} \land \text{empty})$ | 38, $\lor$ introduction |
| 40 | $(w_{C2} \land \circ f_{3a} ; f_{3b}) ; ((w_{C3} ; \neg w_{C5'}) \land (\circ f_{6a} ; f_{4a} ; f_{4b})) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}$ | CP assumption |
| 41 | $(w_{C2} ; w_{C3} ; \neg w_{C5'}) \land (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})$ | 40, TwoChop-RulesImp2 |

| | | |
|---|---|---|
| 42 | $((w_{C2} ; w_{C3} ; w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \vee ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \vee ((w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee (\neg w_{C2} \wedge \text{empty})$ | 41, $\vee$ introduction and comm. of $\vee$ |
| 43 | $(w_{C2} \wedge \circ f_{3a} ; f_{3b}) ; (\neg w_{C3} \wedge \circ f_{4c} ; f_{4d}) ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}$ | CP assumption |
| 44 | $(w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})$ | 43, TwoChop-RulesImp2 |
| 45 | $((w_{C2} ; w_{C3} ; w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \vee ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \vee ((w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee (\neg w_{C2} \wedge \text{empty})$ | 44, $\vee$ introduction and comm. of $\vee$ |
| 46 | $(\neg w_{C2} \wedge \text{empty})$ | CP assumption |
| 47 | $((w_{C2} ; w_{C3} ; w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \vee ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) \vee ((w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee (\neg w_{C2} \wedge \text{empty})$ | 46, $\vee$ introduction and comm. of $\vee$ |
| 48 | $((w_{C2} ; w_{C3} ; w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee ((w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee (\neg w_{C2} \wedge \text{empty})$ | 47, $\vee$ elimination |
| 49 | $((w_{C2} ; w_{C3} ; w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rules_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee ((w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee (\neg w_{C2} \wedge \text{empty})$<br>$\vee \neg rule_2$ | 48, $\vee$ introduction |

| | | |
|---|---|---|
| 50 | $rule_2 \supset (((w_{C2} ; w_{C3} ; w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$ $\vee ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$ $\vee ((w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$ $\vee (\neg w_{C2} \wedge \text{empty}))$ | 49, comm. of $\vee$ and definition of $\supset$ |
| 51 | $f_0$ | 1, reiteration |
| 52 | $f_{0a} ; f_{pl} ; f_{0b} ; rule_1 ; f_{0c} ; f_{0d} ; f_{0e}$ | 1, 51, def. subst. |
| 53 | $rule_1 \equiv f_{1a} ; rule_{1'}$ | 2, reiteration |
| 54 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | 2, 52, def. subst. |
| 55 | $rule_{1'} \equiv (w_{C1'} \wedge \circ rule_2 ; f_{1b} ; rule_{1'}) \vee (\neg w_{C1'} \wedge \text{empty})$ | 3, reiteration |
| 56 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; ((w_{C1'} \wedge \circ rule_2 ; f_{1b} ; rule_{1'})$ $\vee (\neg w_{C1'} \wedge \text{empty})) ; f_{0c} ; f_{0d} ; f_{0e}$ | 54, 55, eqv. subst. |
| 57 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (((w_{C1'} \wedge \circ rule_2) ; f_{1b} ; rule_{1'})$ $\vee (\neg w_{C1'} \wedge \text{empty})) ; f_{0c} ; f_{0d} ; f_{0e}$ | 56, StateAndChop |
| 58 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (((w_{C1'} \wedge \circ rule_2) ; f_{1b} ; rule_{1'}) ; f_{0c} ; f_{0d} ; f_{0e}$ $\vee (\neg w_{C1'} \wedge \text{empty}) ; f_{0c} ; f_{0d} ; f_{0e})$ | 57, OrChop |
| 59 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (((w_{C1'} \wedge \circ rule_2) ; f_{1b} ; rule_{1'}) ; f_{0c} ; f_{0d} ; f_{0e}$ $\vee (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e}))$ | 58, StateAndEmpty-Chop |
| 60 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (((w_{C1'} \wedge \circ rule_2) ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ $\vee (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e}))$ | 59, ChopAssoc |
| 61 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (((w_{C1'} \wedge \text{skip} ; rule_2) ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d}$ $; f_{0e}) \vee (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e}))$ | 60, ITL definition of $\circ$ |
| 62 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (((w_{C1'} \wedge \text{skip}) ; rule_2 ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d}$ $; f_{0e}) \vee (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e}))$ | 61, StateAndChop |
| 63 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; ((w_{C1'} \wedge \text{skip}) ; rule_2 ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ;$ $f_{0e}) \vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e})$ | 62, ChopOr |
| 64 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; rule_2 ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ;$ $f_{0e} \vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e})$ | 63, ChopAssoc |

|  |  |  |
|---|---|---|
| 65 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; rule_2 ; f_{1b} ; rule_{1'} ; f_{0c} ;$<br>$f_{0d} ; f_{0e}$ | CP assumption |
| 66 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ;$<br>$(((w_{C2} ; w_{C3} ; w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ;$<br>$f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee ((w_{C2} ; w_{C3} ; \neg w_{C5'}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ;$<br>$f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee ((w_{C2} ; \neg w_{C3}) \wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e}))$<br>$\vee (\neg w_{C2} \wedge \text{empty})) ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | 65, ChopSwapImp3 |
| 67 | $(f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; w_{C3} ; w_{C5'})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e}))$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; w_{C3} ; \neg w_{C5'})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; \neg w_{C3})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e}))$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; (\neg w_{C2} \wedge \text{empty})) ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | 66, ChopOr |
| 68 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; w_{C3} ; w_{C5'})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e})) ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; w_{C3} ; \neg w_{C5'})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; \neg w_{C3})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) ; f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e}$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; (\neg w_{C2} \wedge \text{empty}) ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | 67, OrChop |
| 69 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; w_{C3} ; w_{C5'})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e})) ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | CP assumption |

| | | |
|---|---|---|
| 70 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\land \text{skip} ; \circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ;$<br>$f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 69, TwoChop-RulesImp3 |
| 71 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\land \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 70, ITL definition of $\circ$ |
| 72 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\land \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\lor f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\land \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\lor f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\land \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\lor f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \land \circ f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e})$ | 71, $\lor$ introduction |
| 73 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \land \text{skip}) ; ((w_{C2} ; w_{C3} ; \neg w_{C5'})$<br>$\land (\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | CP assumption |
| 74 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\land \text{skip} ; \circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ;$<br>$f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 73, TwoChop-RulesImp3 |
| 75 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\land \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 74, ITL definition of $\circ$ |

| | | |
|---|---|---|
| 76 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \wedge \circ f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e})$ | 75, $\vee$ introduction and comm. of $\vee$ |
| 77 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; ((w_{C2} ; \neg w_{C3})$<br>$\wedge (\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e})) ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | CP assumption |
| 78 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \text{skip} ; \circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ;$<br>$f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 77, TwoChop-RulesImp3 |
| 79 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 78, ITL definition of $\circ$ |
| 80 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \wedge \circ f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e})$ | 79, $\vee$ introduction and comm. of $\vee$ |
| 81 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} \wedge \text{skip}) ; (\neg w_{C2} \wedge \text{empty}) ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e}$ | CP assumption |

| | | |
|---|---|---|
| 82 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \wedge \text{skip} ; \text{empty} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 81, TwoChop-RulesImp3 |
| 83 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \wedge \text{skip} ; f_{1b} ; rule_{1'} ; f_{0c} ;$<br>$f_{0d} ; f_{0e})$ | 82, EmptyChop |
| 84 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \wedge \circ f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ;$<br>$f_{0e})$ | 83, ITL definition of $\circ$ |
| 85 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \wedge \circ f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e})$ | 84, $\vee$ introduction and comm. of $\vee$ |
| 86 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2}$<br>$\wedge \circ f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$ | 85, $\vee$ elimination |

| | | |
|---|---|---|
| 87 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2}$<br>$\wedge \circ f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e})$ | 86, $\vee$ introduction |
| 88 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e})$ | CP assumption |
| 89 | $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$<br>$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$<br>$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$<br>$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ;$<br>$f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2}$<br>$\wedge \circ f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$<br>$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e})$ | 88, $\vee$ introduction and comm. of $\vee$ |

90 $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$ 89, $\vee$ elimination

$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$

$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$

$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$

$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; w_{C2} ; \neg w_{C3}$

$\circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ;$

$f_{0d} ; f_{0e})$

$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (w_{C1'} ; \neg w_{C2} \wedge \circ f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$\vee f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e})$

---

91 $f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; ((w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$ ITL (OrChopEqv)

$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ;$

$rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$\vee (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$

$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ;$

$rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$\vee (w_{C1'} ; w_{C2} ; \neg w_{C3}$

$\circ\circ f_{3a} ; f_{3b} ; \circ f_{4c} ; f_{4d} ; f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ;$

$f_{0d} ; f_{0e})$

$\vee (w_{C1'} ; \neg w_{C2} \wedge \circ f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$\vee (\neg w_{C1'} \wedge f_{0c} ; f_{0d} ; f_{0e}))$

---

With this transformation and based on the premises $f_0$, $rule_1$, $rule_{1'}$, $rule_2$, $rule_3$, $rule_5$, and $rule_{5'}$ as extracted from the legacy code, the following disjunctive rule structure is concluded:

$f_{0a} ; f_{pl} ; f_{0b} ; f_{1a} ; ($

$(w_{C1'} ; w_{C2} ; w_{C3} ; w_{C5'}$
$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; \circ f_{6b} ; f_{6c} ; rule_{5'} ; f_{4a} ; f_{4b} ;$
$f_{3c} ; f_{3d} ; rule_4 ; f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$\vee (w_{C1'} ; w_{C2} ; w_{C3} ; \neg w_{C5'}$
$\wedge \circ\circ f_{3a} ; f_{3b} ; \circ f_{6a} ; f_{4a} ; f_{4b} ; f_{3c} ; f_{3d} ; rule_4 ;$
$f_{3e} ; f_{1b} ; rule_{1'} ; f_{0c} ; f_{0d} ; f_{0e})$

$$
\begin{aligned}
&\vee\ (w_{C1'}\,;\, w_{C2}\,;\, \neg w_{C3} \\
&\quad \wedge\ \circ\circ f_{3a}\,;f_{3b}\,;\,\circ f_{4c}\,;f_{4d}\,;f_{3c}\,;f_{3d}\,;\, rule_{4}\,; \\
&\qquad f_{3e}\,;f_{1b}\,;\, rule_{1'}\,;f_{0c}\,;f_{0d}\,;f_{0e}) \\
&\vee\ (w_{C1'}\,;\, \neg w_{C2} \wedge \circ f_{1b}\,;\, rule_{1'}\,;f_{0c}\,;f_{0d}\,;f_{0e}) \\
&\vee\ (\neg w_{C1'} \wedge f_{0c}\,;f_{0d}\,;f_{0e}))
\end{aligned}
$$

# Appendix D

# Formal Transformation of Rules Extracted from WSL Slices

In Section 8.2, various rules are extracted from a WSL program using the FermaT Syntactic_Slice tranformation. In this appendix, these extracted rules are transformed using the rule algebra presented in this research. In Section D.1, $rule_{pc10\text{-}2}$ is transformed. In Section D.2, $rule_{pers\text{-}2}$ is transformed. In Section D.3, $rule_{personal\text{-}cond}$ is transformed.

## D.1 Transformation of $rule_{pc10\text{-}2}$

In Section 8.2, $rule_{pc10\text{-}2}$ is described as :

$$rule_{pc10\text{-}2} \equiv (rule_{pc10\text{-}2a(true)} \wedge \circ rule_{pc10\text{-}2b}) \vee rule_{pc10\text{-}2a(false)}$$

where:

$$\begin{aligned} rule_{pc10\text{-}2a(true)} &\triangleq ((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ t) \\ rule_{pc10\text{-}2a(false)} &\triangleq (\neg(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \text{empty}) \\ rule_{pc10\text{-}2b} &\triangleq \quad (t > 3740 \wedge \circ \text{pc10} = t) \\ &\qquad \vee (\neg(t > 3740) \wedge \circ \text{pc10} = 3740) \\ t &\triangleq \text{pc10} - (\text{income} - 16800) / 2 \end{aligned}$$

In the following transformation, $rule_{pc10\text{-}2}$ is transformed and simplified such that:

$$\begin{aligned} & rule_{pc10\text{-}2} \supset \\ & (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t > 3740 \\ & \quad \wedge \circ\circ \text{pc10} = t) \\ & \vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t \leq 3740 \\ & \quad \wedge \circ\circ \text{pc10} = 3740) \\ & \vee ((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800) \wedge \text{empty}) \end{aligned}$$

where:

$$t \triangleq \text{pc10} - (\text{income} - 16800) / 2$$

Alternatively, $rule_{pc10\text{-}2}$ can be described as:

$$\begin{aligned}
&\mathit{rule}_{pc10\text{-}2} \supset \\
&(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t > 3740 \\
&\quad \wedge \bigcirc\bigcirc pc10 = t) \\
&\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge t \leq 3740 \\
&\quad \wedge \bigcirc\bigcirc pc10 = 3740) \\
&\vee (\text{married} \neq 1 \wedge \mathsf{empty}) \\
&\vee (\text{age} < 65 \wedge \mathsf{empty}) \\
&\vee (\text{income} \leq 16800 \wedge \mathsf{empty})
\end{aligned}$$

where:

$$t \triangleq pc10 - (\text{income} - 16800) / 2$$

| | | |
|---|---|---|
| 1 | $\mathit{rule}_{pc10\text{-}2} \equiv (\mathit{rule}_{pc10\text{-}2a(true)} \wedge \bigcirc \mathit{rule}_{pc10\text{-}2b}) \vee \mathit{rule}_{pc10\text{-}2a(false)}$ <br> where: <br> $\mathit{rule}_{pc10\text{-}2a(true)} \triangleq$ <br> $((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \bigcirc t)$ <br> $\mathit{rule}_{pc10\text{-}2a(false)} \triangleq$ <br> $(\neg(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \mathsf{empty})$ <br> $\mathit{rule}_{pc10\text{-}2b} \triangleq$ <br> $(t > 3740 \wedge \bigcirc pc10 = t) \vee (\neg(t > 3740) \wedge \bigcirc pc10 = 3740)$ <br> $t \triangleq pc10 - (\text{income} - 16800) / 2$ | premise |
| 2 | $\bigcirc(t > 3740) \supset (t > 3740)$ | premise |
| 3 | $\bigcirc(t \leq 3740) \supset (t \leq 3740)$ | premise |
| 4 | $(\mathit{rule}_{pc10\text{-}2a(true)} \wedge \bigcirc \mathit{rule}_{pc10\text{-}2b}) \equiv (\mathit{rule}_{pc10\text{-}2a(true)} \wedge \bigcirc \mathit{rule}_{pc10\text{-}2b})$ | tautology |
| 5 | $(\mathit{rule}_{pc10\text{-}2a(true)} \wedge \bigcirc \mathit{rule}_{pc10\text{-}2b}) \equiv$ <br> $((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \bigcirc t)$ <br> $\wedge \bigcirc((t > 3740 \wedge \bigcirc pc10 = t)$ <br> $\vee (\neg(t > 3740) \wedge \bigcirc pc10 = 3740))$ | 1, 4, equiv. subst. |
| 6 | $(\mathit{rule}_{pc10\text{-}2a(true)} \wedge \bigcirc \mathit{rule}_{pc10\text{-}2b}) \equiv$ <br> $((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \bigcirc t)$ <br> $\wedge \bigcirc(((t > 3740) \wedge \bigcirc pc10 = t)$ <br> $\vee ((t \leq 3740) \wedge \bigcirc pc10 = 3740))$ | 5, algebraic equiv. |
| 7 | $(\mathit{rule}_{pc10\text{-}2a(true)} \wedge \bigcirc \mathit{rule}_{pc10\text{-}2b}) \equiv$ <br> $((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \bigcirc t)$ <br> $\wedge (\bigcirc((t > 3740) \wedge \bigcirc pc10 = t)$ <br> $\vee \bigcirc((t \leq 3740) \wedge \bigcirc pc10 = 3740))$ | 6, NextOrDistEqv |

| | | |
|---|---|---|
| 8 | $(rule_{pc10\text{-}2a(true)} \wedge \circ rule_{pc10\text{-}2b}) \equiv$<br>$((\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ t)$<br>$\wedge ((\circ(t > 3740) \wedge \circ\circ pc10 = t)$<br>$\vee (\circ(t \leq 3740) \wedge \circ\circ pc10 = 3740))$ | 7, NextAnd-DistEqv |
| 9 | $(rule_{pc10\text{-}2a(true)} \wedge \circ rule_{pc10\text{-}2b}) \equiv$<br>$(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ t$<br>$\wedge \circ(t > 3740) \wedge \circ\circ pc10 = t)$<br>$\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ t$<br>$\wedge \circ(t \leq 3740) \wedge \circ\circ pc10 = 3740)$ | 8, dist. of $\wedge$ over $\vee$ |
| 10 | $rule_{pc10\text{-}2a(false)} \equiv$<br>$\neg(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800) \wedge \text{empty}$ | 1, reiteration |
| 11 | $rule_{pc10\text{-}2a(false)} \equiv$<br>$(\neg(\text{married} = 1) \vee \neg(\text{age} \geq 65) \vee \neg(\text{income} > 16800)) \wedge$<br>$\text{empty}$ | 10, prop. logic |
| 12 | $rule_{pc10\text{-}2a(false)} \equiv$<br>$(\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800) \wedge \text{empty}$ | 11, algebraic equiv. |
| 13 | $rule_{pc10\text{-}2} \equiv (rule_{pc10\text{-}2a(true)} \wedge \circ rule_{pc10\text{-}2b}) \vee rule_{pc10\text{-}2a(false)}$ | 1, reiteration |
| 14 | $rule_{pc10\text{-}2} \equiv$<br>$(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ t$<br>$\wedge \circ(t > 3740) \wedge \circ\circ pc10 = t)$<br>$\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ t$<br>$\wedge \circ(t \leq 3740) \wedge \circ\circ pc10 = 3740)$<br>$\vee ((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800) \wedge \text{empty})$ | 9, 12, 13, equiv. subst. |
| 15 | $rule_{pc10\text{-}2}$ | CP assumption |
| 16 | $(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ t$<br>$\wedge \circ(t > 3740) \wedge \circ\circ pc10 = t)$<br>$\vee (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ t$<br>$\wedge \circ(t \leq 3740) \wedge \circ\circ pc10 = 3740)$<br>$\vee ((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800) \wedge \text{empty})$ | 14, 15, equiv. subst. |
| 17 | $\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \circ t$<br>$\wedge \circ(t > 3740) \wedge \circ\circ pc10 = t$ | CP assumption |
| 18 | $\circ(t > 3740)$ | 17, $\wedge$ elimination |
| 19 | $(t > 3740)$ | 2, 18, MP |
| 20 | $\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge \circ\circ pc10 = t$ | 17, $\wedge$ elimination |

| | | |
|---|---|---|
| 21 | $\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t > 3740 \wedge \text{oopc10} = t$ | 19, 20, $\wedge$ introduction |
| 22 | $(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t > 3740 \wedge \text{oopc10} = t)$<br>$\vee\, (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t \leq 3740 \wedge \text{oopc10} = 3740)$<br>$\vee\, ((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800)$<br>$\wedge\, \text{empty})$ | 21, $\vee$ introduction |
| 23 | $\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800 \wedge \text{o}t$<br>$\wedge\, \text{o}(t \leq 3740) \wedge \text{oopc10} = 3740$ | CP assumption |
| 24 | $\text{o}(t \leq 3740)$ | 23, $\wedge$ elimination |
| 25 | $t \leq 3740$ | 3, 24, MP |
| 26 | $\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, \text{oopc10} = 3740$ | 23, $\wedge$ elimination |
| 27 | $\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t \leq 3740 \wedge \text{oopc10} = 3740$ | 25, 26, $\wedge$ introduction |
| 28 | $(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t > 3740 \wedge \text{oopc10} = t)$<br>$\vee\, (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t \leq 3740 \wedge \text{oopc10} = 3740)$<br>$\vee\, ((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800)$<br>$\wedge\, \text{empty})$ | 27, $\vee$ introduction |
| 29 | $((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800) \wedge \text{empty})$ | CP assumption |
| 30 | $(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t > 3740 \wedge \text{oopc10} = t)$<br>$\vee\, (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t \leq 3740 \wedge \text{oopc10} = 3740)$<br>$\vee\, ((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800)$<br>$\wedge\, \text{empty})$ | 29, $\vee$ introduction |
| 31 | $(\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t > 3740 \wedge \text{oopc10} = t)$<br>$\vee\, (\text{married} = 1 \wedge \text{age} \geq 65 \wedge \text{income} > 16800$<br>$\wedge\, t \leq 3740 \wedge \text{oopc10} = 3740)$<br>$\vee\, ((\text{married} \neq 1 \vee \text{age} < 65 \vee \text{income} \leq 16800)$<br>$\wedge\, \text{empty})$ | 17-22, 23-28, 29-30, $\vee$ elimination |

| | | |
|---|---|---|
| 32 | $rule_{pc10\text{-}2} \supset$ <br> $(\text{married} = 1 \land \text{age} \geq 65 \land \text{income} > 16800$ <br> $\land\, t > 3740 \land \circ\circ pc10 = t)$ <br> $\lor\, (\text{married} = 1 \land \text{age} \geq 65 \land \text{income} > 16800$ <br> $\land\, t \leq 3740 \land \circ\circ pc10 = 3740)$ <br> $\lor\, ((\text{married} \neq 1 \lor \text{age} < 65 \lor \text{income} \leq 16800)$ <br> $\land\, \text{empty})$ | 15-31, $\supset$ introduction |
| 33 | $rule_{pc10\text{-}2} \supset$ <br> $(\text{married} = 1 \land \text{age} \geq 65 \land \text{income} > 16800$ <br> $\land\, t > 3740 \land \circ\circ pc10 = t)$ <br> $\lor\, (\text{married} = 1 \land \text{age} \geq 65 \land \text{income} > 16800$ <br> $\land\, t \leq 3740 \land \circ\circ pc10 = 3740)$ <br> $\lor\, (\text{married} \neq 1 \land \text{empty})$ <br> $\lor\, (\text{age} < 65 \land \text{empty})$ <br> $\lor\, (\text{income} \leq 16800 \land \text{empty})$ | 32, dist. of $\land$ over $\lor$ |

## D.2 Transformation of *rule*$_{pers\text{-}2}$

In Section 8.2, $rule_{pers\text{-}2}$ is described as :

$$rule_{pers\text{-}2} \equiv (rule_{pers\text{-}2a(true)} \land \circ rule_{pers\text{-}2b}) \lor rule_{pers\text{-}2a(false)} \qquad (8.2\text{-}39)$$

where:

$$\begin{aligned}
&rule_{pers\text{-}2a(true)} \triangleq ((\text{age} \geq 65 \land \text{income} > 16800) \land \circ t)\\
&rule_{pers\text{-}2a(false)} \triangleq (\neg(\text{age} \geq 65 \land \text{income} > 16800) \land \text{empty})\\
&rule_{pers\text{-}2b} \triangleq \quad (t > 4335 \land \circ\text{personal} = t)\\
&\qquad\qquad\quad \lor (\neg(t > 4335) \land \circ\text{personal} = 4335)\\
&t \triangleq \text{personal} - (\text{income} - 16800) / 2
\end{aligned}$$

In the following transformation, $rule_{pers\text{-}2}$ is transformed and simplified such that:

$$\begin{aligned}
&rule_{pers\text{-}2} \supset\\
&(\text{age} \geq 65 \land \text{income} > 16800 \land \circ t > 4335 \land \circ\circ\text{personal} = t)\\
&\lor (\text{age} \geq 65 \land \text{income} > 16800 \land \circ t \leq 4335 \land \circ\circ\text{personal} = 4335)\\
&\lor (\text{age} < 65 \land \text{empty})\\
&\lor (\text{income} \leq 16800 \land \text{empty})
\end{aligned}$$

where:

$t \triangleq \text{personal} - (\text{income} - 16800) / 2$

| | | |
|---|---|---|
| 1 | $rule_{pers\text{-}2}$<br>where:<br>$rule_{pers\text{-}2} \triangleq (rule_{pers\text{-}2a(true)} \wedge \circ rule_{pers\text{-}2b})$<br>$\vee\ rule_{pers\text{-}2a(false)}$<br>$rule_{pers\text{-}2a(true)} \triangleq ((\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ t)$<br>$rule_{pers\text{-}2a(false)} \triangleq (\neg(\text{age} \geq 65 \wedge \text{income} > 16800)$<br>$\wedge\ \text{empty})$<br>$rule_{pers\text{-}2b} \triangleq (t > 4335 \wedge \circ\text{personal} = t)$<br>$\vee\ (\neg(t > 4335) \wedge \circ\text{personal} = 4335)$<br>$t \triangleq \text{personal} - (\text{income} - 16800) / 2$ | premise |
| 2 | $w_1 \triangleq (\text{age} \geq 65 \wedge \text{income} > 16800)$<br>$\circ f_1 \triangleq \circ t$<br>$w_2 \triangleq (t > 4335)$<br>$\neg w_2 \triangleq (t \leq 4335)$<br>$\circ f_2 \triangleq (\circ\text{personal} = t)$<br>$\circ f_3 \triangleq (\circ\text{personal} = 4335)$ | premise (definitions) |
| 3 | $rule_{pers\text{-}2a(true)} \triangleq w_1 \wedge \circ f_1$ | 1, 2, def. subst. |
| 4 | $rule_{pers\text{-}2a(false)} \triangleq \neg w_1 \wedge \text{empty}$ | 1, 2, def. subst. |
| 5 | $rule_{pers\text{-}2b} \triangleq (w_2 \wedge \circ f_2) \vee (\neg w_2 \wedge \circ f_3)$ | 1, 2, def. subst. |
| 6 | $rule_{pers\text{-}2}$ | 1, reiteration |
| 7 | $(rule_{pers\text{-}2a(true)} \wedge \circ rule_{pers\text{-}2b}) \vee rule_{pers\text{-}2a(false)}$ | 1, 6, def. subst. |
| 8 | $rule_{pers\text{-}2a(true)} \wedge \circ rule_{pers\text{-}2b}$ | CP assumption |
| 9 | $(w_1 \wedge \circ f_1) \wedge \circ((w_2 \wedge \circ f_2) \vee (\neg w_2 \wedge \circ f_3))$ | 3, 5, 8, def. subst. |
| 10 | $(w_1 \wedge \circ f_1) \wedge (\circ(w_2 \wedge \circ f_2) \vee \circ(\neg w_2 \wedge \circ f_3))$ | 9, NextOrDistEqv |
| 11 | $((w_1 \wedge \circ f_1) \wedge \circ(w_2 \wedge \circ f_2))$<br>$\vee\ ((w_1 \wedge \circ f_1) \wedge \circ(\neg w_2 \wedge \circ f_3))$ | 10, comm. of $\wedge$ over $\vee$ |
| 12 | $(w_1 \wedge \circ f_1) \wedge \circ(w_2 \wedge \circ f_2)$ | CP assumption |
| 13 | $w_1 \wedge \circ f_1 \wedge \circ w_2 \wedge \circ\circ f_2$ | 12, NextAndDistEqv |
| 14 | $w_1 \wedge \circ w_2 \wedge \circ\circ f_2$ | 13, $\wedge$ elimination |
| 15 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee\ (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$ | 14, $\vee$ introduction |
| 16 | $(w_1 \wedge \circ f_1) \wedge \circ(\neg w_2 \wedge \circ f_3)$ | CP assumption |
| 17 | $w_1 \wedge \circ f_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3$ | 16, NextAndDistEqv |
| 18 | $w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3$ | 17, $\wedge$ elimination |

| | | |
|---|---|---|
| 19 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$ | 18, $\vee$ introduction |
| 20 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$ | 12-15, 16-19,<br>$\vee$ elimination |
| 21 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee (\neg w_1 \wedge \text{empty})$ | 20, $\vee$ introduction |
| 22 | $rule_{pers\text{-}2a(false)}$ | CP assumption |
| 23 | $\neg w_1 \wedge \text{empty}$ | 4, 22, def. subst. |
| 24 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee (\neg w_1 \wedge \text{empty})$ | 23, $\vee$ introduction |
| 25 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee (\neg w_1 \wedge \text{empty})$ | 8-21, 22-24,<br>$\vee$ elimination |
| 26 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee (\neg(\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \text{empty})$ | 2, 25, def. subst. |
| 27 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee ((\neg(\text{age} \geq 65) \vee \neg(\text{income} > 16800)) \wedge \text{empty})$ | 26, prop. logic |
| 28 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee ((\neg(\text{age} \geq 65) \wedge \text{empty}))$<br>$\vee (\neg(\text{income} > 16800) \wedge \text{empty}))$ | 27, prop. logic |
| 29 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee (\text{age} < 65 \wedge \text{empty})$<br>$\vee (\text{income} \leq 16800 \wedge \text{empty})$ | 28, algebraic equiv. |
| 30 | $rule_{pers\text{-}2}$ | CP assumption |
| 31 | $(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee (\text{age} < 65 \wedge \text{empty})$<br>$\vee (\text{income} \leq 16800 \wedge \text{empty})$ | 29, reiteration |

|  |  |  |
|---|---|---|
| 32 | $rule_{pers\text{-}2} \supset$<br>$(w_1 \wedge \circ w_2 \wedge \circ\circ f_2)$<br>$\vee\ (w_1 \wedge \circ\neg w_2 \wedge \circ\circ f_3)$<br>$\vee\ (\text{age} < 65 \wedge \text{empty})$<br>$\vee\ (\text{income} \leq 16800 \wedge \text{empty})$ | 30-31, $\supset$ introduction |
| 33 | $rule_{pers\text{-}2} \supset$<br>$((\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ(t > 4335)$<br>$\quad \wedge\ \circ(\circ\text{personal} = t))$<br>$\vee\ ((\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ\neg(t > 4335)$<br>$\quad \wedge\ \circ(\circ\text{personal} = 4335))$<br>$\vee\ (\text{age} < 65 \wedge \text{empty})$<br>$\vee\ (\text{income} \leq 16800 \wedge \text{empty})$ | 2, 28, def. subst. |
| 34 | $rule_{pers\text{-}2} \supset$<br>$((\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ(t > 4335)$<br>$\quad \wedge\ \circ(\circ\text{personal} = t))$<br>$\vee\ ((\text{age} \geq 65 \wedge \text{income} > 16800) \wedge \circ(t \leq 4335)$<br>$\quad \wedge\ \circ(\circ\text{personal} = 4335))$<br>$\vee\ (\text{age} < 65 \wedge \text{empty})$<br>$\vee\ (\text{income} \leq 16800 \wedge \text{empty})$ | 33, algebraic equiv. |

## D.3 Transformation of *rule*$_{personal\text{-}cond}$

In Section 8.2, $rule_{personal\text{-}cond}$ is described as :

$rule_{personal\text{-}cond} \supset$

$(\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;
$(\text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t)$ ;
$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;
$(\text{income} > 16800 \wedge \circ t \leq 4335\ \wedge \circ\circ\text{personal} = 4335)$ ;
$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;
$(\text{income} \leq 16800 \wedge \text{empty})$ ;
$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)$ ;
$(\text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t)$ ;
$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\mathrm{age} < 75 \wedge \circ\circ\mathrm{personal} = 5720)\ ;$
$(\mathrm{income} > 16800 \wedge \circ t \leq 4335 \ \wedge \circ\circ\mathrm{personal} = 4335)\ ;$
$(\circ\mathrm{personal} = \mathrm{personal} + 1380)$

$\vee\ (\mathrm{age} < 75 \wedge \circ\circ\mathrm{personal} = 5720)\ ;$
$(\mathrm{income} \leq 16800 \wedge \mathsf{empty})\ ;$
$(\circ\mathrm{personal} = \mathrm{personal} + 1380)$

In the following transformation, $rule_{personal\text{-}cond}$ is transformed and simplified such that:

| | | |
|---|---|---|
| 1 | $\begin{array}{l} rule_{personal\text{-}cond} \supset \\ (\mathrm{age} \geq 75 \wedge \circ\mathrm{personal} = 5980)\ ; \\ (\mathrm{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\mathrm{personal} = t)\ ; \\ (\circ\mathrm{personal} = \mathrm{personal} + 1380) \\ \vee\ (\mathrm{age} \geq 75 \wedge \circ\mathrm{personal} = 5980)\ ; \\ (\mathrm{income} > 16800 \wedge \circ t \leq 4335 \ \wedge \circ\circ\mathrm{personal} = 4335)\ ; \\ (\circ\mathrm{personal} = \mathrm{personal} + 1380) \\ \vee\ (\mathrm{age} \geq 75 \wedge \circ\mathrm{personal} = 5980)\ ; \\ (\mathrm{income} \leq 16800 \wedge \mathsf{empty})\ ; \\ (\circ\mathrm{personal} = \mathrm{personal} + 1380) \\ \vee\ (\mathrm{age} < 75 \wedge \circ\circ\mathrm{personal} = 5720)\ ; \\ (\mathrm{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\mathrm{personal} = t)\ ; \\ (\circ\mathrm{personal} = \mathrm{personal} + 1380) \\ \vee\ (\mathrm{age} < 75 \wedge \circ\circ\mathrm{personal} = 5720)\ ; \\ (\mathrm{income} > 16800 \wedge \circ t \leq 4335 \ \wedge \circ\circ\mathrm{personal} = 4335)\ ; \\ (\circ\mathrm{personal} = \mathrm{personal} + 1380) \\ \vee\ (\mathrm{age} < 75 \wedge \circ\circ\mathrm{personal} = 5720)\ ; \\ (\mathrm{income} \leq 16800 \wedge \mathsf{empty})\ ; \\ (\circ\mathrm{personal} = \mathrm{personal} + 1380) \\ \text{where:} \\ \quad t \triangleq \mathrm{personal} - (\mathrm{income} - 16800) / 2 \end{array}$ | premise |
| 2 | $\circ(\mathrm{income} < 20090) \supset (\mathrm{income} < 20090)$ | premise |
| 3 | $\circ(\mathrm{income} \geq 20090) \supset (\mathrm{income} \geq 20090)$ | premise |
| 4 | $\circ(\mathrm{income} < 19570) \supset (\mathrm{income} < 19570)$ | premise |
| 5 | $\circ(\mathrm{income} \geq 19570) \supset (\mathrm{income} \geq 19570)$ | premise |

| | | |
|---|---|---|
| 6 | $(\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;<br>$(\text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\vee\ (\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;<br>$(\text{income} > 16800 \wedge \circ t \leq 4335 \wedge \circ\circ\text{personal} = 4335)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\vee\ (\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;<br>$(\text{income} \leq 16800 \wedge \text{empty})$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\vee\ (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)$ ;<br>$(\text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\vee\ (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)$ ;<br>$(\text{income} > 16800 \wedge \circ t \leq 4335 \wedge \circ\circ\text{personal} = 4335)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\vee\ (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)$ ;<br>$(\text{income} \leq 16800 \wedge \text{empty})$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | CP assumption |
| 7 | $(\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;<br>$(\text{income} > 16800 \wedge \circ(t > 4335) \wedge \circ\circ\text{personal} = t)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | CP assumption<br>(disjunct #1) |
| 8 | $t > 4335 \equiv \text{income} < 20090$ | 1, 7, semantics of ITL |
| 9 | $\circ(t > 4335) \equiv \circ(\text{income} < 20090)$ | 8, ITL (NextEqvNext) |
| 10 | $(\text{age} \geq 75 \wedge \circ\text{personal} = 5980)$ ;<br>$(\text{income} > 16800 \wedge \circ(\text{income} < 20090)$<br>$\wedge\ \circ\circ\text{personal} = t)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 7, 9, equiv. subst. |
| 11 | $\text{income} > 16800 \wedge \circ(\text{income} < 20090)$<br>$\wedge\ \circ\circ\text{personal} = t$ | CP assumption |
| 12 | $\circ(\text{income} < 20090)$ | 11, $\wedge$ elimination |
| 13 | $\text{income} < 20090$ | 2, 12, MP |
| 14 | $\text{income} > 16800 \wedge \circ\circ\text{personal} = t$ | 11, $\wedge$ elimination |
| 15 | $\text{income} > 16800 \wedge \text{income} < 20090$<br>$\wedge\ \circ\circ\text{personal} = t$ | $\wedge$ introduction |

| | | |
|---|---|---|
| 16 | $(\text{income} > 16800 \land \circ(\text{income} < 20090)$<br>$\land \circ\circ\text{personal} = t) \supset$<br>$(\text{income} > 16800 \land \text{income} < 20090$<br>$\land \circ\circ\text{personal} = t)$ | 11-15,<br>$\supset$ introduction |
| 17 | $(\text{age} \geq 75 \land \circ\text{personal} = 5980)$ ;<br>$(\text{income} > 16800 \land \text{income} < 20090$<br>$\land \circ\circ\text{personal} = t)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 10, 16,<br>ChopSwapImp3 |
| 18 | $(\text{age} \geq 75 \land \circ\text{personal} = 5980)$ ;<br>$(\text{income} > 16800 \land \text{income} < 20090$<br>$\land \circ\circ\text{personal} = t)$ | CP assumption |
| 19 | $\text{age} \geq 75$ ; $(\text{income} > 16800 \land \text{income} < 20090)$<br>$\land \circ\text{personal} = 5980$ ; $\circ\circ\text{personal} = t$ | 18,<br>TwoChopRulesImp |
| 20 | $(\text{age} \geq 75 \land \circ\text{personal} = 5980)$ ;<br>$(\text{income} > 16800 \land \text{income} < 20090$<br>$\land \circ\circ\text{personal} = t)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\supset \text{age} \geq 75$ ; $(\text{income} > 16800 \land \text{income} < 20090)$<br>$\land \circ\text{personal} = 5980$ ; $\circ\circ\text{personal} = t$ | 18-19,<br>$\supset$ introduction |
| 21 | $(\text{age} \geq 75$ ; $(\text{income} > 16800 \land \text{income} < 20090)$<br>$\land \circ\text{personal} = 5980$ ; $\circ\circ\text{personal} = t)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 17, 20,<br>ChopSwapImp2 |
| 22 | $\text{age} \geq 75$ ; $(\text{income} > 16800 \land \text{income} < 20090)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\land \circ\text{personal} = 5980$ ; $\circ\circ\text{personal} = t$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 21, AndChopImp |
| 23 | $\text{age} \geq 75$ ; $(\text{income} > 16800 \land \text{income} < 20090)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 22, $\land$ elimination |
| 24 | $\text{age} \geq 75$ ; $(\text{income} > 16800 \land \text{income} < 20090)$ | 23, ITL (semantics of chop) |
| 25 | $\circ\text{personal} = 5980$ ; $\circ\circ\text{personal} = t$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 22, $\land$ elimination |
| 26 | $\text{fin}(\text{personal} = 15760 - \text{income}/2)$ | 25, ITL (semantics of fin) |
| 27 | $\text{age} \geq 75$ ; $(\text{income} > 16800 \land \text{income} < 20090)$<br>$\land \text{fin}(\text{personal} = 15760 - \text{income}/2)$ | 24, 26,<br>$\land$ introduction |

| | | |
|---|---|---|
| 28 | $(age \geq 75 ; (income > 16800 \land income < 20090)$<br>$\land \mathrm{fin}(personal = 15760 - income/2))$<br>$\lor (age \geq 75 ; (income > 16800 \land income \geq 20090)$<br>$\land \mathrm{fin}(personal = 5715))$<br>$\lor (age \geq 75 ;\ income \leq 16800$<br>$\land \mathrm{fin}(personal = 7360))$<br>$\lor (age < 75 ; (income > 16800 \land income < 19570)$<br>$\land \mathrm{fin}(personal = 15500 - income/2))$<br>$\lor (age < 75 ; (income > 16800 \land income \geq 19570)$<br>$\land \mathrm{fin}(personal = 5715))$<br>$\lor (age < 75 ; income \leq 16800$<br>$\land \mathrm{fin}(personal = 7100))$ | 27, $\lor$ introduction |
| 29 | $(age \geq 75 \land \circ personal = 5980) ;$<br>$(income > 16800 \land \circ(t \leq 4335)$<br>$\land \circ\circ personal = 4335) ;$<br>$(\circ personal = personal + 1380)$ | CP assumption<br>(disjunct #2) |
| 30 | $(t \leq 4335) \equiv (income \geq 20090)$ | 1, 29, semantics of ITL |
| 31 | $\circ(t \leq 4335) \equiv \circ(income \geq 20090)$ | 30, ITL (NextEqvNext) |
| 32 | $(age \geq 75 \land \circ personal = 5980) ;$<br>$(income > 16800 \land \circ(income \geq 20090)$<br>$\land \circ\circ personal = 4335) ;$<br>$(\circ personal = personal + 1380)$ | 29, 32, equiv. subst |
| 33 | $income > 16800 \land \circ(income \geq 20090)$<br>$\land \circ\circ personal = 4335$ | CP assumption |
| 34 | $\circ(income \geq 20090)$ | 33, $\land$ elimination |
| 35 | $income \geq 20090$ | 3, 34, MP |
| 36 | $income > 16800 \land \circ\circ personal = 4335$ | $\land$ elimination |
| 37 | $income > 16800 \land income \geq 20090$<br>$\land \circ\circ personal = 4335$ | $\land$ introduction |
| 38 | $(income > 16800 \land \circ(income \geq 20090)$<br>$\land \circ\circ personal = 4335) \supset$<br>$(income > 16800 \land income \geq 20090$<br>$\land \circ\circ personal = 4335)$ | 33-37,<br>$\supset$ introduction |

| | | |
|---|---|---|
| 39 | (age ≥ 75 ∧ ○personal = 5980) ; (income > 16800 ∧ income ≥ 20090 ∧ ○○personal = 4335) ; (○personal = personal + 1380) | 32, 38, ChopSwapImp3 |
| 40 | (age ≥ 75 ∧ ○personal = 5980) ; (income > 16800 ∧ income ≥ 20090 ∧ ○personal = 4335) | CP assumption |
| 41 | age ≥ 75 ; (income > 16800 ∧ income ≥ 20090) ∧ ○personal = 5980 ; ○personal = 4335 | 40, TwoChopRulesImp |
| 42 | (age ≥ 75 ∧ ○personal = 5980) ; (income > 16800 ∧ income ≥ 20090 ∧ ○personal = 4335) ⊃ (age ≥ 75 ; (income > 16800 ∧ income ≥ 20090) ∧ ○personal = 5980 ; ○personal = 4335) | 41, ⊃ introduction |
| 43 | (age ≥ 75 ; (income > 16800 ∧ income ≥ 20090) ∧ ○personal = 5980 ; ○personal = 4335) ; (○personal = personal + 1380) | 42, ChopSwapImp2 |
| 44 | age ≥ 75 ; (income > 16800 ∧ income ≥ 20090) ; (○personal = personal + 1380) ∧ ○personal = 5980 ; ○personal = 4335 ; (○personal = personal + 1380) | 43, AndChopImp |
| 45 | age ≥ 75 ; (income > 16800 ∧ income ≥ 20090) ; (○personal = personal + 1380) | 44, ∧ elimination |
| 46 | age ≥ 75 ; (income > 16800 ∧ income ≥ 20090) | 45, ITL (semantics of chop) |
| 47 | ○personal = 5980 ; ○personal = 4335 ; (○personal = personal + 1380) | 46, ∧ elimination |
| 48 | fin(personal = 5715) | 47, ITL (semantics of fin) |
| 49 | age ≥ 75 ; (income > 16800 ∧ income ≥ 20090) ∧ fin(personal = 5715) | 46, 48, ∧ introduction |

| | | |
|---|---|---|
| 50 | $(age \geq 75 ; (income > 16800 \wedge income < 20090)$<br>$\wedge\ fin(personal = 15760 - income/2))$<br>$\vee (age \geq 75 ; (income > 16800 \wedge income \geq 20090)$<br>$\wedge\ fin(personal = 5715))$<br>$\vee (age \geq 75 ;\ income \leq 16800$<br>$\wedge\ fin(personal = 7360))$<br>$\vee (age < 75 ; (income > 16800 \wedge income < 19570)$<br>$\wedge\ fin(personal = 15500 - income/2))$<br>$\vee (age < 75 ; (income > 16800 \wedge income \geq 19570)$<br>$\wedge\ fin(personal = 5715))$<br>$\vee (age < 75 ; income \leq 16800$<br>$\wedge\ fin(personal = 7100))$ | 49, $\vee$ introduction |
| 51 | $(age \geq 75 \wedge \bigcirc personal = 5980) ;$<br>$(income \leq 16800 \wedge empty) ;$<br>$(\bigcirc personal = personal + 1380)$ | CP assumption<br>(disjunct #3) |
| 52 | $(age \geq 75 \wedge \bigcirc personal = 5980) ;$<br>$(income \leq 16800 \wedge empty)$ | CP assumption |
| 53 | $age \geq 75 ;\ income \leq 16800$<br>$\wedge \bigcirc personal = 5980 ; empty$ | 52,<br>TwoChopRulesImp |
| 54 | $age \geq 75 ;\ income \leq 16800 \wedge \bigcirc personal = 5980$ | 53, ITL<br>(ChopEmpty) |
| 55 | $(age \geq 75 \wedge \bigcirc personal = 5980) ;$<br>$(income \leq 16800 \wedge empty)$<br>$\supset age \geq 75 ;\ income \leq 16800 \wedge \bigcirc personal = 5980$ | 51-54,<br>$\supset$ introduction |
| 56 | $(age \geq 75 ;\ income \leq 16800 \wedge \bigcirc personal = 5980) ;$<br>$(\bigcirc personal = personal + 1380)$ | 51, 55,<br>ChopSwapImp2 |
| 57 | $age \geq 75 ;\ income \leq 16800 ;$<br>$(\bigcirc personal = personal + 1380)$<br>$\wedge \bigcirc personal = 5980 ;$<br>$(\bigcirc personal = personal + 1380)$ | 56, AndChopImp |
| 58 | $age \geq 75 ;\ income \leq 16800 ;$<br>$(\bigcirc personal = personal + 1380)$ | 57, $\wedge$ elimination |
| 59 | $age \geq 75 ;\ income \leq 16800$ | 58, ITL (semantics<br>of chop) |
| 60 | $\bigcirc personal = 5980 ;$<br>$(\bigcirc personal = personal + 1380)$ | 57, $\wedge$ elimination |

| | | |
|---|---|---|
| 61 | $\mathrm{fin}(personal = 7360)$ | 60, ITL (semantics of fin) |
| 62 | $age \geq 75$ ; $income \leq 16800$ $\wedge\ \mathrm{fin}(personal = 7360)$ | 61, $\wedge$ introduction |
| 63 | $(age \geq 75 ; (income > 16800 \wedge income < 20090)$ $\wedge\ \mathrm{fin}(personal = 15760 - income/2))$ $\vee\ (age \geq 75 ; (income > 16800 \wedge income \geq 20090)$ $\wedge\ \mathrm{fin}(personal = 5715))$ $\vee\ (age \geq 75 ;\ income \leq 16800$ $\wedge\ \mathrm{fin}(personal = 7360))$ $\vee\ (age < 75 ; (income > 16800 \wedge income < 19570)$ $\wedge\ \mathrm{fin}(personal = 15500 - income/2))$ $\vee\ (age < 75 ; (income > 16800 \wedge income \geq 19570)$ $\wedge\ \mathrm{fin}(personal = 5715))$ $\vee\ (age < 75 ; income \leq 16800$ $\wedge\ \mathrm{fin}(personal = 7100))$ | 62, $\vee$ introduction |
| 64 | $(age < 75 \wedge \circ\circ personal = 5720)$ ; $(income > 16800 \wedge \circ(t > 4335) \wedge \circ\circ personal = t)$ ; $(\circ personal = personal + 1380)$ | CP assumption (disjunct #4) |
| 65 | $(t > 4335) \equiv (income < 19570)$ | 1, 64, semantics of ITL |
| 66 | $\circ(t > 4335) \equiv \circ(income < 19570)$ | 65, ITL (NextEqvNext) |
| 67 | $(age < 75 \wedge \circ\circ personal = 5720)$ ; $(income > 16800 \wedge \circ(income < 19570)$ $\wedge\ \circ\circ personal = t)$ ; $(\circ personal = personal + 1380)$ | 64, 67, equiv. subst |
| 68 | $income > 16800 \wedge \circ(income < 19570)$ $\wedge\ \circ\circ personal = t$ | CP assumption |
| 69 | $\circ(income < 19570)$ | 68, $\wedge$ elimination |
| 70 | $income < 19570$ | 4, 69, MP |
| 71 | $income > 16800 \wedge \circ\circ personal = t$ | 68, $\wedge$ elimination |
| 72 | $income > 16800 \wedge income < 19570$ $\wedge\ \circ\circ personal = t$ | 70, 72, $\wedge$ introduction |

| | | |
|---|---|---|
| 73 | $(income > 16800 \wedge \bigcirc(income < 19570) \wedge \bigcirc\bigcirc personal = t) \supset (income > 16800 \wedge income < 19570 \wedge \bigcirc\bigcirc personal = t)$ | 68-72, $\supset$ introduction |
| 74 | $(age < 75 \wedge \bigcirc\bigcirc personal = 5720) ; (income > 16800 \wedge income < 19570 \wedge \bigcirc\bigcirc personal = t) ; (\bigcirc personal = personal + 1380)$ | 67, 73, ChopSwapImp3 |
| 75 | $(age < 75 \wedge \bigcirc\bigcirc personal = 5720) ; (income > 16800 \wedge income < 19570 \wedge \bigcirc\bigcirc personal = t)$ | CP assumption |
| 76 | $age < 75 ; (income > 16800 \wedge income < 19570) \wedge \bigcirc\bigcirc personal = 5720 ; \bigcirc\bigcirc personal = t$ | 75, TwoChopRulesImp |
| 77 | $(age < 75 \wedge \bigcirc\bigcirc personal = 5720) ; (income > 16800 \wedge income < 19570 \wedge \bigcirc\bigcirc personal = t) \supset age < 75 ; (income > 16800 \wedge income < 19570) \wedge \bigcirc\bigcirc personal = 5720 ; \bigcirc\bigcirc personal = t$ | 75-77, $\supset$ introduction |
| 78 | $(age < 75 ; (income > 16800 \wedge income < 19570) \wedge \bigcirc\bigcirc personal = 5720 ; \bigcirc\bigcirc personal = t) ; (\bigcirc personal = personal + 1380)$ | 74, 77, ChopSwapImp2 |
| 79 | $age < 75 ; (income > 16800 \wedge income < 19570) ; (\bigcirc personal = personal + 1380) \wedge \bigcirc\bigcirc personal = 5720 ; \bigcirc\bigcirc personal = t ; (\bigcirc personal = personal + 1380)$ | 78, AndChopImp |
| 80 | $age < 75 ; (income > 16800 \wedge income < 19570) ; (\bigcirc personal = personal + 1380)$ | 79, $\wedge$ elimination |
| 81 | $age < 75 ; (income > 16800 \wedge income < 19570)$ | 80, ITL (semantics of chop) |
| 82 | $\bigcirc\bigcirc personal = 5720 ; \bigcirc\bigcirc personal = t ; (\bigcirc personal = personal + 1380)$ | 79, $\wedge$ elimination |
| 83 | $fin(personal = 15500 - income/2)$ | 82, ITL (semantics of fin) |
| 84 | $age < 75 ; (income > 16800 \wedge income < 19570) \wedge fin(personal = 15500 - income/2)$ | 81, 83, $\wedge$ introduction |

| | | |
|---|---|---|
| 85 | $(age \geq 75 ; (income > 16800 \wedge income < 20090) \wedge fin(personal = 15760 - income/2))$ $\vee (age \geq 75 ; (income > 16800 \wedge income \geq 20090) \wedge fin(personal = 5715))$ $\vee (age \geq 75 ;\ income \leq 16800 \wedge fin(personal = 7360))$ $\vee (age < 75 ; (income > 16800 \wedge income < 19570) \wedge fin(personal = 15500 - income/2))$ $\vee (age < 75 ; (income > 16800 \wedge income \geq 19570) \wedge fin(personal = 5715))$ $\vee (age < 75 ; income \leq 16800 \wedge fin(personal = 7100))$ | 84, $\vee$ introduction |
| 86 | $(age < 75 \wedge \circ\circ personal = 5720) ;$ $(income > 16800 \wedge \circ(t \leq 4335) \wedge \circ\circ personal = 4335) ;$ $(\circ personal = personal + 1380)$ | CP assumption (disjunct #5) |
| 87 | $(t \leq 4335) \equiv (income \geq 19570)$ | 1, 86, semantics of ITL |
| 88 | $\circ(t \leq 4335) \equiv \circ(income \geq 19570)$ | 87, ITL (NextEqvNext) |
| 89 | $(age < 75 \wedge \circ\circ personal = 5720) ;$ $(income > 16800 \wedge \circ(income \geq 19570) \wedge \circ\circ personal = 4335) ;$ $(\circ personal = personal + 1380)$ | 86, 88, equiv. subst |
| 90 | $income > 16800 \wedge \circ(income \geq 19570) \wedge \circ\circ personal = 4335$ | CP assumption |
| 91 | $\circ(income \geq 19570)$ | 90, $\wedge$ elimination |
| 92 | $income \geq 19570$ | 5, 91, MP |
| 93 | $income > 16800 \wedge \circ\circ personal = 4335$ | 90, $\wedge$ elimination |
| 94 | $income > 16800 \wedge income \geq 19570 \wedge \circ\circ personal = 4335$ | 91, 93, $\wedge$ introduction |
| 95 | $(income > 16800 \wedge \circ(income \geq 19570) \wedge \circ\circ personal = 4335) \supset$ $(income > 16800 \wedge income \geq 19570 \wedge \circ\circ personal = 4335)$ | 90-94, $\supset$ introduction |

| | | |
|---|---|---|
| 96 | $(\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)$ ;<br>$(\text{income} > 16800 \wedge \text{income} \geq 19570$<br>$\wedge\ \circ\circ\text{personal} = 4335)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 89, 95,<br>ChopSwapImp3 |
| 97 | $(\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)$ ;<br>$(\text{income} > 16800 \wedge \text{income} \geq 19570$<br>$\wedge\ \circ\circ\text{personal} = 4335)$ | CP assumption |
| 98 | $\text{age} < 75$ ; $(\text{income} > 16800 \wedge \text{income} \geq 19570)$<br>$\wedge\ \circ\circ\text{personal} = 5720$ ; $\circ\circ\text{personal} = 4335$ | 97,<br>TwoChopRulesImp |
| 99 | $(\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)$ ;<br>$(\text{income} > 16800 \wedge \text{income} \geq 19570$<br>$\wedge\ \circ\circ\text{personal} = 4335) \supset$<br>$\text{age} < 75$ ; $(\text{income} > 16800 \wedge \text{income} \geq 19570)$<br>$\wedge\ \circ\circ\text{personal} = 5720$ ; $\circ\circ\text{personal} = 4335$ | 97-98,<br>$\supset$ introduction |
| 100 | $(\text{age} < 75$ ; $(\text{income} > 16800 \wedge \text{income} \geq 19570)$<br>$\wedge\ \circ\circ\text{personal} = 5720$ ; $\circ\text{personal} = 4335)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 96, 99,<br>ChopSwapImp2 |
| 101 | $\text{age} < 75$ ; $(\text{income} > 16800 \wedge \text{income} \geq 19570)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\wedge\ \circ\circ\text{personal} = 5720$ ; $\circ\text{personal} = 4335$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 100, AndChopImp |
| 102 | $\text{age} < 75$ ; $(\text{income} > 16800 \wedge \text{income} \geq 19570)$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 101, $\wedge$ elimination |
| 103 | $\text{age} < 75$ ; $(\text{income} > 16800 \wedge \text{income} \geq 19570)$ | 102, ITL (semantics of chop) |
| 104 | $\circ\circ\text{personal} = 5720$ ; $\circ\text{personal} = 4335$ ;<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 101, $\wedge$ elimination |
| 105 | $\text{fin}(\text{personal} = 5715)$ | 104, ITL (semantics of fin) |
| 106 | $\text{age} < 75$ ; $(\text{income} > 16800 \wedge \text{income} \geq 19570)$<br>$\wedge\ \text{fin}(\text{personal} = 5715)$ | 103, 105,<br>$\wedge$ introduction |

| | | |
|---|---|---|
| 107 | $(\text{age} \geq 75 \,;\, (\text{income} > 16800 \wedge \text{income} < 20090)$<br>$\wedge\ \text{fin}(\text{personal} = 15760 - \text{income}/2))$<br>$\vee\ (\text{age} \geq 75 \,;\, (\text{income} > 16800 \wedge \text{income} \geq 20090)$<br>$\wedge\ \text{fin}(\text{personal} = 5715))$<br>$\vee\ (\text{age} \geq 75 \,;\ \text{income} \leq 16800$<br>$\wedge\ \text{fin}(\text{personal} = 7360))$<br>$\vee\ (\text{age} < 75 \,;\, (\text{income} > 16800 \wedge \text{income} < 19570)$<br>$\wedge\ \text{fin}(\text{personal} = 15500 - \text{income}/2))$<br>$\vee\ (\text{age} < 75 \,;\, (\text{income} > 16800 \wedge \text{income} \geq 19570)$<br>$\wedge\ \text{fin}(\text{personal} = 5715))$<br>$\vee\ (\text{age} < 75 \,;\, \text{income} \leq 16800$<br>$\wedge\ \text{fin}(\text{personal} = 7100))$ | 106, $\vee$ introduction |
| 108 | $(\text{age} < 75 \wedge \circ\circ\text{personal} = 5720) \,;$<br>$(\text{income} \leq 16800 \wedge \text{empty}) \,;$<br>$(\circ\text{personal} = \text{personal} + 1380)$ | CP assumption<br>(disjunct #6) |
| 109 | $(\text{age} < 75 \wedge \circ\circ\text{personal} = 5720) \,;$<br>$(\text{income} \leq 16800 \wedge \text{empty})$ | CP assumption |
| 110 | $\text{age} < 75 \,;\, \text{income} \leq 16800$<br>$\wedge\ \circ\circ\text{personal} = 5720 \,;\, \text{empty}$ | 109,<br>TwoChopRulesImp |
| 111 | $\text{age} < 75 \,;\, \text{income} \leq 16800 \wedge \circ\circ\text{personal} = 5720$ | 110, ITL<br>(ChopEmpty) |
| 112 | $((\text{age} < 75 \wedge \circ\circ\text{personal} = 5720) \,;$<br>$(\text{income} \leq 16800 \wedge \text{empty})) \supset$<br>$(\text{age} < 75 \,;\, \text{income} \leq 16800 \wedge \circ\circ\text{personal} = 5720)$ | 109-111,<br>$\supset$ introduction |
| 113 | $(\text{age} < 75 \,;\, \text{income} \leq 16800 \wedge \circ\circ\text{personal} = 5720) \,;$<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 108, 112,<br>ChopSwapImp2 |
| 114 | $\text{age} < 75 \,;\, \text{income} \leq 16800 \,;$<br>$(\circ\text{personal} = \text{personal} + 1380)$<br>$\wedge\ \circ\circ\text{personal} = 5720 \,;$<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 113, AndChopImp |
| 115 | $\text{age} < 75 \,;\, \text{income} \leq 16800 \,;$<br>$(\circ\text{personal} = \text{personal} + 1380)$ | 114, $\wedge$ elimination |
| 116 | $\text{age} < 75 \,;\, \text{income} \leq 16800$ | 115, ITL (semantics<br>of chop) |
| 117 | $\circ\circ\text{personal} = 5720 \,;\, (\circ\text{personal} = \text{personal} + 1380)$ | 114, $\wedge$ elimination |

| | | |
|---|---|---|
| 118 | fin(personal $= 7100$) | 117, ITL (semantics of fin) |
| 119 | age $< 75$ ; income $\leq 16800$<br>$\wedge$ fin(personal $= 7100$) | 116, 118,<br>$\wedge$ introduction |
| 120 | (age $\geq 75$ ; (income $> 16800 \wedge$ income $< 20090$)<br>$\wedge$ fin(personal $= 15760$ - income/2))<br>$\vee$ (age $\geq 75$ ; (income $> 16800 \wedge$ income $\geq 20090$)<br>$\wedge$ fin(personal $= 5715$))<br>$\vee$ (age $\geq 75$ ; income $\leq 16800$<br>$\wedge$ fin(personal $= 7360$))<br>$\vee$ (age $< 75$ ; (income $> 16800 \wedge$ income $< 19570$)<br>$\wedge$ fin(personal $= 15500$ - income/2))<br>$\vee$ (age $< 75$ ; (income $> 16800 \wedge$ income $\geq 19570$)<br>$\wedge$ fin(personal $= 5715$))<br>$\vee$ (age $< 75$ ; income $\leq 16800$<br>$\wedge$ fin(personal $= 7100$)) | 119, $\vee$ introduction |
| 121 | (age $\geq 75$ ; (income $> 16800 \wedge$ income $< 20090$)<br>$\wedge$ fin(personal $= 15760$ - income/2))<br>$\vee$ (age $\geq 75$ ; (income $> 16800 \wedge$ income $\geq 20090$)<br>$\wedge$ fin(personal $= 5715$))<br>$\vee$ (age $\geq 75$ ; income $\leq 16800$<br>$\wedge$ fin(personal $= 7360$))<br>$\vee$ (age $< 75$ ; (income $> 16800 \wedge$ income $< 19570$)<br>$\wedge$ fin(personal $= 15500$ - income/2))<br>$\vee$ (age $< 75$ ; (income $> 16800 \wedge$ income $\geq 19570$)<br>$\wedge$ fin(personal $= 5715$))<br>$\vee$ (age $< 75$ ; income $\leq 16800$<br>$\wedge$ fin(personal $= 7100$)) | 7-28, 29-50, 51-63, 64-85, 86-107, 108-120,<br>$\vee$ elimination |

122 $(\text{age} \ge 75 \wedge \circ\text{personal} = 5980)\ ;$ 6-121,

$(\text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t)\ ;$ $\supset$ introduction

$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} \ge 75 \wedge \circ\text{personal} = 5980)\ ;$

$(\text{income} > 16800 \wedge \circ t \le 4335 \wedge \circ\circ\text{personal} = 4335)\ ;$

$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} \ge 75 \wedge \circ\text{personal} = 5980)\ ;$

$(\text{income} \le 16800 \wedge \mathsf{empty})\ ;$

$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)\ ;$

$(\text{income} > 16800 \wedge \circ t > 4335 \wedge \circ\circ\text{personal} = t)\ ;$

$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)\ ;$

$(\text{income} > 16800 \wedge \circ t \le 4335 \wedge \circ\circ\text{personal} = 4335)\ ;$

$(\circ\text{personal} = \text{personal} + 1380)$

$\vee\ (\text{age} < 75 \wedge \circ\circ\text{personal} = 5720)\ ;$

$(\text{income} \le 16800 \wedge \mathsf{empty})\ ;$

$(\circ\text{personal} = \text{personal} + 1380)$

$\supset (\text{age} \ge 75\ ;\ (\text{income} > 16800 \wedge \text{income} < 20090)$

$\wedge\ \mathsf{fin}(\text{personal} = 15760 - \text{income}/2))$

$\vee\ (\text{age} \ge 75\ ;\ (\text{income} > 16800 \wedge \text{income} \ge 20090)$

$\wedge\ \mathsf{fin}(\text{personal} = 5715))$

$\vee\ (\text{age} \ge 75\ ;\ \text{income} \le 16800$

$\wedge\ \mathsf{fin}(\text{personal} = 7360))$

$\vee\ (\text{age} < 75\ ;\ (\text{income} > 16800 \wedge \text{income} < 19570)$

$\wedge\ \mathsf{fin}(\text{personal} = 15500 - \text{income}/2))$

$\vee\ (\text{age} < 75\ ;\ (\text{income} > 16800 \wedge \text{income} \ge 19570)$

$\wedge\ \mathsf{fin}(\text{personal} = 5715))$

$\vee\ (\text{age} < 75\ ;\ \text{income} \le 16800$

$\wedge\ \mathsf{fin}(\text{personal} = 7100))$

123 $rule_{personal\text{-}cond} \supset$ 1, 122, prop. logic

(age $\geq$ 75 ; (income > 16800 $\wedge$ income < 20090)

$\wedge$ fin(personal = 15760 - income/2))

$\vee$ (age $\geq$ 75 ; (income > 16800 $\wedge$ income $\geq$ 20090)

$\wedge$ fin(personal = 5715))

$\vee$ (age $\geq$ 75 ; income $\leq$ 16800

$\wedge$ fin(personal = 7360))

$\vee$ (age < 75 ; (income > 16800 $\wedge$ income < 19570)

$\wedge$ fin(personal = 15500 - income/2))

$\vee$ (age < 75 ; (income > 16800 $\wedge$ income $\geq$ 19570)

$\wedge$ fin(personal = 5715))

$\vee$ (age < 75 ; income $\leq$ 16800

$\wedge$ fin(personal = 7100))

# Appendix E

# Formal Transformation of Rules Created to Refine a Specification

In Section 9.1, the following rules are developed to refine the description of the state sequence get_pin. (The state sequence get_pin was extracted from a concrete specification describing the operation of an automated teller machine in Section 7.2.) the following rules and rule structures have been developed as part of the refinement of get_pin:

$$\text{get\_pin} \hat{=} \text{init\_pin\_entry} \; ; \; \mathit{rule}_{\text{pin\_entry}}$$

$$\begin{aligned} \mathit{rule}_{\text{pin\_entry}} \hat{=} & (((\neg \mathit{attempt\_limit} \wedge \neg \mathit{valid\_pin}) \\ & \wedge \circ\text{process\_pin}) \; ; \; \mathit{rule}_{\text{pin\_entry}})) \\ & \vee (\mathit{valid\_pin} \wedge \text{empty}) \\ & \vee (\mathit{attempt\_limit} \wedge \text{empty}) \end{aligned}$$

$$\text{process\_pin} \hat{=} \text{display\_pin\_screen} \; ; \; \mathit{rule}_{\text{read\_key\_pad}} \; ; \; \mathit{rule}_{\text{validate\_pin}}$$

$$\begin{aligned} \mathit{rule}_{\text{read\_key\_pad}} \hat{=} & (\neg \mathit{enter\_key} \wedge \circ\text{key\_buffer}) \; ; \; \mathit{rule}_{\text{read\_key\_pad}} \\ & \vee (\mathit{enter\_key} \wedge \circ\text{increment\_attempt}) \end{aligned}$$

$$\begin{aligned} \mathit{rule}_{\text{validate\_pin}} \hat{=} & (\mathit{pin\_length} \wedge \circ\mathit{rule}_{\text{compare\_pin}}) \\ & \vee (\neg \mathit{pin\_length} \wedge \circ\text{display\_invalid\_screen}) \end{aligned}$$

$$\begin{aligned} \mathit{rule}_{\text{compare\_pin}} \hat{=} & (\mathit{pin\_match} \wedge \circ\text{pin\_valid}) \\ & \vee (\neg \mathit{pin\_match} \wedge \circ\text{display\_invalid\_screen}) \end{aligned}$$

To facilitate analysis, the following variable name substitutions are made:

| | |
|---|---|
| dps $\hat{=}$ | display_pin_screen |
| dis $\hat{=}$ | display_invalid_screen |
| gp $\hat{=}$ | get_pin |
| ia $\hat{=}$ | increment_attempt |
| ipe $\hat{=}$ | init_pin_entry |
| kb $\hat{=}$ | key_buffer |
| pe $\hat{=}$ | pin_entry |
| pp $\hat{=}$ | process_pin |
| pv $\hat{=}$ | pin_valid |
| *xal* $\hat{=}$ | *attempt_limit* |

$xek \triangleq$ *enter_key*
$xpl \triangleq$ *pin_length*
$xpm \triangleq$ *pin_match*
$xvp \triangleq$ *valid_pin*

Regarding these variable names, rule conditions variables begin with the letter $x$ and are depicted in italics. Using these rule conditions and rule state variable names, the rules of interest are rewritten as:

$$\text{gp} \triangleq \text{ipe} \; ; \; rule_{\text{pin\_entry}}$$

$$\begin{aligned} rule_{\text{pin\_entry}} \triangleq \; & (((\neg xal \wedge \neg xvp) \wedge \circ\text{pp}) \; ; \; rule_{\text{pin\_entry}})) \\ & \vee (xvp \wedge \text{empty}) \\ & \vee (xal \wedge \text{empty}) \end{aligned}$$

$$\text{pp} \triangleq \quad \text{dps} \; ; \; rule_{\text{read\_key\_pad}} \; ; \; rule_{\text{validate\_pin}}$$

$$rule_{\text{read\_key\_pad}} \triangleq (\neg xek \wedge \circ\text{kb}) \; ; \; rule_{\text{read\_key\_pad}} \vee (xek \wedge \circ\text{ia})$$

$$rule_{\text{validate\_pin}} \triangleq (xpl \wedge \circ rule_{\text{compare\_pin}}) \vee (\neg xpl \wedge \circ\text{dis})$$

$$rule_{\text{compare\_pin}} \triangleq (xpm \wedge \circ\text{pv}) \vee (\neg xpm \wedge \circ\text{dis})$$

Five of these rules and rule structures are assumed as premises for this tranformation – $rule_{\text{pin\_entry}}$, process_pin (pp), $rule_{\text{read\_key\_pad}}$, $rule_{\text{validate\_pin}}$, and $rule_{\text{compare\_pin}}$. The formal transformation of these rules is as follows:

| | | |
|---|---|---|
| 1 | $rule_{\text{pin\_entry}}$<br>where:<br>$rule_{\text{pin\_entry}} \triangleq (((\neg xal \wedge \neg xvp) \wedge \circ\text{pp}) \; ; \; rule_{\text{pin\_entry}}))$<br>$\vee (xvp \wedge \text{empty}) \vee (xal \wedge \text{empty})$ | premise |
| 2 | pp<br>where:<br>$\text{pp} \equiv \text{dps} \; ; \; rule_{\text{read\_key\_pad}} \; ; \; rule_{\text{validate\_pin}}$ | premise |
| 3 | $rule_{\text{read\_key\_pad}}$<br>where:<br>$rule_{\text{read\_key\_pad}} \triangleq (\neg xek \wedge \circ\text{kb}) \; ; \; rule_{\text{read\_key\_pad}}$<br>$\vee (xek \wedge \circ\text{ia})$ | premise |

| | | |
|---|---|---|
| 4 | $rule_{\text{validate\_pin}}$<br>where:<br>$rule_{\text{validate\_pin}} \mathrel{\hat{=}} (xpl \wedge \circ rule_{\text{compare\_pin}}) \vee (\neg xpl \wedge \circ\text{dis})$ | premise |
| 5 | $rule_{\text{compare\_pin}}$<br>where:<br>$rule_{\text{compare\_pin}} \mathrel{\hat{=}} (xpm \wedge \circ\text{pv}) \vee (\neg xpm \wedge \circ\text{dis})$ | premise |
| 6 | $rule_{\text{validate\_pin}} \equiv (xpl \wedge \circ rule_{\text{compare\_pin}}) \vee (\neg xpl \wedge \circ\text{dis})$ | 4, reiteration |
| 7 | $rule_{\text{validate\_pin}} \equiv$<br>$(xpl \wedge \circ((xpm \wedge \circ\text{pv}) \vee (\neg xpm \wedge \circ\text{dis})))$<br>$\vee (\neg xpl \wedge \circ\text{dis})$ | 5, 6, eqv. subst. |
| 8 | $rule_{\text{validate\_pin}} \equiv$<br>$(xpl \wedge (\circ(xpm \wedge \circ\text{pv}) \vee \circ(\neg xpm \wedge \circ\text{dis}))$<br>$\vee (\neg xpl \wedge \circ\text{dis})$ | 7, NextOrDistEqv |
| 9 | $rule_{\text{validate\_pin}} \equiv$<br>$(xpl \wedge ((\circ xpm \wedge \circ\circ\text{pv}) \vee (\circ\neg xpm \wedge \circ\circ\text{dis})))$<br>$\vee (\neg xpl \wedge \circ\text{dis})$ | 8, NextAndDistEqv |
| 10 | $rule_{\text{validate\_pin}} \equiv$<br>$(xpl \wedge (\circ xpm \wedge \circ\circ\text{pv}))$<br>$\vee (xpl \wedge (\circ\neg xpm \wedge \circ\circ\text{dis}))$<br>$\vee (\neg xpl \wedge \circ\text{dis})$ | 9, dist. of $\wedge$ over $\vee$ |
| 11 | $rule_{\text{validate\_pin}} \equiv$<br>$((xpl \wedge \circ xpm) \wedge \circ\circ\text{pv})$<br>$\vee ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\text{dis})$<br>$\vee (\neg xpl \wedge \circ\text{dis})$ | 10, prop. logic |
| 12 | dps ; $rule_{\text{read\_key\_pad}}$ ; $rule_{\text{validate\_pin}}$ | 2, reiteration |
| 13 | dps ; $rule_{\text{read\_key\_pad}}$ ;<br>$(((xpl \wedge \circ xpm) \wedge \circ\circ\text{pv})$<br>$\vee ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\text{dis})$<br>$\vee (\neg xpl \wedge \circ\text{dis}))$ | 11, 12, eqv. subst. |
| 14 | $rule_{\text{read\_key\_pad}}$ | CP assumption |
| 15 | $(\neg xek \wedge \circ\text{kb})$ ; $rule_{\text{read\_key\_pad}} \vee (xek \wedge \circ\text{ia})$ | 3, 14, eqv. subst. |

| | | |
|---|---|---|
| 16 | $(\neg xek \wedge \circ\mathrm{kb}) ; rule_{\mathrm{read\_key\_pad}}$ | CP assumption |
| 17 | $\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}$ | 16, StateAndChop |
| 18 | $(\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) \vee (xek \wedge \circ\mathrm{ia})$ | 17, $\vee$ introduction |
| 19 | $(xek \wedge \circ\mathrm{ia})$ | CP assumption |
| 20 | $(\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) \vee (xek \wedge \circ\mathrm{ia})$ | 19, $\vee$ introduction |
| 21 | $(\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) \vee (xek \wedge \circ\mathrm{ia})$ | 16–18, 19–20,<br>$\vee$ elimination |
| 22 | $rule_{\mathrm{read\_key\_pad}} \supset$<br>$(\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) \vee (xek \wedge \circ\mathrm{ia})$ | 14–21, $\supset$ introduction |
| 23 | $\mathrm{dps} ; ((\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) \vee (xek \wedge \circ\mathrm{ia})) ;$<br>$(((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$<br>$\vee ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$<br>$\vee (\neg xpl \wedge \circ\mathrm{dis}))$ | 13, 22,<br>ChopSwapImp3 |
| 24 | $(\mathrm{dps} ; (\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) \vee \mathrm{dps} ; (xek \wedge \circ\mathrm{ia})) ;$<br>$(((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$<br>$\vee ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$<br>$\vee (\neg xpl \wedge \circ\mathrm{dis}))$ | 23, ChopOrEqv |
| 25 | $\mathrm{dps} ; (\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) ;$<br>$(((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$<br>$\vee ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$<br>$\vee (\neg xpl \wedge \circ\mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ;$<br>$(((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$<br>$\vee ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$<br>$\vee (\neg xpl \wedge \circ\mathrm{dis}))$ | 24, OrChopEqv |
| 26 | $\mathrm{dps} ; (\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}}) ;$<br>$(((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$<br>$\vee ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$<br>$\vee (\neg xpl \wedge \circ\mathrm{dis}))$ | CP assumption |
| 27 | $\mathrm{dps} ; (\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$ | 26, semantics of chop |

| | | |
|---|---|---|
| 28 | $\mathrm{dps} ; (\neg xek \wedge \circ\mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek ; \neg xpl) \wedge (\circ\mathrm{ia} ; \circ\mathrm{dis}))$ | 27, $\vee$ introduction |
| 29 | $\mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ; (((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$<br>$\vee\ ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$<br>$\vee\ (\neg xpl \wedge \circ\mathrm{dis}))$ | 28, CP assumption |
| 30 | $\mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ; ((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$<br>$\vee\ \mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ; ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$<br>$\vee\ \mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ; (\neg xpl \wedge \circ\mathrm{dis})$ | 29, ChopOrEqv |
| 31 | $\mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ; ((xpl \wedge \circ xpm) \wedge \circ\circ\mathrm{pv})$ | CP assumption |
| 32 | $\mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{pv}))$ | 31, TwoChopRulesImp4 |
| 33 | $\mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek ; \neg xpl) \wedge (\circ\mathrm{ia} ; \circ\mathrm{dis}))$ | 32, $\vee$ introduction |
| 34 | $\mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ; ((xpl \wedge \circ\neg xpm) \wedge \circ\circ\mathrm{dis})$ | CP assumption |
| 35 | $\mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{dis}))$ | 34, TwoChopRulesImp4 |
| 36 | $\mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek ; \neg xpl) \wedge (\circ\mathrm{ia} ; \circ\mathrm{dis}))$ | 35, $\vee$ introduction |
| 37 | $\mathrm{dps} ; (xek \wedge \circ\mathrm{ia}) ; (\neg xpl \wedge \circ\mathrm{dis})$ | CP assumption |
| 38 | $\mathrm{dps} ; ((xek ; \neg xpl) \wedge (\circ\mathrm{ia} ; \circ\mathrm{dis}))$ | 36, TwoChopRulesImp4 |
| 39 | $\mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ\mathrm{ia} ; \circ\circ\mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek ; \neg xpl) \wedge (\circ\mathrm{ia} ; \circ\mathrm{dis}))$ | 38, $\vee$ introduction |

|  |  |  |
|---|---|---|
| 40 | $\mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis}))$ | 31–33, 34–36, 29–39,<br>$\vee$ elimination |
| 41 | $\mathrm{dps} ; (\neg xek \wedge \circ \mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis}))$ | 40, $\vee$ introduction |
| 42 | $\mathrm{dps} ; (\neg xek \wedge \circ \mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis}))$ | 26–28, 29–41,<br>$\vee$ elimination |
| 43 | $\mathrm{pp}$ | CP assumption |
| 44 | $\mathrm{dps} ; (\neg xek \wedge \circ \mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis}))$ | 42, reiteration |
| 45 | $\mathrm{pp} \supset (\mathrm{dps} ; (\neg xek \wedge \circ \mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis})))$ | 44, $\supset$ introduction |
| 46 | $\circ \mathrm{pp} \supset \circ(\mathrm{dps} ; (\neg xek \wedge \circ \mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis})))$ | 45, NextImpNext |
| 47 | $\circ \mathrm{pp} \supset (\circ \mathrm{dps} ; (\neg xek \wedge \circ \mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \circ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \circ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \circ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis})))$ | 46, NextOrDistEqv |
| 48 | $\circ \mathrm{pp} ; rule_{\mathrm{pin\_entry}} \supset (\circ \mathrm{dps} ; (\neg xek \wedge \circ \mathrm{kb} ; rule_{\mathrm{read\_key\_pad}})$<br>$\vee\ \circ \mathrm{dps} ; ((xek ; (xpl \wedge \circ xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{pv}))$<br>$\vee\ \circ \mathrm{dps} ; ((xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ \mathrm{ia} ; \circ\circ \mathrm{dis}))$<br>$\vee\ \circ \mathrm{dps} ; ((xek\ ; \neg xpl\,) \wedge (\circ \mathrm{ia} ; \circ \mathrm{dis}))) ; rule_{\mathrm{pin\_entry}}$ | 47, LeftChopImpChop |

| | |
|---|---|
| $le_{\text{pin\_entry}}$ | 1, reiteration |
| $(\neg xal \wedge \neg xvp) \wedge \text{opp}) ; rule_{\text{pin\_entry}}))$ <br> $(xvp \wedge \text{empty}) \vee (xal \wedge \text{empty})$ | 1, 49, eqv. subst. |
| $((\neg xal \wedge \neg xvp) \wedge \text{opp}) ; rule_{\text{pin\_entry}}$ | CP assumption |
| $(\neg xal \wedge \neg xvp) \wedge (\text{opp} ; rule_{\text{pin\_entry}})$ | , StateAndChop |
| $\text{opp} ; rule_{\text{pin\_entry}}$ | $\wedge$ elimination |
| $(\text{odps} ; (\neg xek \wedge \text{okb} ; rule_{\text{read\_key\_pad}})$ <br> $\vee \text{odps} ; ((xek ; (xpl \wedge \text{o}xpm)) \wedge (\text{oia} ; \text{oopv}))$ <br> $\vee \text{odps} ; ((xek ; (xpl \wedge \text{o}\neg xpm)) \wedge (\text{oia} ; \text{oodis}))$ <br> $\vee \text{odps} ; ((xek \; ; \neg xpl) \wedge (\text{oia} ; \text{odis}))) ; rule_{\text{pin\_entry}}$ | MP |
| $\text{odps} ; (\neg xek \wedge \text{okb} ; rule_{\text{read\_key\_pad}}) ; rule_{\text{pin\_entry}}$ <br> $\vee \text{odps} ; ((xek ; (xpl \wedge \text{o}xpm)) \wedge (\text{oia} ; \text{oopv})) ;$ <br> $rule_{\text{pin\_entry}}$ <br> $\vee \text{odps} ; ((xek ; (xpl \wedge \text{o}\neg xpm)) \wedge (\text{oia} ; \text{oodis}));$ <br> $rule_{\text{pin\_entry}}$ <br> $\vee \text{odps} ; ((xek \; ; \neg xpl) \wedge (\text{oia} ; \text{odis})) ; rule_{\text{pin\_entry}}$ | OrChopEqv |
| $(\neg xal \wedge \neg xvp)$ | $\wedge$ elimination |
| $(\neg xal \wedge \neg xvp) \wedge$ <br> $(\text{odps} ; (\neg xek \wedge \text{okb} ; rule_{\text{read\_key\_pad}}) ; rule_{\text{pin\_entry}}$ <br> $\vee \text{odps} ; ((xek ; (xpl \wedge \text{o}xpm)) \wedge (\text{oia} ; \text{oopv})) ;$ <br> $rule_{\text{pin\_entry}}$ <br> $\vee \text{odps} ; ((xek ; (xpl \wedge \text{o}\neg xpm)) \wedge (\text{oia} ; \text{oodis}));$ <br> $rule_{\text{pin\_entry}}$ <br> $\vee \text{odps} ; ((xek \; ; \neg xpl) \wedge (\text{oia} ; \text{odis})) ; rule_{\text{pin\_entry}})$ | $\wedge$ introduction |
| $((\neg xal \wedge \neg xvp) \wedge (\text{odps} ; (\neg xek$ <br> $\wedge \text{okb} ; rule_{\text{read\_key\_pad}}) ; rule_{\text{pin\_entry}}))$ <br> $\vee ((\neg xal \wedge \neg xvp) \wedge (\text{odps} ; ((xek ; (xpl \wedge \text{o}xpm))$ <br> $\wedge (\text{oia} ; \text{oopv})) ; rule_{\text{pin\_entry}}))$ <br> $\vee ((\neg xal \wedge \neg xvp) \wedge (\text{odps} ; ((xek ; (xpl \wedge \text{o}\neg xpm))$ <br> $\wedge (\text{oia} ; \text{oodis})); rule_{\text{pin\_entry}}))$ <br> $\vee ((\neg xal \wedge \neg xvp) \wedge (\text{odps} ; ((xek \; ; \neg xpl)$ <br> $\wedge (\text{oia} ; \text{odis})) ; rule_{\text{pin\_entry}}))$ | dist. of $\wedge$ over $\vee$ |

op

on

lesImp2

n

ın

esImp2

ı

| | | |
|---|---|---|
| 66 | $((\neg xal \wedge \neg xvp) \wedge \circ\text{dps}) ; ((xek ; (xpl \wedge \circ\neg xpm))$<br>$\wedge (\circ\text{ia} ; \circ\circ\text{dis})) ; rule_{\text{pin\_entry}}$ | CP assumption |
| 67 | $((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\circ\text{dis} ; rule_{\text{pin\_entry}})$ | TwoChopRulesImp2 |
| 68 | $(((\neg xal \wedge \neg xvp) ; \neg xek) \wedge$<br>$(\circ\text{dps} ;\circ\text{kb} ; rule_{\text{read\_key\_pad}} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ xpm)) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\circ\text{pv} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\circ\text{dis} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek \ ; \neg xpl ) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\text{dis} ; rule_{\text{pin\_entry}}))$ | $\vee$ introduction |
| 69 | $((\neg xal \wedge \neg xvp) \wedge \circ\text{dps}) ; ((xek \ ; \neg xpl )$<br>$\wedge (\circ\text{ia} ; \circ\text{dis})) ; rule_{\text{pin\_entry}}$ | CP assumption |
| 70 | $((\neg xal \wedge \neg xvp) ; xek \ ; \neg xpl ) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\text{dis} ; rule_{\text{pin\_entry}})$ | TwoChopRulesImp2 |
| 71 | $(((\neg xal \wedge \neg xvp) ; \neg xek) \wedge$<br>$(\circ\text{dps} ;\circ\text{kb} ; rule_{\text{read\_key\_pad}} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ xpm)) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\circ\text{pv} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\circ\text{dis} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek \ ; \neg xpl ) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\text{dis} ; rule_{\text{pin\_entry}}))$ | $\vee$ introduction |
| 72 | $(((\neg xal \wedge \neg xvp) ; \neg xek) \wedge$<br>$(\circ\text{dps} ;\circ\text{kb} ; rule_{\text{read\_key\_pad}} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ xpm)) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\circ\text{pv} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\circ\text{dis} ; rule_{\text{pin\_entry}}))$<br>$\vee (((\neg xal \wedge \neg xvp) ; xek \ ; \neg xpl) \wedge$<br>$(\circ\text{dps} ; \circ\text{ia} ; \circ\text{dis} ; rule_{\text{pin\_entry}}))$ | $\vee$ elimination |

| | | |
|---|---|---|
| 73 | $(((\neg xal \wedge \neg xvp) ; \neg xek) \wedge (\circ dps ; \circ kb ; rule_{read\_key\_pad} ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ xpm)) \wedge (\circ dps ; \circ ia ; \circ\circ pv ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ dps ; \circ ia ; \circ\circ dis ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; \neg xpl) \wedge (\circ dps ; \circ ia ; \circ dis ; rule_{pin\_entry})) \vee (xvp \wedge \text{empty}) \vee (xal \wedge \text{empty})$ | $\vee$ introduction |
| 74 | $(xvp \wedge \text{empty})$ | CP assumption |
| 75 | $(((\neg xal \wedge \neg xvp) ; \neg xek) \wedge (\circ dps ; \circ kb ; rule_{read\_key\_pad} ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ xpm)) \wedge (\circ dps ; \circ ia ; \circ\circ pv ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ dps ; \circ ia ; \circ\circ dis ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; \neg xpl) \wedge (\circ dps ; \circ ia ; \circ dis ; rule_{pin\_entry})) \vee (xvp \wedge \text{empty}) \vee (xal \wedge \text{empty})$ | $\vee$ introduction |
| 76 | $(xal \wedge \text{empty})$ | CP assumption |
| 77 | $(((\neg xal \wedge \neg xvp) ; \neg xek) \wedge (\circ dps ; \circ kb ; rule_{read\_key\_pad} ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ xpm)) \wedge (\circ dps ; \circ ia ; \circ\circ pv ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge (\circ dps ; \circ ia ; \circ\circ dis ; rule_{pin\_entry})) \vee (((\neg xal \wedge \neg xvp) ; xek ; \neg xpl) \wedge (\circ dps ; \circ ia ; \circ dis ; rule_{pin\_entry})) \vee (xvp \wedge \text{empty}) \vee (xal \wedge \text{empty})$ | $\vee$ introduction |

78 $(((\neg xal \wedge \neg xvp) ; \neg xek) \wedge$ (○dps ;○kb ; $rule_{\text{read\_key\_pad}}$ ; $rule_{\text{pin\_entry}}$)) $\vee$ elimination

$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ xpm)) \wedge$

(○dps ; ○ia ; ○○pv ; $rule_{\text{pin\_entry}}$))

$\vee (((\neg xal \wedge \neg xvp) ; xek ; (xpl \wedge \circ\neg xpm)) \wedge$

(○dps ; ○ia ; ○○dis ; $rule_{\text{pin\_entry}}$))

$\vee (((\neg xal \wedge \neg xvp) ; xek ; \neg xpl ) \wedge$

(○dps ; ○ia ; ○dis ; $rule_{\text{pin\_entry}}$))

$\vee$ ($xvp \wedge$ empty)

$\vee$ ($xal \wedge$ empty)

Based on these transformations, the possible behaviors associated with $rule_{\text{pin\_entry}}$ are:

(((¬*attempt_limit* ∧ ¬*valid_pin*) ; ¬*enter_key*)
∧ (○display_pin_screen ;○key_buffer ;
$rule_{\text{read\_key\_pad}}$ ; $rule_{\text{pin\_entry}}$))

∨ (((¬*attempt_limit* ∧ ¬*valid_pin*) ; *enter_key* ;
(*pin_length* ∧ ○*pin_match*))
∧ (○display_pin_screen ; ○increment_attempt ;
○○pin_valid ; $rule_{\text{pin\_entry}}$))

∨ (((¬*attempt_limit* ∧ ¬*valid_pin*) ; *enter_key* ;
(*pin_length* ∧ ○¬*pin_match*))
∧ ○display_pin_screen ; ○increment_attempt ;
○○display_invalid_screen ; $rule_{\text{pin\_entry}}$))

∨ (((¬*attempt_limit* ∧ ¬*valid_pin*) ; *enter_key* ; ¬*pin_length*)
∧ (○display_pin_screen ; ○increment_attempt ;
○display_invalid_screen ; $rule_{\text{pin\_entry}}$))

∨ (*valid_pin* ∧ empty)

∨ (*attempt_limit* ∧ empty)

# Appendix F

## Creating Rules to Describe a Simple Hardware System

In this appendix, rules are used to describe the behavior of a simple hardware system. Consider the simple NOR-based flip-flop system (Feynman, 1996) presented in Figure F-1.

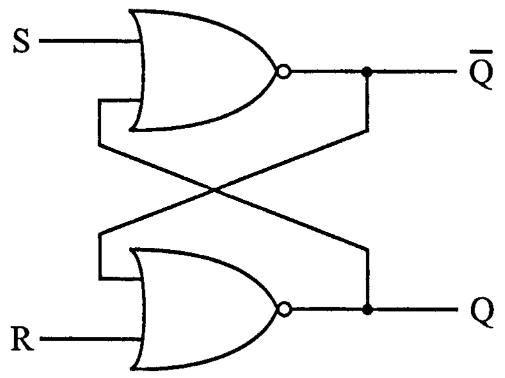


Figure F-1: A Simple Flip-Flop

Depending on the current state of Q and the values of the Set (S) and Reset (R) lines, the next state of Q is specified. The behavior of this simple flip-flop is described in Table F-1.

Table F-1 Behavior of a simple flip-flop

| Current Q | Set (S) | Reset (R) | Next Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

For the purposes of this exercise, the two possible cases where both Set (S) and Reset (R) are equal to one are undefined. Alternatively, the behavior of this simple flip-flop is described using the ITL next operator in Table F-2.

Table F-2 Behavior of a simple flip-flop expressed in ITL

| Q | S | R | $\circ$Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Under the system behavior as defined in Table F-2, each of the four conditions must hold for each of the six cases. Therefore, six general-form rules can be composed to describe the behavior of the simple flip-flip presented in Figure F-1:

$$(\mathrm{Q}=0 \wedge \mathrm{S}=0 \wedge \mathrm{R}=0) \wedge (\circ\mathrm{Q}=0) \tag{F-1a}$$

$$(\mathrm{Q}=0 \wedge \mathrm{S}=0 \wedge \mathrm{R}=1) \wedge (\circ\mathrm{Q}=0) \tag{F-1b}$$

$$(\mathrm{Q}=0 \wedge \mathrm{S}=1 \wedge \mathrm{R}=0) \wedge (\circ\mathrm{Q}=1) \tag{F-1c}$$

$$(\mathrm{Q}=1 \wedge \mathrm{S}=0 \wedge \mathrm{R}=0) \wedge (\circ\mathrm{Q}=1) \tag{F-1d}$$

$$(\mathrm{Q}=1 \wedge \mathrm{S}=0 \wedge \mathrm{R}=1) \wedge (\circ\mathrm{Q}=0) \tag{F-1e}$$

$$(\mathrm{Q}=1 \wedge \mathrm{S}=1 \wedge \mathrm{R}=0) \wedge (\circ\mathrm{Q}=1) \tag{F-1f}$$

Given that the domain of R and S are {0,1}, the following definitions are made:

$$x \triangleq (\mathrm{Q}=1) \text{ and } \neg x \triangleq (\mathrm{Q}=0) \tag{F-2a}$$
$$y \triangleq (\mathrm{R}=1) \text{ and } \neg y \triangleq (\mathrm{R}=0) \tag{F-2b}$$
$$z \triangleq (\mathrm{S}=1) \text{ and } \neg z \triangleq (\mathrm{S}=0) \tag{F-2c}$$

Substituting the definitions at (F-2) into (F-1) yields:

$$(\neg x \wedge \neg z \wedge \neg y) \wedge (\circ\neg x) \tag{F-3a}$$

$$(\neg x \wedge \neg z \wedge y) \wedge (\circ\neg x) \tag{F-3b}$$

$$(\neg x \wedge z \wedge \neg y) \wedge (\circ x) \tag{F-3c}$$

$$(x \wedge \neg z \wedge \neg y) \wedge (\circ x) \tag{F-3d}$$

$$(x \wedge \neg z \wedge y) \wedge (\circ\neg x) \tag{F-3e}$$

$$(x \wedge z \wedge \neg y) \wedge (\circ x) \tag{F-3f}$$

These six individual rules can be combined disjunctively to form a single rule-base structure that describes the behavior of the flip-flop system:

$$\begin{aligned}
&(\neg x \wedge \neg z \wedge \neg y \wedge \circ\neg x)\\
&\vee (\neg x \wedge \neg z \wedge y \wedge \circ\neg x)\\
&\vee (\neg x \wedge z \wedge \neg y \wedge \circ x)\\
&\vee (x \wedge \neg z \wedge \neg y \wedge \circ x)\\
&\vee (x \wedge \neg z \wedge y \wedge \circ\neg x)\\
&\vee (x \wedge z \wedge \neg y \wedge \circ x)
\end{aligned} \tag{F-4}$$

Consider the following pair of disjunctively connected rules:

$$(\neg x \wedge \neg z \wedge y \wedge \circ\neg x) \vee (x \wedge \neg z \wedge y \wedge \circ\neg x) \tag{F-5}$$

Applying propositional logic to (F-5) yields the equivalent expression:

$$(\neg x \wedge x) \vee (\neg z \wedge y \wedge \circ\neg x) \tag{F-6}$$

Applying propositional logic to (F-6) yields the equivalent expression:

$$(\neg z \wedge y \wedge \circ\neg x) \tag{F-7}$$

Combining (F-5), (F-6), and (F-7) yields:

$$(\neg x \wedge \neg z \wedge y \wedge \circ\neg x) \vee (x \wedge \neg z \wedge y \wedge \circ\neg x) \equiv (\neg z \wedge y \wedge \circ\neg x) \tag{F-8}$$

Applying the equivalence (F-8) to (F-4) yields:

$$\begin{aligned}
&(\neg x \wedge \neg z \wedge \neg y \wedge \circ\neg x)\\
&\vee (\neg x \wedge z \wedge \neg y \wedge \circ x)\\
&\vee (x \wedge \neg z \wedge \neg y \wedge \circ x)\\
&\vee (x \wedge z \wedge \neg y \wedge \circ x)\\
&\vee (\neg z \wedge y \wedge \circ\neg x)
\end{aligned} \tag{F-9}$$

Consider the following pair of disjunctively connected rules:

$$(\neg x \wedge z \wedge \neg y \wedge \circ x) \vee (x \wedge z \wedge \neg y \wedge \circ x) \tag{F-10}$$

Applying propositional logic to (F-10) yields the equivalent expression:

$$(\neg x \wedge x) \vee (z \wedge \neg y \wedge \circ x) \tag{F-11}$$

Applying propositional logic to (F-11) yields the equivalent expression:

$$(z \wedge \neg y \wedge \circ x) \tag{F-12}$$

Combining (F-10), (F-11), and (F-12) yields:

$$(\neg x \wedge z \wedge \neg y \wedge \circ x) \vee (x \wedge z \wedge \neg y \wedge \circ x) \equiv (z \wedge \neg y \wedge \circ x) \tag{F-13}$$

Applying the equivalence (F-13) to (F-9) yields:

$$\begin{aligned}&(\neg x \wedge \neg z \wedge \neg y \wedge \circ\neg x)\\ &\vee (x \wedge \neg z \wedge \neg y \wedge \circ x)\\ &\vee (z \wedge \neg y \wedge \circ x)\\ &\vee (\neg z \wedge y \wedge \circ\neg x)\end{aligned} \tag{F-14}$$

Applying the definitions presented at (F-2) to (F-14) yields:

$$\begin{aligned}&(\mathrm{Q} = 0 \wedge \mathrm{S} = 0 \wedge \mathrm{R} = 0 \wedge \circ\mathrm{Q} = 0)\\ &\vee (\mathrm{Q} = 1 \wedge \mathrm{S} = 0 \wedge \mathrm{R} = 0 \wedge \circ\mathrm{Q} = 1)\\ &\vee (\mathrm{S} = 1 \wedge \mathrm{R} = 0 \wedge \circ\mathrm{Q} = 1)\\ &\vee (\mathrm{S} = 0 \wedge \mathrm{R} = 1 \wedge \circ\mathrm{Q} = 0)\end{aligned} \tag{F-15}$$

Remembering that the domain of Q is {0, 1}, consider the following definitions:

$$\circ\mathrm{Q}_{unchanged} \triangleq (\mathrm{Q} = 1 \wedge \circ\mathrm{Q} = 1) \tag{F-16a}$$
$$\circ\mathrm{Q}_{unchanged} \triangleq (\mathrm{Q} = 0 \wedge \circ\mathrm{Q} = 0) \tag{F-16b}$$

Applying these definitions to (F-16) yields:

$$\begin{aligned}&(S = 0 \wedge R = 0 \wedge \circ Q_{unchanged})\\ &\vee (S = 0 \wedge R = 0 \wedge \circ Q_{unchanged})\\ &\vee (S = 1 \wedge R = 0 \wedge \circ Q = 1)\\ &\vee (S = 0 \wedge R = 1 \wedge \circ Q = 0)\end{aligned} \qquad \text{(F-17)}$$

Applying propositional logic to (F-17) yields:

$$\begin{aligned}&(S = 0 \wedge R = 0 \wedge \circ Q_{unchanged})\\ &\vee (S = 1 \wedge R = 0 \wedge \circ Q = 1)\\ &\vee (S = 0 \wedge R = 1 \wedge \circ Q = 0)\end{aligned} \qquad \text{(F-18)}$$

With these transformations, the original six rules of (F-1) and the corresponding six-rule disjunctive structure of (F-4) have been reduced to three rules expressed as a disjunctive structure at (F-18). (F-18) describes the behavior of the hardware system presented in Figure F-1 and described in Table F-1. Although purposefully limited in scope, this example is a demonstration of how rules can be formed using this rule model and rule algebra to describe a given system.

These rules can be used to reason about, analyze, and/or understand the target system. For example, in comparison with the information conveyed in Figure F-1 and Table F-1, these rules provide a clear and succinct description of the system behavior: if Set (S) is high and Reset (R) is low, the next Q is set high; if Set (S) is low and Reset (R) is high, the next Q is set low; and if both Set (S) and Reset (R) are low, the next Q is unchanged from its current status. Whereas this information can be gleaned from Figure F-1 and Table F-1, the rules in (F-18) provide a simple and immediately comprehensible depiction of system behavior. As these rules have been developed using the rule model and rule algebra presented in this research, these rules can be integrated into a larger rule system, as appropriate, using the rule algebra.