Characterizing Model Transformations Using Modal μ-calculus

Chaki, Rikayan

Award date:
2021

Technische Universiteit Eindhoven University of Technology

Department of Mathematics and Computer Science
Software Engineering and Technology Research Group

Characterizing Model Transformations Using Modal μ-calculus

Master Thesis

Rikayan Chaki

Student Number : 1473689

Supervisor:
Dr.ing. Anton Wijs

Committee Members:
Prof. Alexander Serebrenik
Dr.ir. Tim A.C. Willemse

Final version

Eindhoven, September 2021

Abstract

The subject of analyzing software can be primarily divided into two kinds - formal and informal. Formal modeling techniques, which are the focus of this thesis, rely on using techniques whose syntax and semantics are defined with precision. The use of these techniques can allow, for instance, verification of the satisfiability of a property by a model. In addition, it can also help in refining a model by transforming it using well-defined model transformation rules. This is a use case that commonly happens whenever a system evolves, or, during its design process. These aspects of formal modeling, in particular, are used throughout this thesis. The area of our work is one of active research. Works that utilize formal models vary in the kind of model that they use, ranging from Labeled Transition Systems (LTSs) to Families of Deterministic Finite Automata (FDFAs), each having their own unique benefits and drawbacks. There is a similar variety in the formal logics that provide specifications to these formal models. Combining these two ideas - models and formal logics, one can also arrive at the topic of property verification, which is concerned with checking if a given property is satisfied by a model. Finally, the topic that is closest to this work is that of property-preserving model verification. This aims to find whether or not the satisfiability of a property is preserved when a given model transformation is applied on some model. These aspects of research are highlighted in this work, providing the reasoning for the choices that were made therein. Coming to our work, the focus is on analyzing a model transformation which is applied on a model that satisfies a given property. With this analysis, the goal is to find some characteristics of the model that results from the transformation. In order to achieve this goal, three major steps are taken. Firstly, an abstract representation of the system model is considered, which takes into account the fact that the given model transformation is satisfied by the same. This model is then refined by utilizing the knowledge that the source model satisfies the given property. The resulting model is transformed by applying the given transformation rule on it. Finally, properties characterizing this model are extracted. Using these characteristic properties, one can make inferences about the transformed model, without having to actually construct it. This can, in some cases, mean a massive amount of time is saved, as the original source and target models are often much larger than the abstracted models which are used throughout these process.

Preface

This thesis is a culmination of almost seven months of efforts that was performed as the concluding work of the Master's of Computer Science and Engineering program at the Eindhoven University of Technology.

Primarily, I would like to thank my supervisor, Prof. Anton Wijs. His guidance throughout this project has been instrumental. Due to the COVID-19 pandemic, a significant portion of the collaboration between myself and Prof. Wijs had to be done in a remote manner. Despite this, I can confidently say that, without your comments and suggestions, my work surely would have stagnated at some point.

In addition to this, the guidance and support of my family has been great as well. Their support has certainly motivated me and provided a great deal of satisfaction during the process of working on this thesis.

Contents

List of Figures

Chapter 1

Introduction

A popular approach to software engineering is the *model-driven* approach [8] [9] [10]. This approach is primarily characterized by the use of models as the primary driving force that guides the software development process. Such models can be either formal or informal. Informal modeling approaches include languages such as the Unified Modeling Language(UML) [11], Systems Modeling Language(SysML) [12], and Business Process Modeling Language(BPML) [13]. They often provide simple and more flexible representations of artifacts in the software development process. However, the main drawback of these approaches lies within their semantics, which are informal. Thus, expert knowledge or interpretation is often being necessary to understand these kinds of models. In addition, formal methods also support the task of verifying properties of a system when they are used. For these reasons, in this work, the focus is on using formal models, as well as on their verification.

The syntax and semantics of modeling paradigms such as LTSs [14] and Finite-State Automata [15] can be represented in a completely formal, unambiguous manner. These are examples of formal software modeling, and will play a key role in our work. When using techniques such as these, there is often a schema for representing any specifications (requirements) that need to be fulfilled by the system at hand. Such properties, also represented by using formal logics, can be automatically verified on systems that are represented by formal models. This is known as formal verification.

Suppose that there is a system being developed iteratively. This often happens when designing complex software, whenever creating a model from scratch is not feasible. Iterative approaches can be classified into two major kinds — the "top-down" and the "bottom-up" approach. In this thesis, we will focus on the top-down approach.

A top-down approach begins with a coarse view of the system model that is *refined* into increasingly feature-rich and concretely specified models. The refinements themselves are formalized using specialized models known as *model transformations.* A very efficient way in which a property of a system can be proven to *persist* across such a model transformation is to show that the property holds for the model transformation itself [3] [5], irrespective of the model on which it is applied.

At the beginning, as well as in these intermediate stages of development, it is often the case that the models are *incomplete* in some sense. This incompleteness may be due to either a lack of features required by stakeholders, or due to errors in existing facets of the system which require modifications to fix the same. Recently, there has been some progress made in dealing with such incomplete models [16] formally.

The work on incomplete model checking is used as the inspiration for the theme of merging new knowledge about a model to obtain a refined version of the same. In the works by Wijs et. al. [3] [4] [5], for instance, such incremental improvements were modeled by using model transformations. The main focus of those works was checking whether a (formal) property is preserved when a system is transformed.

In the case of our work, the intent is to help with cases where the evolution begins at the level of system requirements. This can happen, for instance, with changing interests of stakeholders. Suppose a requirement r_1 is modified into another, r_2. As such a change at the specification level is propagated through to the system level, those involved in this process often would like to understand the structure of the evolved system. For a system of non-trivial size, actually transforming the system results in a modified system that is very complex to analyze. Further, if the transformation itself is localized to a small part of the system, analyzing the entire system is also wasteful. In such a case, what is really needed is an approach that remains agnostic towards the system model, but still utilizes the information that such a model satisfied r_1 and that the given model transformation is applicable on it. This work aims to formalize this information into a property. This is a format that is easy to interpret for those that deal with requirements, that is, the requirements engineers that must design the new requirements for the system.

In this thesis, our focus will be to establish a counterpoint to this question by focusing on the characteristics of the result of the model transformation itself, instead of the property satisfied by the original system. What happens when a system that satisfies a property undergoes a model transformation? This question motivates our main *research goal*. The goal of our work is:

To design a procedure and build a proof-of-concept tool that generates a property characterizing a model transformation, given the knowledge that the source system on which said transformation is applied satisfies a given property.

This thesis is structured as follows. The second chapter gives some preliminary formalisms that are essential in understanding the remainder of the thesis. Following this, the third chapter contains a literature review for the related work. Then, in the fourth chapter, the techniques used to achieve the research objective are discussed. This is followed by the fifth and sixth chapters, where experimental results are described and subsequently analyzed in order to obtain inferences relevant for the research objective. Finally, the seventh chapter draws the conclusions that this work has enabled us to obtain. An appendix is also added at the end, containing miscellaneous formulae that did not find their place in the main body of the report.

1.1 Background and Broad Scope of Work

The motivation behind this thesis, as work on it began, was provided by works that focused on the evolution of the requirements of a software system as it evolved. There has been some amount of research generally in the field of software evolutions and its associated problems [17], [18], [19], [20]. In this section, two works are discussed that highlight this topic in further detail.

The work by Harker et. al. [17] highlights a top-down view of the problem of requirements evolution. Their work focuses on the issue of problem uncertainty in the context of requirements engineering. They classify requirements of a system based on whether or not they can change as the software evolves from an organizational, high-level perspective. If a requirement is allowed to change, it is further classified on why a requirement can change, with reasons such as stakeholder engagement, system use, (software)environmental turbulence, etc. They also discuss some techniques for handling such changes to requirements. Such ideas, however, appear to be quite outdated today. However, it still provided us with several scenarios in which requirements can be modified as a software is evolved. When one compounds the complexity of a modern software system onto this realization, it becomes abundantly clear that our goal of easing the task of the requirements engineer is rather relevant.

The work by McGee et. al. [18], on the other hand, focuses far more on proposing a potential solution to the problem/necessity that is requirements evolution. In order to achieve this, they create a complete taxonomy of the sources of changes to requirements, distinguishing specifically between factors that contribute to requirement uncertainty and those that will trigger a change in the requirement. From this perspective, our work focuses on the latter of these two issues. This is because we will look at system evolution (model transformations) where the original requirement (property) is not satisfied by the original model being transformed, and thus *must* be changed.

Chapter 2

Preliminaries

In this section of the report, the details of the tools and techniques that we used while striving towards our research goals are made apparent.

2.1 Labelled Transition System (LTS)

An LTS $\mathcal{G}$ is formally defined as a tuple (Q, Q_0, A, δ) where

1. Q is a set whose elements are known as *states* of $\mathcal{G}$.
2. Q_0 is a non-empty subset of Q, that is, $Q_0 \subseteq Q$. The members of Q_0 are the *initial states* of $\mathcal{G}$.
3. A is a set consisting of the *actions* of $\mathcal{G}$.
4. δ is a relation of the form $\delta \subseteq Q \times A \times Q$. It describes *transitions* with labels viz. $a \in A$ from $q \in Q$ that leads to $q' \in Q$. If $(q, a, q') \in \delta$, then this fact is usually represented in one of two ways:
   (a) $\delta(q, a) = q'$, or
   (b) $q \xrightarrow[\delta]{a} q'$

In subsequent sections of this report, a subscript-based notation will be used to refer to the attributes of LTSs. As an example of this, $\mathcal{G}_Q$, $\mathcal{G}_{Q_0}$, $\mathcal{G}_A$, and $\mathcal{G}_\delta$ refer to the set of states, starting states, action set, and transition relation of the LTS $\mathcal{G}$.

Informally, one can view LTSs as graphs with labeled transitions. In order to find the *language* of an LTS, an auxiliary concept is needed - *traces*. A trace of an LTS M is formally defined as a sequence

$$w = \epsilon \mid w_0\, w_1\, \dots\, w_n \text{ for some } n \in N \cup \{0\} \text{ or } n = \infty$$

Here ϵ is used to denote the empty trace. Further, in the context of this definition, for any non-empty trace $w = w_0 \dots w_n$, the following two constraints hold

1. $\forall_{i=0}^{n} w_i \in M_A$
2. $\exists_{q_0 \in M_{Q_0}, q_1, \dots, q_{n+1} \in M_Q} \forall_{i=0}^{n} q_i \xrightarrow[M_\delta]{w_i} q_{i+1}$

In other words, a trace of M records the sequence of actions that are the labels of a sequence of state transitions occurring in the context of M. In this report, $w(i)$ is used to denote the i-suffix of w, that is,

$$w(i) = w_i\, w_{i+1}, \dots w_n$$

The language of M is, then

$$L(M) = \{w \mid w \text{ is a trace of } M\}$$

2.2 Büchi Automata

A Büchi Automaton is formally defined as a 5-tuple (Q, Q_0, A, δ, F) where the definitions Q, Q_0, A, and, δ are the same as in the case of LTS. F is a subset of Q. Its elements are known as the *final states* of the automaton. The subscript notation introduced with LTSs will also be used in the context of Büchi Automata. Thus, for instance, $\mathcal{M}_F$ refers to the set of final states of a Büchi Automaton $\mathcal{M}$.

This addition of final states into the concept of Büchi automata allows a broader scope of languages to be representable in comparison to LTS. The major distinction between the semantics of an LTS and a Büchi Automaton is that when simulating any trace of a Büchi automaton, one must pass through at least one of the final states infinitely often. Note that this requirement implicitly adds the constraint that all traces of a Büchi Automata are infinite.

2.3 Formal Logics

While solving the research goal at hand, inevitably, the manner in which properties of a system are formalized is, obviously, quite crucial. In this section of the report, our aim is to focus our attention on the three kinds of such descriptions used at various stages of our work - Linear Temporal Logic (LTL), modal μ-calculus, and systems of modal μ-calculus equations.

2.3.1 Linear Temporal Logic (LTL)

The syntax of an LTL property is recursively defined as follows:

$$\phi = X\phi_1 \mid G\phi_1 \mid F\phi_1 \mid \phi_1 U \phi_2 \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid a$$

where ϕ_1 and ϕ_2 are LTL formulae, and a is an atomic proposition. For the purposes of this report, it is sufficient to consider an atomic proposition as being equivalent to a Boolean variable. One of the characteristics of an LTL formula is the set of atomic propositions, AP, that contextualizes these atomic propositions.

Suppose now that there exists a model M such that $M_A = \mathcal{P}(AP) \setminus \{\emptyset\}$, where $\mathcal{P}(AP)$ is the power-set — the set of all subsets, of AP. It is important to note here that in our work, the valuation $\emptyset$, which corresponds to all the atomic propositions evaluating to FALSE, is not present in M_A. The reason for this is as follows. It is considered that for a given transition, if an atomic proposition a is true, it indicates that a process corresponding to a occurs as the transition is crossed during a given run of the system. Therefore, a valuation in which all atomic propositions are false indicates non-progress in terms of all properties of the system except the state in the LTS that corresponds to it. In a sense, it is a hidden action that cannot be detected by an external observer. In order to preclude such a possibility from happening, $\emptyset$ is removed from M_A.

The semantics of an LTL property describe when it is satisfied by such a model M. Suppose that w is an infinite trace of M. Then, $\phi \models w$ signifies that a formula ϕ is satisfied by a trace w. An LTS M is said to satisfy ϕ if and only if all of its traces satisfy ϕ as well. Formally, this is indicated by the following assertion

$$\phi \models M \iff \forall_{w \in L(M)} \phi \models w$$

The recursive structure of ϕ can be used to describe its semantics. This description enables the resolution of the assertion $\phi \models w$ for some trace w. Below, a formal description of the semantics of LTL is provided. Note that it is assumed here that w_i is the i^{th} element of the trace w.

$$\begin{aligned}
X\phi \models w &\iff \phi \models w_1 \\
G\phi \models w &\iff \forall_{i\geq 0}\phi \models w_i \\
F\phi \models w &\iff \exists_{i\geq 0}\phi \models w_i \\
\phi_1 U \phi_2 \models w &\iff \exists_{j\geq 0}\left((\forall_{i<j}\phi_1 \models w_i) \ \wedge \ \phi_2 \models w_j\right) \\
\neg\phi \models w &\iff \neg(\phi \models w) \\
\phi_1 \wedge \phi_2 \models w &\iff (\phi_1 \models w) \wedge (\phi_2 \models w) \\
a \models w &\iff a \in w_0
\end{aligned}$$

Note that a proposition a is satisfied with respect to the set of actions M_A of M at the first state of the trace being verified against the proposition.

Verification of an LTS property

Given a system M and an LTL formula ψ, the task of verifying M with respect to ψ can be achieved by following the sequence of steps listed below

1. $\neg\psi$ is converted into a Büchi automaton $G_{\neg\psi}$ such that the language of $G_{\neg\psi}$, $L(G_{\neg\psi})$ contains exactly all those traces that satisfy $\neg\psi$. In other words, these are the traces that do not satisfy ψ. There are several methods for performing this transformation, such as a work by Babiak et. al. [21].

   The set of actions of this Büchi automaton is the same as that for M. However, it is often the case that the transitions on the labels of $G_{\neg\psi}$ are annotated with propositional logic formulae over AP, instead of sets of APs. A transition between two states, labelled with such a formula P, is semantically equivalent to transitions, each labeled with a set in $G_{\neg\psi,A}$ that satisfies P, between the same pair of states.

2. $G_{\neg\psi}$ is combined with M to form a product Büchi automaton $M \times G_{\neg\psi}$ with the property that
$$L(M \times G_{\neg\psi}) = L(M) \cap L(G_{\neg\psi})$$
   The model $M \times G_{\neg\psi}$ is defined as follows
$$\begin{aligned}
Q &= M_Q \times G_{\neg\psi,Q} \\
Q_0 &= M_{Q_0} \times G_{\neg\psi,Q_0} \\
A &= M_A \cap G_{\neg\psi,A}
\end{aligned}$$
$$\delta \subseteq Q \times A \times Q \text{ is defined as follows : } q \xrightarrow[M_\delta]{a} q' \iff$$
$$phi\exists_{r_1,r_2\in M_Q \wedge s_1,s_2 \in G_{\neg\psi,Q}} q = (r_1,s_1) \wedge q' = (r_2,s_2) \wedge$$
$$r_1 \xrightarrow[M_\delta]{a} r_2 \wedge s_1 \xrightarrow[G_{\neg\psi,\delta}]{a} s_2$$
$$F = \{(s,r) \mid s \in M_F \vee r \in G_{\neg\psi,F}\}$$
   where, given two sets S_1 and S_2,
$$S_1 \times S_2 = \{(x,y) \mid x \in S_1 \wedge y \in S_2\}$$

3. Finally, a test is performed to check whether the language of this product automaton, $L(M \times G_{\neg\psi})$, is empty. This is the case if and only if there does not exist any cycle in $M \times G_{\neg\psi}$

that contains a final state and is reachable from one of the initial states. If $L(M \times G_{\neg\psi})$ turns out to indeed be empty (that is, the set ϕ), then we have

$$\begin{aligned} L(M \times G_{\neg\psi}) = \phi &\implies L(M) \cap L(G_{\neg\psi}) = \phi \\ &\implies \nexists_{\text{Trace } T} T \in L(M) \wedge T \in L(G_{\neg\psi}) \\ &\implies \forall_{\text{Trace } T} T \in L(M) \longrightarrow T \notin L(G_{\neg\psi}) \end{aligned}$$

Since $L(G_{\neg\psi})$ contains precisely all those traces that do not satisfy ψ, a trace T that is *not* in $L(G_{\neg\psi})$ must necessarily satisfy ψ. Thus, we have

$$\forall_{\text{Trace } T} T \in L(M) \longrightarrow \psi \models T \implies M \text{ satisfies } \psi$$

2.3.2 Modal μ-calculus

As with LTL, the discussion of modal μ-calculi is separated into a discussion of its syntax, as well as semantics. The breadth of properties supported by it is larger than in the case of LTL. Therefore, both these aspects are comparatively more involved for this logic. For brevity's sake, henceforth, we will use the term "μ-calculus" when referring to modal μ-calculus. The syntax of a μ-calculus formula is defined recursively as follows:

$$\phi = \mathit{True} \mid \mathit{False} \mid \langle a\rangle\phi_1 \mid [a]\phi_1 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \mu X \,.\, \phi_1 \mid \nu X \,.\, \phi_1 \mid X$$

Note here that ϕ_1 and ϕ_2 are μ-calculus formulae, a is an action that can belong to the model on which this property is to be validated, and X is a *variable* that represents a set of states. Evaluating ϕ requires two additional components. Firstly, a model M on which the property is to be checked is required. As before, for such a model, $M_A = \mathcal{P}(AP) \setminus \{\emptyset\}$ must hold, where AP is a set of atomic propositions.

LTS is used as the model type in our analysis. Secondly, an *environment* variable η is needed to associate variables with concrete subsets of the set of states of M. That is,

$$\forall X \in \phi_{Variables}\, \eta(X) = S \subseteq M_Q \text{ or } \eta(X) \text{ is unknown}$$

Given this contextualization, the semantics of ϕ produces a set of states:

$$\begin{aligned} [\![\mathit{True}]\!]_{M,\eta} &= M_Q \\ [\![\mathit{False}]\!]_{M,\eta} &= \emptyset \\ [\![\langle a\rangle\phi_1]\!]_{M,\eta} &= \{s \in M_Q \mid \exists_{s' \in M_Q} s \xrightarrow[M.\delta]{a} s' \wedge s' \in [\![\phi_1]\!]_{M,\eta}\} \\ [\![[a]\phi_1]\!]_{M,\eta} &= \{s \in M_Q \mid \forall_{s' \in M_Q} s \xrightarrow[M.\delta]{a} s' \implies s' \in [\![\phi_1]\!]_{M,\eta}\} \\ [\![\phi_1 \wedge \phi_2]\!]_{M,\eta} &= [\![\phi_1]\!]_{M,\eta} \cap [\![\phi_2]\!]_{M,\eta} \\ [\![\phi_1 \vee \phi_2]\!]_{M,\eta} &= [\![\phi_1]\!]_{M,\eta} \cup [\![\phi_2]\!]_{M,\eta} \\ [\![\mu X \,.\, \phi_1]\!]_{M,\eta} &= \bigcap\{S \subseteq M_Q \mid S \supseteq [\![\phi_1]\!]_{M,\eta(X)=S}\} \\ [\![\nu X \,.\, \phi_1]\!]_{M,\eta} &= \bigcup\{S \subseteq M_Q \mid S \subseteq [\![\phi_1]\!]_{M,\eta(X)=S}\} \\ [\![X]\!]_{M,\eta} &= \eta(X) \end{aligned}$$

For this work, a small upgrade is made to the syntax of the actions that can reside in the box and diamond operators. Instead of just being single actions, the contents of a box or a diamond operator, in this extended form, are also allowed to contain propositional logic over the set AP. Formally, for a given propositional logic formula P, the semantics of this form can be defined as follows

$$[\![\langle P\rangle\phi_1]\!]_{M,\eta} = \{s \in M_Q \mid \exists_{s' \in M_Q} \forall_{a \in M_A} P \text{ is satisfied by } a \implies s \xrightarrow[M.\delta]{a} s' \wedge s' \in [\![\phi_1]\!]_{M,\eta}\}$$

$$[\![[P]\phi_1]\!]_{M,\eta} = \{s \in M_Q \mid \forall_{s' \in M_Q} \forall_{a \in M_A} s \xrightarrow[M.\delta]{a} P \text{ is satisfied by } a \implies \Big(s' \implies s' \in [\![\phi_1]\!]_{M,\eta}\Big)\}$$

Comparing these definitions with the definitions of the regular form of the diamond and box operators, the following equivalences can be derived

$$\begin{aligned}
[\![\langle P\rangle\phi_1]\!]_{M,\eta} &= \bigwedge_{a\in M_A \wedge P \text{ is satisfied by } a} [\![\langle a\rangle\phi_1]\!]_{M,\eta} \\
[\![[P]\phi_1]\!]_{M,\eta} &= \bigwedge_{a\in M_A \wedge P \text{ is satisfied by } a} [\![[a]\phi_1]\!]_{M,\eta}
\end{aligned}$$

While dealing with the semantics of μ-calculus formulae, it is important to distinguish between *open* and *closed* formulae. Open formulae are characterized by the existence of variables that are not *bound* to a μ or ν-term. In contrast, for a closed formula, all the variables involved are bound. This means that in order to evaluate an open formula ϕ, η must be initialized with the values of all the non-bound variables in ϕ. In any case, a formula is said to satisfy a model M if its interpretation with respect to this M produces a set of states that contains all of its initial states.

For a closed formula, this means that the following relation holds if and only if M satisfies ϕ

$$M_{Q_0} \subseteq [\![\phi]\!]_{M,\phi}$$

2.3.3 System of modal μ-calculus equations

The syntax of a system of modal μ-calculus equations X is as follows

$$\begin{aligned}
S_0 &= \phi_0 \\
S_1 &= \phi_1 \\
&\vdots \\
S_n &= \phi_n
\end{aligned}$$

where $\phi_0, \phi_1, \ldots, \phi_n$ are μ-calculus formulae. For any $i \in \{0, \ldots, n\}$, the formula ϕ_i can use any of the variables in $S_0, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n$ as unbounded variables in its construction. However, the variable S_i must be bounded, if present in ϕ_i. A model M is said to satisfy X if and only if the all of its initial states are contained in the set corresponding to the variable of the first equation, upon evaluating the system with respect to M. In other words, the following condition must hold

$$M_{Q_0} \subseteq \eta S_0$$

where $\eta(S_0), \ldots, \eta(S_n)$ are defined such that

$$\forall_{i\in\{0,1,\ldots,n\}} \eta(S_i) = [\![\phi_i]\!]_{M,\eta}$$

2.4 Model Transformation

A model transformation, as evident by the name, is the process of transforming a model into another by following a set schema. There are several approaches that can help specify such a process formally. In particular, the approach used here is adapted from a work by Wijs et. al. [6].

In this context, the core element of a model transformation is a rule. A rule R is defined as a pair of *pattern LTSs*

$$R = (\mathcal{L}, \mathcal{R})$$

A crucial part of the transformation process relies on matching the states that are common to both $\mathcal{L}$ and $\mathcal{R}$, that is, states in $\mathcal{L}_Q \cap \mathcal{R}_Q$. These states are termed as *glue states*. In addition, a pattern LTS such as $\mathcal{L}$ is also annotated with in-states ($\mathcal{L}_I$) and exit-states ($\mathcal{L}_E$). These sets are related as follows

$$\text{Glue states of } \mathcal{L}, \mathcal{R} = \mathcal{L}_Q \cap \mathcal{R}_Q = \mathcal{L}_I \cup \mathcal{L}_E = \mathcal{R}_I \cup \mathcal{R}_E$$

An in-state is a glue state that can only be matched on system states with only outgoing transitions that are also matched by the pattern. Conversely, an exit-state can be matched by a system state that has only incoming transitions that are also matched by the pattern. The set of initial states of a pattern LTS is the same as its in-states. This is because any of these states *could* be matched with an initial state of the system LTS on which the pattern is applied (or in the resulting LTS after the transformation) can be an initial state of the system.

Rules may be applied on system LTSs to produce transformed versions of the same. A rule R is said to be applicable on a given system LTS M if and only if the set of states of $\mathcal{L}$ can be mapped onto a set of states of M with a mapping $m\colon \mathcal{L}_Q \to M_Q$, such that the following condition holds on m, for all states $q \in \mathcal{L}_Q$

$$\forall_{a \in \mathcal{L}_A, q' \in \mathcal{L}_Q} \exists_{r \in M_Q} m(q) = r \wedge q \xrightarrow[\mathcal{L}_\delta]{a} q' \iff \exists_{r' \in M_Q} r \xrightarrow[M_\delta]{a} r' \wedge m(q') = r'$$

It may be the case that more than one occurrence of such matches happen in M. In such a case, it is assumed that these matching instances are disjoint. That is, for all matches m, m' in this context, the following assertion holds true

$$\forall_{q,q' \in \mathcal{L}_Q} m(q) \neq m'(q')$$

Once these relevant matches are identified, the states that they refer to are replaced by using the double pushout graph rewriting method. This method proceeds by using the glue states as anchors and replacing the non-glue states of $\mathcal{L}$ by non-glue states of $\mathcal{R}$, in a manner that preserves the relationship between the glue and non-glue states. Given a match m, the following technique is adopted, with *each step being repeated* as long as changes are happening to M by performing that step.

1. If $q \in \mathcal{L}_I$, and $m(q) \xrightarrow[M_\delta]{a} m(q')$ for some $a \in M_A$ and q' is a non-glue state, then delete this transition.
2. If $q \in \mathcal{L}_I$, $m(q) \xrightarrow[M_\delta]{a} r'$ for some $a \in M_A$, and r' is a non-glue state of $\mathcal{R}$, then add this transition along with the state r' to M. Also, add a to the set M_A.
3. If $q \in \mathcal{L}_E$, and $m(q') \xrightarrow[M_\delta]{a} m(q)$ for some $a \in M_A$ and q' is a non-glue state, then delete this transition.
4. If $q \in \mathcal{L}_E$, $r' \xrightarrow[M_\delta]{a} m(q)$ for some $a \in M_A$, and r' is a non-glue state of $\mathcal{R}$, then add this transition to M along with the state r'. Also, add a to the set M_A.
5. Delete any component of M that is not connected with any of the states in M_Q.

The primary tenet when reasoning about a transformation rule preserving a given property is to remain independent of the source model on which a given rule is applied. The only assumption made on the system model is that the given rule must be applicable on it. Since the rule is applicable on the system model, there must be *at least one* section of said model that matches with $\mathcal{L}$. Utilizing this knowledge, the LTS that is the κ-extended version of $\mathcal{L}$, $\mathcal{L}^\kappa$, is constructed

$$\mathcal{L}^\kappa = (\mathcal{L}.Q, \mathcal{L}_I \cup \{\kappa\}, \mathcal{L}.A \cup \{\sigma_q \mid q \in \mathcal{L}_I\} \cup \{\epsilon_q \mid q \in \mathcal{L}_E\} \cup \{\delta_q, q' \mid q \in \mathcal{L}_E \wedge q' \in \mathcal{L}_I\}, \delta')$$

where for all actions $a \in \mathcal{L}.A$, δ' can be constructed as follows

$$\begin{aligned} q \xrightarrow[\mathcal{L}.\delta]{a} q' &\implies q \xrightarrow[\delta']{a} q' \\ q \in \mathcal{L}_I &\implies \kappa \xrightarrow[\delta']{\sigma_q} q \\ q \in \mathcal{L}_E &\implies q \xrightarrow[\delta']{\epsilon_q} \kappa \\ q \in \mathcal{L}_E \wedge q' \in \mathcal{L}_I &\implies q \xrightarrow[\delta']{\gamma_{q,q'}} q' \end{aligned}$$

The state κ is considered as an abstraction of all the states and transitions of the system LTS that are not a part of a given matched section with $\mathcal{L}$. In a sense, it acts as a black-box state. The σ and ϵ transitions are used to indicate transitions to and from the in-states and exit-states of the pattern LTS, respectively. Finally, γ transitions simulate transitions from exit-states to in-states which may be present in the original system. Since the actions on these transitions (if any) are unknown, σ, ϵ, and γ serve as placeholder labels. The LTS $\mathcal{L}^\kappa$ is, then, a model that demonstrates an LTS on which rule R is applicable.

The pattern $\mathcal{R}$ can also be similarly constructed. The result, $\mathcal{R}^\kappa$, shows how a system LTS on which R is applied appears after the transformation is completed. These LTSs, $\mathcal{L}^\kappa$ and $\mathcal{R}^\kappa$, will form the basis of our strategy.

2.4.1 Example of Model Transformation

In order to demonstrate this process of model transformation, consider this example. Consider the LTS of a system M given in Figure 2.1.

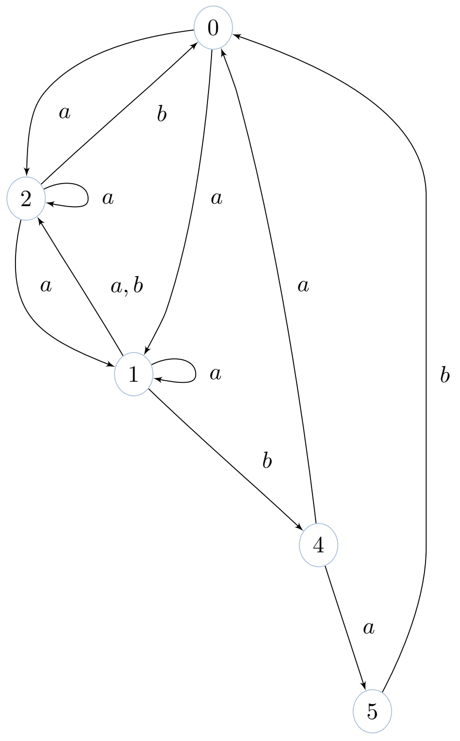

Figure 2.1: A system M to be transformed

Suppose that this system is to be transformed by a rule $R = (\mathcal{L}, \mathcal{R})$, and that the κ-extended versions of these pattern LTSs, namely, $\mathcal{L}^\kappa$ and $\mathcal{R}^\kappa$, are described in Figure 2.2. For these LTSs, state 0 is the only in-glue state, and state 1 is the only out-glue state.

κ contains those states and transitions that are not directly related to the pattern $\mathcal{L}$. This can be observed by comparing $\mathcal{L}^\kappa$ with M, it is easy to observe that the state κ is an abstraction of the states 4 and 5, as well as the a-transition from state 4 to state 5. The transitions $5 \xrightarrow[M_\delta]{b} 0$ and $4 \xrightarrow[M_\delta]{a} 0$ are abstracted by σ_0 in $\mathcal{L}^\kappa$, while the ϵ_1 transition in it abstracts the $1 \xrightarrow[M_\delta]{b} 4$ transition. Finally, the $\gamma_{1,0}$-transition of $\mathcal{L}^\kappa$ is an abstraction of the transition $1 \xrightarrow[M_\delta]{a} 0$. This comparison demonstrated how $\mathcal{L}^\kappa$ represents the source model M.

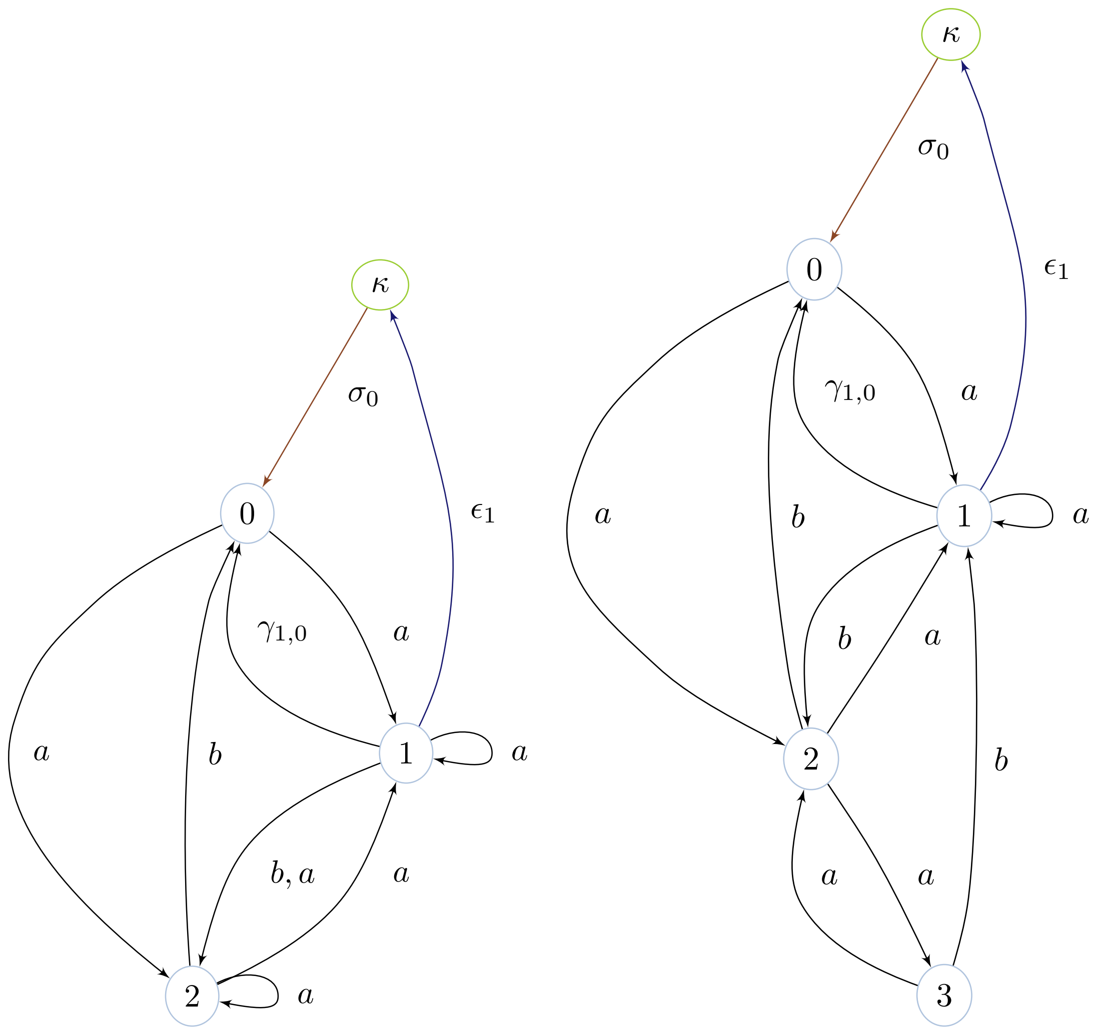

Figure 2.2: $\mathcal{L}^{\kappa}$(Left) and $\mathcal{R}^{\kappa}$(Right) of rule being applied

When the transformation rule R is applied on M, it yields a new model, M_R, which is shown in Figure 2.3.

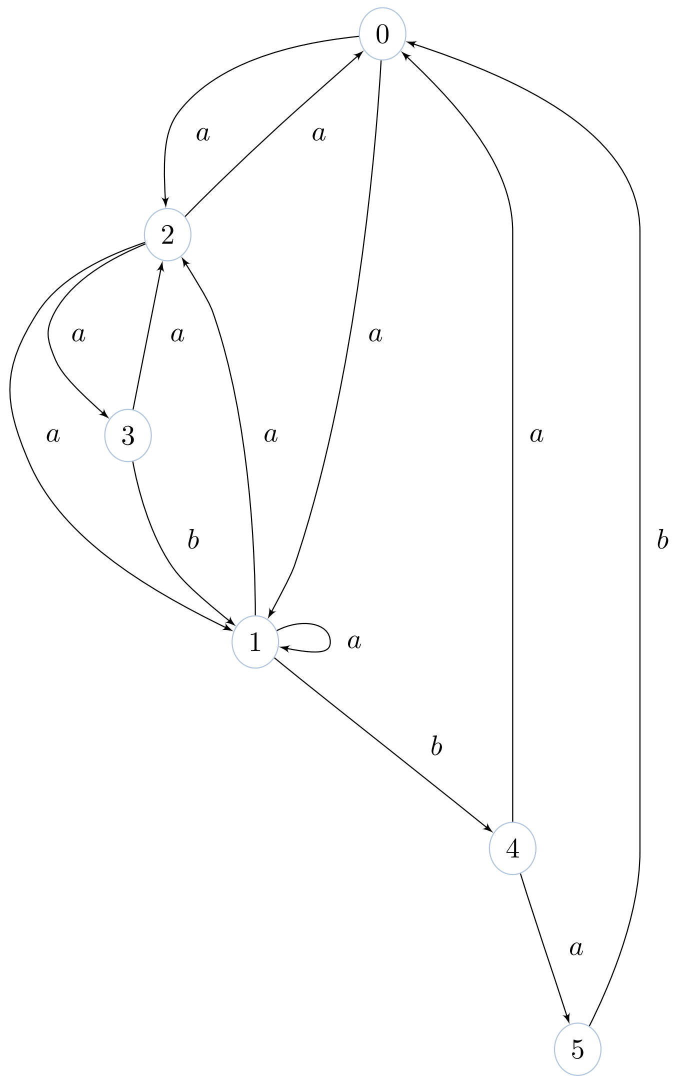

Figure 2.3: Transformed System M_R

It is important to note that the transformation process did not affect any of the states or transitions represented by κ, as well as the labels corresponding to σ_0, ϵ_1, and $\gamma_{1,0}$. The reason for this is that these elements of $\mathcal{L}^\kappa$ and $\mathcal{R}^\kappa$ are external to the pattern LTSs $\mathcal{L}$ and $\mathcal{R}$. Conversely, everything *inside* the pattern LTS $\mathcal{L}$ - the states 0, 1, and 2, and all the transitions between them, are affected by the transformation. A close comparison between M_R and $\mathcal{R}^\kappa$ reveals that this transformation was applied correctly, and resulted in the affected states being replaced by the states 0, 1, 2, and 3 of $\mathcal{R}^\kappa$, as well as all the transitions between them.

2.5 Simulations

A model represented by an LTS M is said to simulate another LTS N (denoted by $M \rightsquigarrow N$) only if there exists a relation $\mathcal{M}$ between the states of M and N such that if $(s,t) \in \mathcal{M}$ for some $s \in N_Q$

and $t \in M_Q$, then

$$\forall_{a \in N_A, s' \in N_Q} s \xrightarrow[N_\delta]{a} s' \implies \exists_{t' \in M_A} t \xrightarrow[M_\delta]{a} t' \wedge (s', t') \in \mathcal{M} \tag{2.1}$$

Further, it must hold that for all starting states t_0 of N, there exists a state $s_0 \in M_{Q_0}$ such that $(s_0, t_0) \in \mathcal{M}$.

In simpler terms, the condition above states that any execution of N can be matched by some execution of M. This direction of this simulation can also be reversed, with M being *simulated by* N only if there exists a relation $\mathcal{M}$ between the states of M and N such that if $(s, t) \in \mathcal{M}$ for some $s \in M_Q$ and $t \in N_Q$, then

$$\forall_{a \in N_A, s' \in M_Q} s \xrightarrow[M_\delta]{a} s' \implies \exists_{t' \in N_A} t \xrightarrow[N_\delta]{a} t' \wedge (s', t') \in \mathcal{M} \tag{2.2}$$

Further, it must hold that for all starting states s_0 of M, there exists a state $t_0 \in M_{Q_0}$ such that $(s_0, t_0) \in \mathcal{M}$.

Given a pair of LTSs M and N, if M simulates N, then there could be multiple matching relations $\mathcal{M}$ that satisfy the simulation criteria given by equation 2.1.

Functionals are essentially functions on top lattices formed by relations. Functionals $F_{\rightsquigarrow}$ and $F_{\leftrightsquigarrow}$ can be defined on the lattice of relations $R \subseteq M_Q \times N_Q$, with the following characteristics

$$F_{\rightsquigarrow}(R) \overset{\text{def}}{=} \{(s, t) \mid s, t \text{ satisfies } 2.1 \text{ with respect to models } M \text{ and } N\}$$

$$F_{\leftrightsquigarrow}(R) \overset{\text{def}}{=} \{(s, t) \mid s, t \text{ satisfies } 2.2 \text{ with respect to models } M \text{ and } N\}$$

2.5.1 Bisimulation

As its name indicates, bisimulation is a relation R between the states of two LTSs M and N that satisfies the conditions in both equation 2.1 and equation 2.2. The set of such relations is denoted by $\sim$. This set is closed under union. Thus, the union of all the members of $\sim$ is the single largest bisimulation relation. This relation is referred to as the bisimilarity relation. If such a relation exists for M and N, then M is said to be strongly bisimilar to N. This is denoted by $M \underset{sb}{\leftrightarrow} N$. The final concept that will be considered here is that of the *bisimulation functional*. A bisimulation functional $F_\sim$ is defined on the lattice of relations $R \subseteq M_Q \times N_Q$, with the following characteristic

$$F_\sim(R) \overset{\text{def}}{=} \{(s, t) \mid s, t \text{ satisfy the bisimulation conditions}\}$$

The prefix *strong* used here seems to imply that there exists an alternative bisimulation relation that is somehow weaker. Indeed, such a relation exists, and is known as weak bisimulation. Weak bisimulation utilizes a type of transition known as τ or internal transitions. These transitions are often used to abstract away details of a system that are not relevant in a given context. Weak bisimulation is concerned with contrasting the similarity of models while effectively ignoring these transitions. Formally, the weak bisimulation relation is the largest relation R such that the following conditions hold on R

1. $$\forall_{s \in M_Q, t \in N_Q} (s, t) \in R \implies s \xrightarrow[M_\delta]{\tau} s' \implies \exists_{t' \in N_Q} t \xrightarrow[N_\delta]{\tau^*} t' \wedge (s', t') \in R$$

2. $$\forall_{s \in M_Q, t \in N_Q} (s, t) \in R \implies s \xrightarrow[M_\delta]{a} s' \implies \exists_{r, q, t' \in N_Q} t \xrightarrow[N_\delta]{\tau^*} r \wedge r \xrightarrow[N_\delta]{a} q \wedge q \xrightarrow[N_\delta]{\tau^*} t' \wedge (s', t') \in R$$

3. $$\forall_{s \in M_Q, t \in N_Q} (s, t) \in R \implies t \xrightarrow[N_\delta]{\tau} t' \implies \exists_{s' \in M_Q} s \xrightarrow[M_\delta]{\tau^*} s' \wedge (s', t') \in R$$

4.
$$\forall_{s\in M_Q, t\in N_Q}(s,t)\in R \implies t \xrightarrow[N_\delta]{a} t' \implies \exists_{r,q,s'\in M_Q} s \xrightarrow[M_\delta]{\tau^*} r \wedge r \xrightarrow[M_\delta]{a} q \wedge q \xrightarrow[M_\delta]{\tau^*} s' \wedge (s',t')\in R$$

Note that in this formalization, the τ^* label on transitions implies zero or more τ-transitions being taken.

Chapter 3

Related Works

In this chapter, as mentioned previously, the goal will be to establish a background for the present work. This is done by considering relevant work in three areas that are the closest to our work. Firstly, the subject of model transformations is discussed, where the choice of modeling technique is taken into consideration as well. This is followed by a section on property verification on models. Finally, the focus shifts even closer towards the current work, with a review of work on property-preserving model transformation.

3.1 On Models and their Transformations

Any discussion on the topic of model transformations must first specify the kind of models that will be used to perform such a transformation on, as well how such a transformation is formalized. Thus, this section is divided into two parts. In the first, some works that deal with formal models are discussed. This is then followed by a subsection that focuses on works that involve the transformation of said models.

3.1.1 Works on Models

In this work, Labeled Transition Systems (LTSs) are used extensively. They are used to represent both the models of systems, as well as for the left and right side of transformation rules. For those interested, a good book on model transformations, as well as MDSE in general, is one by Brambilla et. al. [8]. The discussion that follows here attempts to summarize the present research environment in both modeling in general as well as model transformations in particular.

The decision to use LTSs can be contextualized by considering several works such as ones by Angluin et. al. [22], Lang et. al. [23], Goranko et. al. [24], and Abdulla et. al. [25]. In these works, some common modeling approaches are used, exposing their relative strengths and weaknesses.

Families of DFAs as Acceptors of ω-Regular Languages [22] The goal of this work is to describe ω-regular languages by using families of *saturated* FDFAs (Families of Deterministic Finite Automata). A saturated FDFA $\mathcal{F}$ has the property that if two ω-words uv^ω and $u'v'^\omega$ are equal, then $\mathcal{F}$ accepts uv^ω if and only if it also accepts $u'v'^\omega$. The difference in the finite initial parts of two words such as these - u and u', provides a source of non-determinism in an otherwise deterministic family of automata. This fact is used in this work to show that saturated FDFAs can be used to represent ω-regular languages.

While the set of languages that can be represented by a saturated FDFA is comprehensive, it is also a rather complicated modeling type, which leads to common model-checking operations being less efficient while using it. Further, the behaviour of systems can normally be represented by regular languages, without the need for the ω-extension. Thus, this model representation, though powerful, is not suited for our purposes.

Partial Model Checking Using Networks of Labelled Transition Systems and Boolean Equation Systems [23] As the title of this work suggests, the aim here is to check a system consisting of a network of LTSs and Boolean Equation Systems (BESs). The partial verification approach incorporates a μ-calculus formula to be verified incrementally on each of the processes in the network (represented by either LTSs or BESs). After the semantic elements of a process are incorporated, the formula is made more compact so as to ensure its tractability. Each process transforms the formula at the previous step by a technique known as quotienting.

In this approach, networks of LTSs are used to describe models. Further, μ-calculus properties are being verified. Our work, in essence, assumes a simplified view of the approach presented here, with single LTSs, instead of networks of LTSs, being used. The reason for this is that we aim to present a proof-of-concept tool as the final conclusion of our work, and, therefore, aim to deal with processes that are small enough to be tractable without a need for decomposition into a network of "concurrent" process LTSs.

3.1.2 Works on Model Transformations

The objective of this work is closely associated with model transformations. This is the reason why several works that utilize the same either to obtain some practical benefits, or for some theoretical considerations, are illustrated here. Further, these works are also contrasted and appraised with respect to our goals.

DSL/Model Co-Evolution in Industrial EMF-Based MDSE Ecosystems [26]

In order to understand this work, it is important to first be introduced to two concepts, namely Domain Specific Languages(DSLs) and Object Constraint Language(OCL). DSLs [27] are programming languages that are designed to handle the requirements of a specialized domain of work. They are often developed by smaller development teams, and they evolve much faster than general purpose languages. OCL [28] is a declarative language for specifying rules for a given model (or meta-model) that allow designers to annotate them with constraints that do not have any side-effects.

The goal of this work was the determination of an architecture that enables the automatic co-evolution of DSL (Domain Specific Language) artifacts viz. parsers, editors, models, etc., as they themselves evolve. The main goal of doing so is that this would reduce potential sources of human error by eliminating the tedious work that is usually involved in manually porting over the artifacts as the DSLs' metamodels evolve. The DSLs that are used in a particular industry and which exhibit some co-dependence are considered to be part of a common *ecosystem*.

The authors of this work concluded that it is necessary to know about the characteristics of the specific metamodel and OCL specification that are associated with a DSL, along with a given DSL (meta-)model transformation in order to produce a specification for automatic co-evolution of that particular DSL.

As a system evolves, this evolution as well as the original property are taken into account when designing the property that is targeted by our research objective. This is similar, in a sense, to this work, where the target is to evolve artifacts directly, instead of focusing on properties of the system. However, the major distinction that arises here is that our work does not intend to co-evolve the property directly, but rather, produce a property that characterizes the model transformation as well as the *source* property.

Reliable yet flexible software through formal model transformation (rule definition) [29]

The aim of this work is to show why reliability and flexibility have become increasingly vital elements of software design. This is followed by a new approach that uses model transformations as one of its core tenets.

The main conclusion that they had from their literature review was that there was a need to combine formal (FMM) and semi-formal (SFMM) methods of modeling in order to maximize both reliability and flexibility simultaneously. This is because FMMs provide reliability with their unambiguous semantics, whereas SFMMs provide flexibility with their heuristic nature.

This is where model transformation came into play, in this work. It is used to transform models between Object-Z, which supports FMM, and UML, which supports SFMM. By checking the reliability of a system model with Object-Z and then transforming it to facilitate visualization and flexibility, one can get the best of both worlds.

This work is very generic in the sense that it creates a use for model transformations across the entirety of software development. This illustrates the importance of understanding the semantics of model transformations in a concise manner, which is the focus of our work.

3.2 On Property Verification

The topic of property verification may be approached in several ways. The fundamental distinguishing factor between these approaches is in the expressiveness of the language used to specify the properties and the difficulty in verifying said properties. These two features of a specification language are often in contrast, so that languages that have high expressiveness are also the ones whose verification tasks are more difficult.

One of the most common property specification languages that is used is LTL (Linear Temporal Logic). Works such as ones by Han et. al. [1] and Giacomo et. al. [2] illustrate different facets of work with this logic. In the former work, focus is placed on enhancing the theoretical basis.

3.2.1 Linear temporal logic for hybrid dynamical systems: Characterizations and sufficient conditions [1]

Hybrid dynamical systems are systems in which both continuous-time events as well as the classical "discrete" events are modeled. The aim of this paper was to present semantics for a logic system which, intuitively, follows LTL logic. Then, using the newly-defined semantics, some common properties of hybrid dynamical systems are represented.

This work shows the breadth of models whose properties can be specified and validated by using formal logic systems. This is illustrated by the fact that hybrid dynamic systems capture a strict superset of discrete-time systems represented by approaches such as LTSs, which act as the primary vehicle of representing the semantics of many software systems already in use today.

3.2.2 Reasoning on LTL on finite traces: Insensitivity to infiniteness [2]

LTL_f is a formal logic language that is a superset of LTL. It aims to extend the capabilities of LTL by being capable of handling finite traces. To this end, in addition to the operations ordinarily supported by LTL, two additional operators are present for LTL_f, namely, the *weak until* and the weak release operators. These operators use an additional action *last* which is only valid at the last element of the finite trace being handled. This is the reason why not all LTL_f formulae are insensitive to infiniteness.

This work aims to understand formally the cases where an LTL_f formula can be handled *correctly* by assuming that any trace consists of an initial head section followed by an infinitely repeating tail section. Formally, this can be written as follows:

$$\textsc{Trace} = H(T)*$$

The precise condition that they conclude for this to be true is not relevant in regards to our work (though interesting in its own right). However, this work demonstrates an important fact - different logic families are more(or less) suited for handling different kinds of models on which they can be applied. This is the reason why, for instance, an LTL_f property cannot *always* be used with respect to an infinite trace - it is meant to be used for finite traces.

3.3 On Property-preserving model transformation

The problem of verifying model transformations, especially in the context of incremental model verification, has been treated extensively in a series of research publications by A. Wijs et. al. in [3] [4] [5] [6]. In the following sub-sections, we will discuss each of these papers in some detail.

3.3.1 Incremental Formal Verification for Model Refining [2012] [3]

This paper, authored by Anton Wijs and Luc Engelen, seeks to define a basic formal structure that would enable verifying model transformations. The system proposed in it can verify whether a model transformation preserves a given LTL (Linear Temporal Logic) specification. In particular, the SLCO (Simple Language for Communicating Objects, [30]) language was chosen to describe the models. Concurrent and communicating objects form the basis of this modelling language.

A system modeled by SLCO often has behaviour that is distributed amongst several component sub-systems, each having their own model. This is the reason why the systems discussed in this paper are viewed as networks of sub-models, each corresponding to a different component of the system at hand. Formally, every model $\mathcal{M}$ is described as a composition of a set of LTSs $P_1, \ldots, P_n$ (Labelled transition Systems), each of which corresponds to a process model. In addition, a set of synchronization rules $\sigma_1, \ldots, \sigma_m$ that describe how the composition is done is also a part of the network. The set of states of $\mathcal{M}$ is defined as follows

$$\mathcal{M}_Q = P_{1,Q} \times P_{2,Q} \times \ldots \times P_{n,Q}$$

Each synchronization rule maps a set of actions for a subset of the constituent LTSs to a single action for the final model. These enable the construction of the transition relation $\Gamma \subseteq \mathcal{M}_Q \times \mathcal{M}_A \times \mathcal{M}_Q$, where $\mathcal{M}_A$ is the set of actions of $\mathcal{M}$. Semantically, then, one can think of the system model as the concurrent combination of the process models, with every action in it corresponding to a set of actions in some of the process models, as described by *some* synchronization rule. In the process models for which no action is specified by the rule, the corresponding process can be thought of as being inactive in the present action of the system.

Unlike the models, the model transformations are not described using LTSs. Rather, they are defined by a set of transformation rules of the form $\langle L, R \rangle$ where L, R are LTSs. Each rule is applied by matching the left side L of the rule with a part of the source model, and transforming the same into the LTS specified by the right side rule R. This technique is known as the "double pushout" method. It is critical to note that the rule systems are assumed here to be confluent — their application, in any order, lead to the same final model. This allows us to apply a set of rules on a system model, with the confidence that applying them all would result in the same target model.

In order to perform the actual property preservation verification, the synchronization rules for the model before and after the transformation, σ_L and σ_R are required. σ_L is used to group the rules of the transformation into sets of dependent transformation rules, based on whether they contain actions belonging to the same synchronization rule. Thereafter, a technique called "property-based hiding" is applied on all subsets of these sets. This technique yields a pair of LTSs $\langle L, R \rangle$ for every such subset. This hides the behaviour irrelevant for determining if the model transformation satisfies the given formula. The formula is said to be preserved if and only if L and R are equivalent modulo divergence-sensitive branching bisimulation [31].

The primary novelty of this approach is that it can be applied directly on the model transformation, and the result is independent of the input model onto which such a transformation is applied. The authors claimed that checking property preservation of transformations takes much less time than checking the property on the source and target models individually.

3.3.2 Efficient Property Preservation Checking of Model Refinements [2013] [4]

This work improves the framework discussed in the previous section by improving on its efficiency. It also formally states the complexity of the maximal hiding and divergence-sensitive branching bisimulation equivalence testing that were already discussed in a more informal manner in the previous paper.

The process of transforming a networked model directly, without the main model being constructed, is also introduced formally here. Here, the idea of "glue states" — states common to both the left and right side of a rule, is one that is explored in detail.

This paper refined the concept of rule systems by introducing a set $\hat{V}$ that contains synchronization rules to be introduced as a result of the transformation of the source model. Formally, $\Sigma = \langle R, \hat{V} \rangle$ denotes a rule system, with R being a set of transformation rules. The idea of the rule system being confluent is refined by presenting two concrete rules for ensuring the same. Firstly, the action sets of the left patterns in the rules must be disjoint. Secondly, for each rule, no two matches in a model can intersect.

Figure 3.1 describes the steps that are required to achieve the increased efficiency that is the goal of this paper.

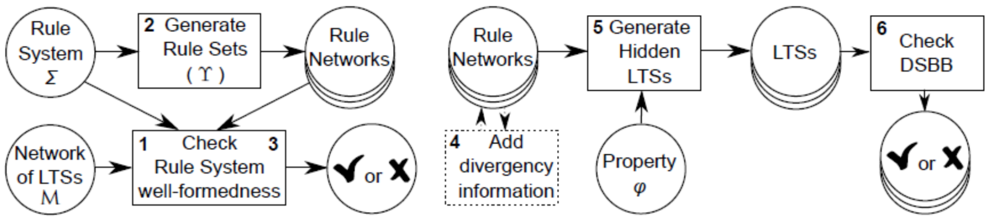

Figure 3.1: Steps for property preservation and well-formedness checks [4]

3.3.3 REFINER: Towards Formal Verification of Model Transformations [2014] [5]

The main aim of this paper was to make a tool that can both create model transformations, as well as verify whether the semantic, safety, or liveness properties of these transformations are maintained. It can also be used to check whether the rule set is confluent. Additionally, the tool provides its users with flexibility regarding whether divergences in the models are to be preserved, when, for instance, checking property preservation.

This tool is the first and only tool thus far to support source model-independent property testing for model transformations. The authors have claimed that one of the major advantages of the tool is its multi-core capability. However, we note that the tool being implemented in PYTHON, an interpreted language, affects its performance adversely.

3.3.4 A formal verification technique for behavioural model-to-model transformations [2018] [6]

This article summarizes the previous findings. It also contributes two new ideas to the general topic of property-preserving model transformations.

Firstly, it improves the flexibility with which transformation rules can be created by distinguishing between "incoming" and "outgoing" glue states. Note that glue states are those states that are common to both the left and right side of a transformation rule. The incoming glue states are those to which a transition exists from outside the matched region of the LTS in the source model, while the outgoing glue states contain transitions towards the source model external to the matched part.

Secondly, it greatly improves on the efficiency of the proposed system. The previous articles required an exponential number of checks for verifying the property-preserving nature of the model transformation, while the present article reduces the number of checks to a linear order. This is formally proven using the Coq theorem prover [1]. In simple terms, the context in which the rules are applied is also considered, thereby allowing a more fine-grained verification that works directly on the individual rules, instead of having to be applied on subsets of the rule set.

In general, a large portion of this article is devoted to formally defining and analyzing many of the ideas introduced in the previous papers. A framework for applying rule systems is also discussed in some detail.

With these papers, we gain considerable insight on property-preserving model transformations. Unfortunately, there is not much more literature from other groups on the topic of property-preserving model transformations. At this point, therefore, we turn to our second topic of discussion, incompletely-specified models. Menghi et. al. first published a paper on this topic in 2016 [16]. We will focus on a more comprehensive pre-print version of this paper [7].

3.3.5 Modeling, refining and analyzing Incomplete Büchi Automata [7]

The goal of this paper is to model the semantics of partially-specified models. Such models often arise when a system is continuously being refined, and new features are being added. The core modeling construct used here is Büchi automata.

A Büchi automaton is very similar to an LTS in that it has a graph-like structure with states and transitions with actions (labels). There are, however, two major differences between the two modeling systems. Firstly, the concept of τ transitions is not present with Büchi automata. Secondly, and most importantly, a word is accepted by a Büchi automaton if and only if during the run for the word, it visits one of the “final states” of the Büchi automaton infinitely often. The words in the context of a Büchi automaton must be infinite.

Incomplete models are described by introducing a modeling framework known as Incomplete Büchi Automaton (IBA). The paper assumes a view of incomplete models that is recursive in nature. This implies that a model may contain one or more black-box states which may be replaced recursively by either IBAs or regular Büchi Automata during refinement. This view is congruent with the top-down sequential refinement design process for systems under development.

The extension to IBA from regular Büchi automata (BA) is done by partitioning the state space Q into disjoint sets of states. These are, respectively, a set of black-box states (B), and a set of regular states (R).

For IBAs, a run is defined informally as an infinite sequence of states over an infinite sequence of characters (ω-words). The characters can belong to a larger set of atomic propositions than those defined in the IBA itself. This supports the idea that it may be possible to make additional claims about the states of an evolving software system. A word is said to be definitely accepted if there exists a run for the word where at least one of the final states occurs infinitely often in it and no black-box states are encountered in the run. It is said to be possibly accepted if it is not definitely accepted, and, there exists a run where a final state occurs infinitely often, and there are black-box states in the run. Finally, if no final states occur infinitely often in any run for a word, it is not accepted.

The main goal of incomplete model checking is to verify an incomplete model M against supplied properties given by a formula $\mathcal{F}$. Since parts of the model have not been specified, a three-value logic system is required, consisting of TRUE, FALSE, and UNDEFINED.

As parts of the system are unspecified, three responses to such a verification task is possible : yes, no, or unknown. The unknown response happens when there exists a black-box state in M whose replacement must satisfy an eventuality criterion for $\mathcal{F}$ to hold on it. In this case, the model checker must also provide annotations for the criterion that must hold on each of the black box states of M. It is possible that some black-box states are not required to satisfy any restriction with respect to their replacements, in order for M to satisfy $\mathcal{F}$.

[1] https://coq.inria.fr/

The replacement of a black box state "b" in an IBA M is done in such a manner that in the resulting IBA N, it holds that if a word is definitely accepted (resp. not accepted) in M, then the same also holds in N.

The process of checking whether a replacement satisfies $\mathcal{F}$ is known as "Replacement checking". Replacement checking is used to produce constraints. Consider an IBA $\mathcal{R}'$, which is to serve as a refinement of some model M, against a constraint $C = \langle S, S_p \rangle$. S describes behaviour which, if present in $\mathcal{R}'$, guarantees that $\mathcal{F}$ would not hold on $\mathcal{R}'$. On the other hand, if the behaviour described in S_p occurs in $\mathcal{R}'$, then a violation of $\mathcal{F}$ by $\mathcal{R}'$ is possible.

The satisfaction(or lack thereof) of the constraint C by the replacement by $\mathcal{R}'$ is computed as follows. First, A Büchi automaton $\mathcal{R}_u$ representing the behaviour of $\mathcal{R}'$ that is guaranteed to be present at run-time is computed. This is known as the *under-approximation* of $\mathcal{R}'$. If there exists some behaviour that is present in both $\mathcal{R}_u$ and S (that is, the intersection of their languages is non-empty), then C is not satisfied by $\mathcal{R}'$. If there exists no such a behaviour, then an *over-approximation* $\mathcal{R}_o$ of $\mathcal{R}'$ is computed, which contains all behaviour of $\mathcal{R}'$ that can *possibly* be present at run-time. If there exists some behaviour common to S_p and R_o, then $\mathcal{R}'$ is said to possibly satisfy C. If this is not the case, then C is said be to definitely satisfied by $\mathcal{R}'$.

Chapter 4

Procedure

In this chapter, a detailed description of the approach used to reach the research objective is discussed. As mentioned previously, the LTSs $\mathcal{L}^{\kappa}$ and $\mathcal{R}^{\kappa}$ will be used in order to achieve this. Broadly speaking, the approach used can be divided into three steps

1. **Merging LTL Property $\boldsymbol{\omega}$ and Left Pattern $\boldsymbol{\mathcal{L}^{\kappa}}$.** This step aims to utilize the knowledge that the system represented by $\mathcal{L}^{\kappa}$ satisfies ω to refine it and produce an LTS $\mathcal{L}^{\kappa}_{\omega}$ such that $L(\mathcal{L}^{\kappa}_{\omega}) \supseteq L(M_{sys})$ where M_{sys} is the LTS modeling the system represented by $\mathcal{L}^{\kappa}$.
2. **Transforming the Refined Output.** In this step, the LTS $\mathcal{L}^{\kappa}_{\omega}$ obtained in the previous step is transformed into $\mathcal{R}^{\kappa}_{\omega}$ by applying the rule $R = (\mathcal{L}, \mathcal{R})$ on it, following the approach described previously, in the preliminary chapter.
3. **Extracting a Characteristic Property.** This step uses the LTS $\mathcal{R}^{\kappa}_{\omega}$ obtained in the previous step, and transforms it into properties ψ_n and ψ_s which characterize necessary and sufficient conditions for a model to either be simulated by it, or to simulate it, respectively.

Note that in the remainder of this report, the source model which is represented by $\mathcal{L}^{\kappa}$ and the model which is transformed by R and represented by $\mathcal{R}^{\kappa}$, are represented by the LTSs M_{sys} and M_{trans} respectively.

4.1 First Step : Merging LTL Property ω and Left Pattern $\mathcal{L}^{\kappa}$

The first step of this work is to *refine* $\mathcal{L}^{\kappa}$ by utilizing the knowledge that the system LTS satisfies ω, and, therefore, $\mathcal{L}^{\kappa}$ (which represents the system LTS) must satisfy it as well. This process is termed as *merging* in subsequent discussions.

The first step of the merging process is to transform ω into a Büchi Automaton G_{ω} representing ω. This is done by utilizing the SPOT framework [32]. SPOT produces a Büchi automaton that has propositional logic formulae over the set of atomic propositions, AP, as its transition labels.

The state κ of $\mathcal{L}^{\kappa}$ represents those states which are not matched by the pattern LTS. Now that G_{ω} has been constructed, one of its states, represented by s^{κ}, is selected to be matched against κ.

In Algorithm 1, the MAIN procedure, tasked with performing the merging process, is provided with G_{ω} as an input. Following this, a set of possible refinements to $\mathcal{L}^{\kappa}$ can be constructed. This process utilizes an auxiliary data structure, the *MatchList*, which is defined as follows

$$\begin{aligned} MatchList =& \{m \mid m \text{ is a MatchInfo}\} \text{ where} \\ & \text{MatchInfo} = (\text{Matching}(\mathcal{M}), \text{ArtificialUpdateSet}(\mathcal{U}), \text{FinalEdge}(\mathcal{FE})) \end{aligned}$$

where

1. Matching - A matching is a map defined as $\mathcal{M}\colon \mathcal{L}^\kappa_Q \to \mathcal{P}(G_{\omega,Q})$. Further, $G_{\omega,Q}$ is the set of states of G_ω, With this map, every glue state and κ in $\mathcal{L}^\kappa$ is associated with at least one state of G_ω. The state association is done by considering every transition $t \equiv s \xrightarrow[\mathcal{L}^\kappa_\delta]{a'} s'$ in $\mathcal{L}^\kappa$ for which $\{s, s'\} \subseteq \{\kappa\} \cup \mathcal{L}_I \cup \mathcal{L}_E$. For each such t, the transitions of G_ω $t' \equiv r \xrightarrow[G_{\omega,\delta}]{a'} r'$ are considered where $a \subseteq a'$. These are the candidate transitions that can simulate the transition t. Given at least one of the following holds, t' is considered to be a possible suitable candidate:

   (a) $\mathcal{M}(s)$ is either empty or contains r, and $\mathcal{M}(s')$ is either empty or contains r'.

   (b) Both $\mathcal{M}(s)$ and $\mathcal{M}(s')$ are non-empty, and r is in $\mathcal{M}(s)$ or r' in $\mathcal{M}(s')$.

   Every glue-state of $\mathcal{L}^\kappa$, as well as the κ needs to be matched to some state of G_ω for the remainder of the algorithm to function. Thus, in `line 3` of Algorithm 1, matches in which this does not hold are filtered out. Each candidate transition t' leads to a new matching, created by adding the r and r' respectively to $\mathcal{M}(s)$ and $\mathcal{M}(s')$ respectively.

2. ArtificialUpdateSet - In order to understand the construction of this set, one must first understand the term “artificial label”. An artificial label refers to a dummy action that was introduced during the construction of the κ-extended version of a pattern LTS. Thus, all the σ, ϵ and γ-labels are artificial labels.

   The set ArtificialUpdateSet indicates the concrete updates to artificial labels that were made while the merging algorithm proceeded to find the corresponding matching. It is a set defined as follows:

$$\begin{aligned}\mathcal{U} = \{(t, a) \mid a \in \mathcal{P}(G_{\omega,A)}, t \equiv s \xrightarrow[\mathcal{L}^\kappa_\delta]{a'} s' \text{ where } s, s' \in \mathcal{L}^\kappa_Q, a' \in \mathcal{P}(\mathcal{L}^\kappa_A) \\ \wedge a' \text{ is artificial} \wedge \exists_{r,r' \in G_{\omega,Q}} r \xrightarrow[G_{\omega,\delta}]{a} r' \wedge WAS\_MATCHED(t, r \xrightarrow[G_{\omega,\delta}]{a} r')\}\end{aligned} \tag{4.1}$$

   where the predicate $WAS\_MATCHED(t, r \xrightarrow[G_{\omega,\delta}]{a} r')$ is true if and only if the transition $r \xrightarrow[G_{\omega,\delta}]{a} r'$ was matched against t at some point during the progression of this step.

   In order to understand this formalism, we look towards the process of constructing $\mathcal{M}$. As $\mathcal{M}$ is being constructed, if a transition of $\mathcal{L}^\kappa$ with an artificial label, t is encountered, it can be mapped against any edge t' of G_ω. However, in doing so, the claim is made that t can be simulated by t', and so it must hold that $t_{action} \subseteq t'_{action}$. This information is recorded by associating t with t'_{action} in $\mathcal{U}$.

3. FinalEdge - This is a map defined as

$$\mathcal{FE}\colon \mathcal{L}^\kappa_Q \to \mathcal{P}(\{t \mid t \equiv s \xrightarrow[\mathcal{L}^\kappa_\delta]{a} s', \text{ where } s, s' \in \mathcal{L}^\kappa_Q, a \subseteq \mathcal{L}^\kappa_A\})$$

   As $\mathcal{M}$ is being constructed, if a transition $t \equiv s \xrightarrow[\mathcal{L}^\kappa_\delta]{a'} s'$ in $\mathcal{L}^\kappa$ is matched to a transition $t' \equiv r \xrightarrow[G_{\omega,\delta}]{a'} r'$ of G_ω, then

   (a) If $r \in G_{\omega,F}$, then the set $\mathcal{FE}(s)$ is updated by adding t to it.

   (b) If $r' \in G_{\omega,F}$, then the set $\mathcal{FE}(s')$ is updated by adding t to it.

   This process results in states being associated with one or possibly more than one transitions that are related to transitions to or from final states of G_ω that were simulated by those transitions. This structure is used at a later stage of the analysis to filter out refined solutions that cannot occur due to the requirements for valid traces in G_ω introduced by its final states.

In algorithm 1, an empty *MatchList* is created and passed as a parameter to the GET_MATCHES method on `line 2`. This empty instance is presented as a list consisting of a single element, which contains a *Map*() instance, an empty set, and finally, a second empty *Map*() instance. These three components correspond to the three components of every MatchInfo instance that were discussed above.

The process of computing the set of matches can be analyzed by delving into the GET_MATCHES procedure, whose definition begins on `line 76` of the algorithm. The process that is followed extends the match set by iterating over the set of transitions of $\mathcal{L}^\kappa$(denoted by parameter *Pat* in the procedure). Lines `78-80` denote each iteration step. Firstly, it is ensured that the transition being considered has a state associated with it that is either a glue state, or is κ (`line 78`). This is done since refinements for a transition containing only states that are internal to $\mathcal{L}^\kappa$ cannot be made, as they are already known definitively from the structure of $\mathcal{L}$.

For every transition that passes the test, it, alongside the graph G (equivalent to G_ω from the MAIN context), and the current match set ML, are passed as arguments to the EXTEND_MATCHES procedure, on `line 79`. This procedure augments the present match set by adding the matching information corresponding to the present transition to every matching instance in ML where such an addition does not violate the consistency of the matching.

In the EXTEND_MATCHES procedure, algorithm proceeds by first creating a new empty list ML_{new} (`line 85`) which acts as a container for the results that are obtained from the procedure. Note that the transition passed onto this procedure has a label a_0, as visible in the parameters of `extend_matches` on `line 80`. A set of candidate transitions of G_ω (represented as G in the present context) that can map against such a transition with label a_0 is first constructed (`line 86`). Only transitions that have labels a_1 such that a_0 satisfies the condition a_1, or where a_0 is an artificial label (one of the labels in transitions between κ and the glue states, or amongst the glue states), fit this criteria.

If this set, named *candidates*, turns out to be empty in any call of EXTEND_MATCHES, then in `line 93`, an empty list would be returned and set as the new match list instance on `line 79`. Proceeding forward with the execution of the algorithm at this stage, lines `89, 90` would never be reachable, as ML, passed into EXTEND_MATCHES, would keep being empty. Therefore, in this scenario, the match set returned by GET_MATCHES would be empty. This makes sense, since the *candidates* set being empty would imply that the matching process has, essentially, failed for a particular transition (and so also as a whole).

If *candidates* is not empty (`line 87`), then an iteration is performed over every MatchInstance that is presently in the *MatchList*(ML). This iteration step occurs on `lines 85 and 90`. On `line 89`, the EXTEND_MATCH procedure is called with the set *candidates* passed alongside the matching instance, the transition of $\mathcal{L}^\kappa$ being considered, and G (G_ω). This procedure produces a list of matching instances produced by adding the matches in *candidates* against the transition $s_0 \xrightarrow[M_\delta]{a_0} s_1$ being considered. This list is subsequently added to ML_{new} on `line 90`.

The mechanism for the working of the EXTEND_MATCH procedure can now be detailed. This procedure begins by initializing a list *results* (`line 96`), which, as its name implies, will contain the results of this procedure's execution. Next, an iteration is performed over the transitions in *candidates*. For each such transition, the iteration step proceeds from `line 100` through `line 115`.

Each step begins by first checking if the given transition of candidates, $r_0 \xrightarrow[M_\delta]{a_1} r_1$ can be matched under the present condition of the matching instance MI.

In subsequent parts of this section, M^q will be used to refer to the set of state that are matched against q. The matching can only be extended if one of the following conditions hold

1. Both s_0 or s_1 have not been matched ($M^{s_0} = \emptyset \wedge M^{s_1} = \emptyset$).

2. s_0 has not been matched, and s_1 has been previously matched with r_1 ($M^{s_0} = \emptyset \wedge r_1 \in M^{s_1}$).

3. s_1 has not been matched, and s_0 has been previously matched with r_0 ($r_0 \in M^{s_0} \wedge M^{s_1} = \emptyset$).

4. If s_0 or s_1 have been matched, then either r_0 has been matched with s_0, or r_1 with s_1 ($r_0 \in M^{s_0} \vee r_1 \in M^{s_1}$). This case handles the scenario where two transitions of $\mathcal{L}^{\kappa}$ either begin or end at the same state, and is matched against transitions of G_ω with the same characteristic.

If this check is successful, then a copy of the match instance being considered, called $MICopy$, is created (`line 101`). Then, r_0 and r_1 are added to the matching sets for s_0 and s_1 in $MICopy$ (`line 102`). Next, the FinalEdge map is updated. This is done by using the variable $finalSet$, defined on `line 103` as a copy of the FinalEdge map in the current matching instance. If r_0 and/or r_1 is a final state then the edge $s_0 \xrightarrow[M_\delta]{a_0} s_1$ is created and added to $finalSet_{s_0}$ and/or $finalSet_{s_1}$ respectively (`lines 104-109`). Finally, if a_0 is an artificial label, the ArtificialUpdateSet is updated by adding the tuple of the present transition of $\mathcal{L}^{\kappa}$ being considered, and the new label a_1 for it. This is done in `lines 110-112`. Finally, in `line 113`, the *results* list is updated with this extended matching instance. In the end, it is returned on `line 116`.

Once a $MatchList(ML)$ is obtained from the `get_matches` procedure, this list is first filtered. Based on the choice of s^{κ}, only those solutions where $\mathcal{M}(\kappa) = \{s^{\kappa}\}$ are chosen. This occurs on `line 3` of the algorithm, resulting in the filtered matching list, $MFilt$. the task of obtaining LTSs that are refinements of $\mathcal{L}^{\kappa}$ begins.

Since the goal of this step is to obtain a model $\mathcal{L}^{\kappa}_{\omega}$ whose language is a superset of the language of the system, this refinement process, for a given $\mathcal{M} \in ML$, modifies transitions external to $\mathcal{L}$ following two principles

I. A transition is added if it is *possible* to match it against some transition in G_ω, following the matchings considered in $\mathcal{M}$.

II. A transition is removed if there are no *possible* matches of this transition against any transition in G_ω, following the matchings considered in $\mathcal{M}$.

Below, the steps of this process are enlisted.

1. For every match info MI in ML, and for every state s that is either an in-state or κ, a check is made to see whether the initial state of G_ω is in $MI_{0,s}$ or not. If it is, then, s is marked as an initial state in the solution corresponding to MI. This occurs on `lines 7-11` in the algorithm.

2. For every possible solution, edges are concretized based on information in $\mathcal{U}$, in the CONCRETIZE method called on `line 12`. Here, every action that is satisfied by any of the formulae on the labels in MI_1 of the present matching instance MI, is added (`line 46`). If a transition with an artificial label cannot be concretized, it is removed, following rule II above.

3. Further, self-loops are added on exit-states and κ where these states are matched to states of G_ω that have self-loops on them (`lines 16-23`).

4. Following this, labels of transitions from exit-states to in-states of $\mathcal{L}^{\kappa}$ are updated by adding all actions that are satisfied by labels of those transitions which are present between the states of G_ω that they are matched to (`lines 24-36`).

5. Finally, new states viz. r' are added if there are transitions to s^{κ} from r in G_ω. Transitions between these states and κ are made in a manner analogous to the connections that s^{κ} has. Transitions to or from these states to glue states are also made wherever the glue states are matched to states that are adjacent to s' (`lines 37-40`).

6. In the resulting solution LTSs, for every cycle (`line 55`), there must exist a state s such that $\mathcal{M}(s) \cap G_{\omega,F} \neq \emptyset$ (`line 59`) and there is a transition $t \in \mathcal{FE}(s)$ such that t is in the cycle (`line 60`). This ensures that, in any simulation of an infinite trace, atleast one of the final states of G_ω are visited infinitely often. If this is not the case for any solution concretized by the previous step, that solution is subsequently discarded (`line 42`).

Algorithm 1 Algorithm for combining information in G and L_κ

```
1: procedure MAIN($G_\omega$, $\mathcal{L}^\kappa$, $s^\kappa$)
2:     $MatchList \leftarrow$ GET_MATCHES($[[Map(), \emptyset, Map()]], \mathcal{L}^\kappa, G_\omega$)
3:     $MFilt \leftarrow \{(\mathcal{M},\mathcal{U},\mathcal{FE}) \in MatchList \mid \mathcal{M}(\kappa) = \{s^\kappa\} \wedge \forall_{q \in \mathcal{L}^\kappa_Q} \mathcal{M}(q) \neq \phi\}$
4:     $solutions \leftarrow []$
5:     for $MI \in MFilt$ do
6:         $CandLTS \leftarrow \mathcal{L}^\kappa$
7:         for $state \in G_{\omega,I} \cup \{\kappa\}$ do
8:             if $G_{\omega,Q_0} \cap MI_{0,state} \neq \emptyset$ then
9:                 Add $state$ to $CandLTS_{Q_0}$
10:            end if
11:        end for
12:        $CandLTS \leftarrow$ CONCRETIZE($CandLTS, MI$)        ▷ Add edges related to matching
13:        for $s \in \mathcal{L}^\kappa_I \cup \mathcal{L}^\kappa_E$ do
14:            $oEdges \leftarrow \{s_0 \xrightarrow[G_\delta]{a} s_1 \mid s_0 \in MI_{0,s}\}$
15:            $selfLoops \leftarrow \{s_0 \xrightarrow[G_\delta]{a} s_1 \mid s_0 \xrightarrow[G_\delta]{a} s_1 \in oEdges \wedge s_0 = s_1\}$
16:            Let $uLabel$ be the the set of actions in $\mathcal{L}^{\leq}_{\mathcal{A}}$ satisfied by any of the transitions in
    $selfLoops$
17:            if $uLabel \neq \emptyset$ then
18:                if There exists a self-loop at node $s$ with label $a$ then
19:                    Add transitions with labels $l$ in $uLabel$, if $l \neq a$
20:                else
21:                    Add transitions with labels $l$ in $uLabel$
22:                end if
23:            end if
24:            if $s \in \mathcal{L}^\kappa_E$ then
25:                for $s' \in \mathcal{L}^\kappa_I$ do
26:                    $iEdges \leftarrow \{s_0 \xrightarrow[G_\delta]{a} s_1 \mid s_1 \in MI_{0,s'}\}$
27:                    $common \leftarrow iEdges \cup oEdges$
28:                    Let $uLabel$ be the the set of actions in $\mathcal{L}^{\leq}_{\mathcal{A}}$ satisfied by any of the transitions
    in $common$
29:                    Let $eLabel$ be the existing label on the transition from $s$ to $s'$
30:                    if $eLabel$ is artificial then
31:                        Delete the transition with label $eLabel$ and add transitions with labels in
    $uLabel$
32:                    else
33:                        Add transitions with labels in $uLabel$
34:                    end if
35:                end for
36:            end if
37:            for Every transition of the form $s_1 \xrightarrow[G_\delta]{a_0} s^\kappa$ do
38:                Add state $s_1'$ to $CandLTS$
39:                Add transitions between $\kappa$ and $s_1'$ corresponding to transitions between $s^\kappa$ and
    $s_1$
40:            end for
41:        end for
42:        if IS_VALID($MI, CandLTS, G_\omega$) then
43:            $solutions = solutions + CandLTS$
44:        end if
45:    end for
```

```
46:     return solutions
47: end procedure
48: procedure CONCRETIZE($Pat$, $MI$)
49:     for $s_0 \xrightarrow[Pat_\delta]{a} s_1$ where $a$ is artificial do
50:         Delete $s_0 \xrightarrow[Pat_\delta]{a} s_1$ and add transition $s_0 \xrightarrow[Pat_\delta]{a_{new}} s_1$ for every $a_{new}$ satisfied by any of the
    labels in the transitions mapped in $MI_1$
51:     end for
52: end procedure
53: procedure IS_VALID($MI$, $Pat$, $G$)
54:     Let $Cycles$ be a list of all elementary circuits of $Pat$
55:     for $cycle \in Cycles$ do
56:         Let $cycle = s_0 \xrightarrow[Pat_\delta]{a_0} s_1 \ldots \xrightarrow[Pat_\delta]{a_{n-1}} s_n \xrightarrow[Pat_\delta]{a_n} s_0$
57:         $validated \leftarrow$ FALSE
58:         for $s_i \xrightarrow[Pat_\delta]{a} s_{i+1} \in cycle$ do
59:             if $G_F \cap MI_{0,s_i} \neq \emptyset$ then
60:                 if $s_i \xrightarrow[Pat_\delta]{a_i} s_{i+1} \in MI_{2,s_0}$ then
61:                     $validated \leftarrow$ TRUE
62:                     break
63:                 end if
64:                 if $(i > 0 \wedge s_{i-1} \xrightarrow[Pat_\delta]{a_{i-1}} s_i \in MI_{2,s_0}) \vee s_n \xrightarrow[Pat_\delta]{a_n} s_0$ then
65:                     $validated \leftarrow$ TRUE
66:                     break
67:                 end if
68:             end if
69:         end for
70:         if $validated =$ FALSE then
71:             return FALSE
72:         end if
73:     end for
74:     return TRUE
75: end procedure
76: procedure GET_MATCHES($ML$, $Pat$, $G$)
77:     for $s \xrightarrow[Pat_\delta]{a} s'$ where $s, s' \in Pat_Q$ do
78:         if Either $s$ or $s'$ is $\kappa$ or a glue state then
79:             $ML \leftarrow$ EXTEND_MATCHES($G, s \xrightarrow[Pat_\delta]{a} s', ML$)
80:         end if
81:     end for
82:     return $ML$
83: end procedure
84: procedure EXTEND_MATCHES($G$, $s_0 \xrightarrow{a_0} s_1$, $ML$)
85:     $ML_{new} \leftarrow []$
86:     $candidates \leftarrow \{r_0 \xrightarrow[G.\delta]{a_1} r_1 \mid a_1 \subseteq a_0 \vee a_0$ is artificial$\}$
87:     if $candidates \neq \emptyset$ then
88:         for $MI \in ML$ do
89:             $extendedResults \leftarrow$ EXTEND_MATCH($G, s_0 \xrightarrow{a_0} s_1, MI, candidates$)
90:             Add match instances in $extendedResults$ to the end of $ML_{new}$
91:         end for
92:     end if
93:     return $ML_{new}$
```

```
94: end procedure
95: procedure EXTEND_MATCH($G$, $s_0 \xrightarrow{a_0} s_1$, $MI$, $candidates$)
96:     $results \leftarrow []$
97:     $M^{s_0} \leftarrow MI_{0,s_0}$        ▷ $M^{s_0}$ contains the states that $s_0$ has been matched to in $MI$
98:     $M^{s_1} \leftarrow MI_{0,s_1}$        ▷ $M^{s_1}$ contains the states that $s_1$ has been matched to in $MI$
99:     for $r_0 \xrightarrow{a_1} r_1 \in candidates$ do
100:        if IS_MATCHABLE($M^{s_0}$, $M^{s_1}$, $r_0$, $r_1$) is TRUE then
101:            $MICopy \leftarrow MI$
102:            Create $MICopy_{0,s_0}$ and/or $MICopy_{0,s_1}$ if necessary, and then add $r_0$ and $r_1$ to
    them respectively
103:            $finalSet \leftarrow MICopy_2$
104:            if $r_0 \in G_F$ then
105:                Create $finalSet_{s_0}$ if necessary, and add the edge $s_0 \xrightarrow{a_0} s_1$ to it
106:            end if
107:            if $r_1 \in G_F$ then
108:                Create $finalSet_{s_1}$ if necessary, and add the edge $s_0 \xrightarrow{a_0} s_1$ to it
109:            end if
110:            if $a_0$ is an artificial label then
111:                $MICopy_1 \leftarrow MICopy_1 \cup \{(s_0 \xrightarrow{a_0} s_1, a_1)\}$
112:            end if
113:            $results \leftarrow results + MICopy$
114:        end if
115:    end for
116:    return $results$
117: end procedure
```

4.2 Second Step : Transforming the Refined Output

The previous step can produce multiple solutions. This is because there are potentially multiple ways to match the sets of states of $\mathcal{L}^\kappa$ and G_ω, leading to multiple refined versions of $\mathcal{L}^\kappa$.

The aim of this step in our process is to transform the resulting model by *applying* the rule transformation $R = (\mathcal{L}, \mathcal{R})$ on one of the solutions of the merging step.

The choice of a candidate solution to provide as input to this step is made by comparing the LTSs obtained in the previous one. This is done by using the mCRL2 [33] toolkit's LTSCOMPARE tool. This tool only accepts LTSs that have a single initial state. Thus, for each solution that has multiple initial states in Q_0, a new state is created, and τ-edges from it to every state in Q_0 are added. Finally, Q_0 is replaced by this new dummy state. The resulting LTSs can be compared under weak bisimilarity. This process partitions the set of solutions into equivalence classes under the weak bisimilarity relation. Finally, an arbitrary solution from the largest of these classes is used as input.

Given an input model to this step, M, the actual transformation logic is rather simple. Instead of deleting and replacing transitions of $G_{in} \equiv \mathcal{L}^\kappa_\omega$, an LTS G with a single node, κ is chosen as the starting point (`line 2`). Next, the in- and exit-states of $\mathcal{L}^\kappa_\omega$ are added to the set of states of G (`line 3`). Finally, the set of initial states, in-states, and exit-states of G are set to be the same as that of $\mathcal{L}^\kappa_\omega$ (`line 4`). Essentially, these steps set up the components in the final transformed LTS that will remain unchanged from $\mathcal{L}^\kappa_\omega$.

This is followed by adding transitions whose source and target states are both either glue states of $\mathcal{L}^\kappa_\omega$ or the κ state (`lines 6-9`). Again, this external part of the pattern is unchanged by the transformation. For the same reason, the transformed output must also contain the states neighboring κ in $\mathcal{L}^\kappa_\omega$ which were created in the merging step. These states, along with their transitions, are added on `line 11`. Finally, the states and transitions "inside" the pattern are replaced by those in $\mathcal{R}^\kappa$ (`lines 12, 13`), reflecting the main difference between G and $\mathcal{L}^\kappa_\omega$. In subsequent discussions, the result G of this step will be referred to as $\mathcal{R}^\kappa_\omega$.

Algorithm 2 Algorithm for transforming input model in M to right side of rule R

```
1: procedure TRANSFORM($G^{in}$, $\mathcal{R}^{\kappa}$)
2:     Let $R$ be an LTS with a single node, $\kappa$
3:     $R_Q \leftarrow R_Q \cup G_I^{in} \cup G_E^{in}$
4:     $R_I, R_E, R_{Q_0} \leftarrow G_I^{in}, G_E^{in}, G_{Q_0}^{in}$
5:     $specialStates \leftarrow R_Q$
6:     for $s \in specialStates$ do
7:         $artificialTrans \leftarrow \{s_0 \xrightarrow[G_\delta]{a} s_1 \mid s_0 = s \wedge s_1 \in specialStates\}$
8:         Add all transitions and nodes in $artificialTrans$ to $G$
9:     end for
10:    Add all nodes connected to $\kappa$ in $G^{in}$ which are not glue states, along with their transitions, to $G$
11:    $newTrans \leftarrow \{s_0 \xrightarrow[G_\delta]{a} s_1 \mid \{s_0, s_1\} \cap specialStates = \emptyset\}$
12:    Add all transitions and nodes in $newTrans$ to $G$
13:    return $G$
14: end procedure
```

4.3 Final Step : Extracting a Characteristic Property

In the final stage, the goal is to obtain a property relevant to the research question at hand. Note that the model that we get from the previous step

1. Is obtained by applying the transformation rule R on a model $\mathcal{L}_\omega^\kappa$ which has the relevant characteristics of a system on which the rule can be applied, and

2. Takes into account the fact that the property being satisfied by $\mathcal{L}^\kappa$ is ω. This is because this information was embedded into $\mathcal{L}_\omega^\kappa$, which was, subsequently, transformed.

3. The language of $\mathcal{L}_\omega^\kappa$, $L(\mathcal{L}_\omega^\kappa)$, was a superset of the language of the source model, $L(M_{sys})$. This is because the refinement process in the first step of this algorithm ensured that any transition that has the *possibility* of being present in the source model M_{sys} is also present in the target model $\mathcal{L}_\omega^\kappa$.

Given these two reasons, we claim that $\mathcal{R}_\omega^\kappa$ can act as a suitable source from which we can extract a property that takes both the model transformation and the original property ω into consideration.

At the end of this section, we will end up with two properties ψ_n and ψ_s, corresponding to a necessary and a sufficient condition respectively, so that

1. **Necessary Condition** : Any model that can be simulated by M_{trans} must necessarily satisfy ψ_n.

2. **Sufficient Condition** : ψ_s being satisfied by a model is *sufficient* to imply that it can simulate M_{trans}.

Note that the line numbers described in the remainder of this section must be viewed with respect to Algorithm 3.

The technique used for extracting the desired characteristic formula is obtained from a work by Müller et. al. [34]. Broadly speaking, it proceeds by first creating a system of characteristic equations, and then, using Gaussian elimination, a single characteristic formula may be obtained. Given a system of equations of size n, the process of creating a single equation leads to an exponential increase in the size of the final formula that results from this process, when compared to the sizes of the original formulae. Thus, if the sizes of the formulae in the original system has an upper bound M, then the size of the formula resulting from Gaussian elimination is of the order of $O(M^n)$.

The process of creating the characteristic system of equations for an LTS M that is described in Müller's work assumes that M has only a single starting state. It proceeds by creating a variable X_q and a formula ϕ_q for every state $q \in M_Q$ such that

$$X_q = \phi_q \overset{\text{def}}{=} \bigwedge \left\{ \bigwedge \left\{ \langle a \rangle X_{q'} \mid q \xrightarrow[M_\delta]{a} q' \right\} \mid a \in M_A \right\} \wedge \bigwedge \left\{ [a] \bigvee \left\{ X_{q'} \mid q \xrightarrow[M_\delta]{a} q' \right\} \mid a \in M_A \right\}$$

With this definition, and the assumption that q_0 is the starting state of M, the characteristic system of equations of M is given by

$$\begin{aligned} X_{q_0} &= \phi_{q_0} \\ X_{q_1} &= \phi_{q_1} \\ &\vdots \\ X_{q_n} &= \phi_{q_n} \end{aligned}$$

where q_1 through q_n are the remaining (non-starting) states of M.

The problem with this approach is that only a single start state is assumed to be present for the model that is being analyzed. Moreover, their approach focuses on finding a property representative of the strong bisimulation relation. Therefore, in this work, their approach is split into two parts, corresponding to the two directions of simulation in a bisimulation relation constraint. These two parts form the basis of finding $E_{\rightsquigarrow}$ and $E_{\leftrightsquigarrow}$, which are systems of equations that can be transformed into ϕ_n and ϕ_s respectively. The process of finding these two systems is discussed below.

4.3.1 Necessary Condition

The aim here is to find a condition, which is satisfied by any model M such that M_{trans} simulates M. Since $L(M_{trans}) \subseteq L(\mathcal{R}^\kappa_\omega)$, for such a model M, $\mathcal{R}^\kappa_\omega$ would also simulate M. This fact is denoted by the notation $\mathcal{R}^\kappa_\omega \rightsquigarrow M$. One can consider the derivation process of the characteristic equations given in Müller's [34] work in order to find such a condition. In their work, the property corresponds to a bisimulation relation. In order to find the property corresponding to the uni-directional simulation relation, the derivation process discussed in their work can be simplified. In order to understand this process, one must first grasp the concept of a functional corresponding to a system of μ-calculus equations. Given such a system S, a functional F^S_M corresponding to S being evaluated on model M can be defined as

$$F^S_M\colon Env_M \to Env_M \text{ such that } F^S_M(\rho)(X_i) = [\![\phi_i]\!]_\rho \text{ for i = 1, \dots, n}$$

where

1. Env_M is the set of mappings from the variables X_i of S to the set of states of N. The mapping η resulting from the evaluation of S on N is an element of Env_M.
2. S contains the equations $X_i = \phi_i$ for $i = 1, \dots, n$.

With this knowledge, consider that the goal, for all states s in $\mathcal{R}^\kappa_\omega$, to find formulae $\phi^{\rightsquigarrow}_s$ corresponding to variables X_s in a system of equations $E_{\rightsquigarrow}$ such that the largest solution of $E_{\rightsquigarrow}$ for a given LTS N is η. Further, it must hold that if $\mathcal{R}^\kappa_\omega \rightsquigarrow N$, then

$$U = \bigvee_{s \in \mathcal{R}^\kappa_{\omega,Q_0}} X_s \implies \eta(U) \subseteq N_{Q_0} \tag{4.2}$$

This goal can be achieved by defining these formulae such that

$$\forall_{s \in \mathcal{R}^{\kappa}_{\omega,Q}} \eta(X_s) = \{r \in N_Q \mid s \rightsquigarrow r\} \tag{4.3}$$

This is because, if the formulae were to satisfy such a criteria, then the set of states in $\eta(X_s)$, where $s \in \mathcal{R}^{\kappa}_{\omega,Q_0}$, would contain those states that can be simulated by these states q in N. At least one of these states has to contain a starting state of N, in order for $\mathcal{R}^{\kappa}_{\omega}$ to be able to simulate it. In other words, $\eta(U) = \eta(\bigvee_{s \in \mathcal{R}^{\kappa}_{\omega,Q_0}} X_s) = \bigcup_{s \in \mathcal{R}^{\kappa}_{\omega,Q_0}} \eta(X_s)$ must contain at least one starting state of N. This matches the condition that was described in Equation 4.2.

Now, consider that Env_N is the set of possible solutions of $E_{\rightsquigarrow}$. the mapping $\alpha\colon Env_N \to \mathcal{P}(\mathcal{R}^{\kappa}_{\omega,Q} \times N_Q)$ can be then defined as follows

$$\alpha(\rho) = \{(s,t) \in \mathcal{R}^{\kappa}_{\omega,Q} \times N_Q \mid t \in \rho(X_s)\}$$

Conversely, a mapping $\beta\colon \mathcal{P}(\mathcal{R}^{\kappa}_{\omega,Q} \times N_Q) \to Env_N$ can be defined as $(\beta(R))(X_s) = \{t \in N_Q \mid (s,t) \in R\}$. With these mappings, one can define $E_{\rightsquigarrow}$ such that the simulation functional $F_N^{E_{\rightsquigarrow}}$, as well as the functional corresponding to $E_{\rightsquigarrow}$, $F_N^{E_{\rightsquigarrow}}$, are equal up to the isomorphism induced by the pair of complementary mappings (α, β). In other words, the following condition has to hold

$$F_N^{E_{\rightsquigarrow}} = \beta \circ F_{\rightsquigarrow} \circ \alpha$$

$$\begin{aligned}
&\eta(E_{\rightsquigarrow})(X_s) && \\
&= \nu(F_N^{E_{\rightsquigarrow}})(X_s) && \text{Definition of } \eta \\
&= \beta(\nu F_{\rightsquigarrow})(X_s) && \text{Isomorphism between fixpoints of } F_N^{E_{\rightsquigarrow}} \text{ and } F_{\rightsquigarrow} \\
&= \{t \in N_Q \mid (s,t) \in (\nu F_{\rightsquigarrow})\} && \text{Definition of } \beta \\
&= \{t \in N_Q \mid s \rightsquigarrow t\} && \rightsquigarrow \text{ equals } \nu F_{\rightsquigarrow}
\end{aligned}$$

This follows the requirement that was presented in equation 4.3. From this definition of $F_N^{E_{\rightsquigarrow}}$, it follows that for a state $t \in N_Q$

$$\begin{aligned}
& t \in [\![\phi_s^{\rightsquigarrow}]\!]_{\eta,N} \iff t \in (\beta \circ F_{\rightsquigarrow} \circ \alpha)(\rho)(X_s) && \\
\iff & \exists_{s \in \mathcal{R}^{\kappa}_{\omega,Q}} (s,t) \in (F_{\rightsquigarrow} \circ \alpha)(\rho) && \text{Definition of } \beta \\
\iff & \forall_{a \in N_A} \forall_{t' \in N_Q} t \xrightarrow[N_\delta]{a} t' \implies \exists_{s' \in \mathcal{R}^{\kappa}_{\omega,Q}} s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s' \wedge (s',t') \in \alpha(\rho) && \text{Definition of } F_{\rightsquigarrow} \\
\iff & \forall_{a \in N_A} \forall_{t' \in N_Q} t \xrightarrow[N_\delta]{a} t' \implies \exists_{s' \in \mathcal{R}^{\kappa}_{\omega,Q}} s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s' \wedge t' \in \eta(X_{s'}) && \text{Definition of } \alpha \\
\iff & \forall_{a \in N_A} \forall_{t' \in N_Q} t \xrightarrow[N_\delta]{a} t' \implies t' \in \left[\!\!\left[\bigvee_{s' \in \mathcal{R}^{\kappa}_{\omega,Q} \wedge s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} X_{s'} \right]\!\!\right]_{N,\eta} && \text{Definition of } \bigvee \\
\iff & \forall_{a \in N_A} t \in \left[\!\!\left[[a] \bigvee_{s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} X_{s'} \right]\!\!\right]_{N,\eta} && \text{Definition of } [\![\cdot]\!] \\
\iff & t \in \left[\!\!\left[\bigwedge_{a \in N_A} [a] \bigvee_{s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} X_{s'} \right]\!\!\right]_{N,\eta} && \text{Definition of } \bigwedge
\end{aligned}$$

By definition, $\eta(X_s) = [\![\phi_s^{\rightsquigarrow}]\!]_{N,\eta}$ and $t \in \eta(X_s)$. Therefore, the following conclusion can be obtained

$$\phi_s^{\rightsquigarrow} = \bigwedge_{a \in N_A} [a] \bigvee_{s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} X_s$$

In summary, then, the following system of equations describes the necessary condition that was desired

$$\begin{aligned}
U &= \bigvee_{s \in \mathcal{R}^{\kappa}_{\omega,Q_0}} X_s \\
X_{s_0} &= \bigwedge_{a \in N_A} [a] \bigvee_{s' \in \mathcal{R}^{\kappa}_{\omega,Q} \wedge s_0 \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} X_{s'} \\
&\vdots \\
X_{s_n} &= \bigwedge_{a \in N_A} [a] \bigvee_{s' \in \mathcal{R}^{\kappa}_{\omega,Q} \wedge s_n \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} X_{s'}
\end{aligned}$$

where $s_0, \ldots, s_n$ are the states of $\mathcal{R}^{\kappa}_{\omega}$.

4.3.2 Sufficient Condition

A sufficient condition has to be a system of equations $E_{\leftsquigarrow}$ such that if a model M satisfies said condition, then it is guaranteed that $\mathcal{R}^{\kappa}_{\omega}$ can be simulated by M, that is, $\mathcal{R}^{\kappa}_{\omega} \leftsquigarrow M$. Since $L(\mathcal{R}^{\kappa}_{\omega}) \supseteq L(E_{\leftsquigarrow})$, satisfying such a system of equations would also imply that $M_{trans} \leftsquigarrow M$.- The derivation of a sufficient condition follows a similar logic as that of the necessary one, resulting in a functional $F_N^{E_{\leftsquigarrow}}$ of $E_{\leftsquigarrow}$ having the same restriction as in the previous case, with $F_{\leftsquigarrow}$, the "simulated by" functional replacing $F_{\rightsquigarrow}$.

$$F_N^{E_{\leftsquigarrow}} = \beta \circ F_{\leftsquigarrow} \circ \alpha$$

With this, a derivation for the formula ϕ_s corresponding to state s of $\mathcal{R}^{\kappa}_{\omega}$ can be made as follows

$$\begin{aligned}
& t \in [\![\phi_s^{\rightsquigarrow}]\!]_{\eta,N} \iff t \in (\beta \circ F_{\rightsquigarrow} \circ \alpha)(\rho)(X_s) \\
\iff & \exists_{s \in \mathcal{R}^{\kappa}_{\omega,Q}} (s,t) \in (F_{\rightsquigarrow} \circ \alpha)(\rho) && \text{Definition of } \beta \\
\iff & \forall_{a \in \mathcal{R}^{\kappa}_{\omega,A}} \forall_{s' \in \mathcal{R}^{\kappa}_{\omega,Q}} s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s' \implies \exists_{t' \in N_Q} t \xrightarrow[N_\delta]{a} t' \wedge (s',t') \in \alpha(\rho) && \text{Definition of } F_{\leftsquigarrow} \\
\iff & \forall_{a \in \mathcal{R}^{\kappa}_{\omega,A}} \forall_{s' \in \mathcal{R}^{\kappa}_{\omega,Q}} s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s' \implies \exists_{t' \in N_Q} t \xrightarrow[N_\delta]{a} t' \wedge t' \in \eta(X_{s'}) && \text{Definition of } \alpha \\
\iff & \forall_{a \in \mathcal{R}^{\kappa}_{\omega,A}} \forall_{s' \in \mathcal{R}^{\kappa}_{\omega,Q}} s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s' \implies t \in [\![\langle a \rangle X_{s'}]\!]_{\eta,N} && \text{Definition of } \langle \cdot \rangle \\
\iff & \forall_{a \in \mathcal{R}^{\kappa}_{\omega,A}} t \in \left[\!\!\left[\bigwedge_{s' \in \mathcal{R}^{\kappa}_{\omega,Q} \wedge s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} \langle a \rangle X_{s'} \right]\!\!\right]_{\eta,N} && \text{Definition of } \bigwedge \\
\iff & t \in \left[\!\!\left[\bigwedge_{s' \in \mathcal{R}^{\kappa}_{\omega,Q} \wedge s \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} \langle a \rangle X_{s'} \right]\!\!\right]_{N,\eta} && \text{Definition of } \bigwedge
\end{aligned}$$

Again, analogous to the previous case, the conclusion is that for each state $s \in \mathcal{R}^{\kappa}_{\omega,Q}$, $X_s = \phi_s$. Further, s is simulated by the states in $\eta(X_s) = [\![\phi_s]\!]_{N,\eta}$. For $\mathcal{R}^{\kappa}_{\omega}$ to be simulated by N, a *sufficient* condition is that all of its starting states are simulated by some starting state of N. This is represented by a variable U

$$U = \bigwedge_{s \in \mathcal{R}^{\kappa}_{\omega,Q_0}} X_s$$

The condition described above is achieved if $\eta(U) = \bigcap_{s \in \mathcal{R}^{\kappa}_{\omega,Q_0}} \eta(X_s)$ contains at least one starting state of N. Thus, the system of equations for the sufficient condition is given by

$$\begin{aligned}
U &= \bigwedge_{s \in \mathcal{R}^{\kappa}_{\omega,Q_0}} X_s \\
X_{s_0} &= \bigwedge_{s' \in \mathcal{R}^{\kappa}_{\omega,Q} \wedge s_0 \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} \langle a \rangle X_{s'} \\
&\vdots \\
X_{s_n} &= \bigwedge_{s' \in \mathcal{R}^{\kappa}_{\omega,Q} \wedge s_n \xrightarrow[\mathcal{R}^{\kappa}_{\omega,\delta}]{a} s'} \langle a \rangle X_{s'}
\end{aligned}$$

where $s_0, \ldots, s_n$ are the states of $\mathcal{R}^{\kappa}_{\omega}$.

After $X_{\leadsto}$ and $X_{\leftrightsquigarrow}$ are obtained, they are converted into formulae ϕ_n and ϕ_s by the use of Gaussian elimination. In the algorithm, this happens in the procedure GET_CHARACTERISTIC_FORMULA, to obtain a formula in the variable *charFormula* from an equation system *chEqSystem*, for every starting state (`line 7`). The parameter *type* is used in several procedures to determine if they are supposed to perform actions that lead towards obtaining a sufficient or a necessary condition. At the top-level, the MAIN procedure calls EXTRACT twice, with two different values for *type*, "necessary" and "sufficient". This choice is propagated as these procedures are executed, so that eventually, for the case of the necessary condition, a logical union of the variables corresponding to the start states of $\mathcal{R}^{\kappa}_{\omega}$ is returned (`line 15`). On the other hand, for the case where "type" is sufficient, their intersection is returned (`line 17`). Further, in the GET_EQUATION function which is tasked with obtaining the ϕ_s's corresponding to states s of $\mathcal{R}^{\kappa}_{\omega}$, there is a distinction made in the kind of formula obtained when type is "necessary" (`line 40`) versus when it is sufficient (`line 42`).

Whenever a μ-caculus formula is created or modified, at any point during this step, a compaction process is applied on the formula. This is done by removing redundant operators and/or operands in intersection or union of sub-formulae of a given formula, in the MAKE_COMPACT procedure. This effect is achieved by recursively parsing a given formula named $formula$, and finding sub-formulae consisting of a sequence of intersection or union operators. Wherever this is found, the following steps are taken with respect to the list of sub-formulae (*subFormulae*) whose intersection/union is being considered. This is done by comparing members of *subFormulae* viz. f_1, f_2 where f_2 is located after f_1 in *subFormulae*. The process is repeated in its entirety as long as changes keep happening (`line 73`).

1. If f_1 and f_2 are logically equivalent (`line 85`), delete f_2 from *subFormulae* (`line 86`).

2. If f_1 is a diamond operator, f_2 a box operator, and $\Box == \bigwedge$ ($\Box$ is the intersection operator) with the same actions and contained formulae, then the formula of f_1 can be replaced with TRUE. This follows from the semantic equivalence of $[a]\phi \wedge \langle a \rangle \phi$ and $[a]\text{TRUE} \wedge \langle a \rangle \phi$. This is illustrated by `lines 87-90`.

3. If $f_1 == [a]\phi$ and $f_2 == [b]\phi$, and $\Box == \bigwedge$, then f_2 is deleted, and the action b is incorporated into f_1 by changing it to $[a, b]\phi$. This can be interpreted as

$$[\![[a, b]\phi]\!]_{M,\eta} = \{q \mid \forall_{s',s'' \in M_Q} (s \xrightarrow[M_\delta]{a} s' \vee s \xrightarrow[M_\delta]{b} s'') \implies s', s'' \in [\![\phi]\!]_{M,\eta}\}$$

This process is done on `lines 93-95`.

4. A similar process can be done for the box operator, as demonstrated on `lines 96-98`. This results in formulae of the form $\langle a, b\rangle\phi$ which can be interpreted as follows

$$[\![[a, b]\phi]\!]_{M,\eta} = \{q \mid \exists_{s',s''\in M_Q} s \xrightarrow[M_\delta]{a} s' \wedge s \xrightarrow[M_\delta]{b} s'' \wedge s', s'' \in [\![\phi]\!]_{M,\eta}\}$$

5. Finally, if f_1 and f_2 are both diamond or box operators and contain the same action, but have different formulae that they point towards, then they are grouped into a single set, provided that an *intersection* of the formulae in *subFormulae* is under consideration. This is done by using a map named *compounded*(`line 105`). Later, in `lines 112-120`, these groups are combined by placing their(diamond or box) operator *outside* the $\wedge$ operator of the formulae. In effect the process (in the case of the box operator, for instance) uses the following equivalence

$$[a]\phi_1 \wedge [a]\phi_2 \wedge \ldots [a]\phi_n \equiv [a]\phi_1 \wedge \phi_2 \ldots \wedge \phi_n$$

Algorithm 3 Algorithm for extracting the necessary and sufficient criterion from $\mathcal{R}^\kappa_\omega$

```
1: procedure MAIN($\mathcal{R}^\kappa_\omega$)
2:     return EXTRACT($\mathcal{R}^\kappa_\omega$, "necessary"), EXTRACT($\mathcal{R}^\kappa_\omega$, "sufficient")
3: end procedure
4: procedure EXTRACT($M$, $type$)
5:     $subFormulae \leftarrow []$
6:     for $s \in M_{Q_0}$ do
7:         $M_s \leftarrow M$
8:         $M_{s,Q_0} \leftarrow \{s\}$
9:         $chEqSystem \leftarrow$ GET_CHARACTERISTIC_EQUATION_SYSTEM($M_s, type$)
10:        $charFormula \leftarrow$ GET_CHARACTERISTIC_FORMULA($chEqSystem$)
11:        $charFormula \leftarrow$ MAKE_COMPACT($charFormula$)
12:        Append $charFormula$ to $subFormulae$
13:    end for
14:    if $type$ is "necessary" then
15:        return Union of the formulae in list $subFormulae$
16:    else
17:        return Intersection of the formulae in list $subFormulae$
18:    end if
19: end procedure
20: procedure GET_CHARACTERISTIC_EQUATION_SYSTEM($G$, $type$) ▷ Returns a list consisting of
    pairs of a variable and the formulae that it equates to
21:    $equations \leftarrow []$
22:    $variables \leftarrow Map()$
23:    for $s \in G_Q$ do
24:        Create a new free variable $X_s$
25:        Let $variables(s)$ be $X_s$
26:    end for
27:    for $v \in variables$ do
28:        $pair \leftarrow (v,$ GET_EQUATION($variables, G, s, type$))
29:        Append $pair$ to $equations$
30:    end for
31:    Move the pair corresponding to the initial state of $G$ to the beginning of $equations$
32:    for $var, equation \in equations$ do
33:        $equation \leftarrow$ MAKE_COMPACT($equation$)
34:    end for
```

```
end procedure
procedure GET_EQUATION($variables$, $G$, $s$, $type$)
    $formulaList \leftarrow []$
    for $a \in G_A$ do
        if $type$ is “necessary” then
            $f \leftarrow \bigwedge_{s \xrightarrow[G_\delta]{a} s_1 \text{ where } s_1 \in G_Q} < a > variables\,(s_1)$
        else
            $f \leftarrow [a]bigvee_{s \xrightarrow[G_\delta]{a} s_1 \text{ where } s_1 \in G_Q}\,(variables\,(s_1))$
        end if
        Append $f$ to $formulaList$
    end for
    return Intersection of all formulae in $formulaList$ and $rightFormulae$
end procedure
procedure GET_CHARACTERISTIC_FORMULA($system$) ▷ Perform Gaussian elimination to get
a single $\mu$-calculus formula
    while $|system| > 1$ do
        $system \leftarrow$ APPLY_RULE_1($system$)
        $system \leftarrow$ APPLY_RULE_2($system$)
        $system \leftarrow$ APPLY_RULE_3($system$)
    end while
    $system \leftarrow$ APPLY_RULE_1($system$)
    return $system$
end procedure
procedure APPLY_RULE_1($equations$)                          ▷ Get the last equation
    $X, \phi \leftarrow equations_{|equations|-1}$
    $\psi \leftarrow \nu X.\phi$         ▷ Replace the last equation and bind its variable with $\nu$
    $equations_{|equations|-1} \leftarrow (var, \psi)$
    return $equations$
end procedure
procedure APPLY_RULE_2($equations$)                                   ▷ Substitution
    $X, \phi \leftarrow equations_{|equations|-1}$                        ▷ Get the last equation
    for $i \in \{0, \ldots, |equations| - 2\}$ do
        Replace every occurrence of variable $X$ in formula $equations_{i,1}$ with $\phi$
    end for
    return $equations$
end procedure
procedure APPLY_RULE_3($equations$)                          ▷ Delete last equation
    Delete $equations_{|equations|-1}$ from $equations$
    return $equations$
end procedure
procedure MAKE_COMPACT($formula$)
    Let $\square$ denote the type of the top-level operand of $formula$
    if $\square$ is union or intersection then
        Let $subFormulae$ be the list of formulae whose combination produces $formula$
        for $subFormula \in subFormuae$ do
            MAKE_COMPACT($subFormula$)
        end for
        while Changes are happening to $subFormulae$ do
            $compounded \leftarrow Map()$
            for $i, j \in \{1, \ldots, |subFormulae| - 1\}$ do
                if $i < j$ then
                    if $subFormulae_i = subFormulae_j$ then
```

```
                        Delete index $j$ from $subFormulae$
                    else if $\Box == \bigwedge \wedge \exists_{a,\phi_1} subFormulae_i = [a]\phi_1 \wedge subFormulae_j = \langle a \rangle \phi_1$ then
                        $subFormulae_j \leftarrow \langle a \rangle$ TRUE
                    else if $\Box == \bigwedge \wedge \exists_{a,\phi_1} subFormulae_i = \langle a \rangle \phi_1 \wedge subFormulae_j = [a]\phi_1$ then
                        $subFormulae_i \leftarrow \langle a \rangle$ TRUE
                    else if Both $subFormulae_i$ and $subFormulae_j$ have the same operator $\diamond$ then
                        if $diamond == [\cdot] \vee diamond == \langle \cdot \rangle$ then
                            if $\exists_{a,\phi_1} subFormulae_i = [a]\phi_1 \wedge subFormulae_j = [b]\phi_1$ then
                                Delete $subFormulae_j$
                                $subFormula_i \leftarrow [a,b]\phi_1$
                            else if $\exists_{a,\phi_1} subFormulae_i = \langle a \rangle \phi_1 \wedge subFormulae_j = \langle b \rangle \phi_1$ then
                                Delete $subFormulae_j$
                                $subFormula_i \leftarrow \langle a,b \rangle \phi_1$
                            else if Actions of $subFormulae_i$ and $subFormulae_j$ are the same then
                                Let $a$ be the common action
                                Let $\phi_1, \phi_2$ be the sub-formulae contained in $subFormulae_i$ and $subFormulae_j$
                                if $i$ is not mapped in $compounded$ then
                                    $compounded_i \leftarrow \emptyset$
                                end if
                                Add $j$ to $compunded_i$
                                Delete $subFormulae_j$
                            end if
                        end if
                    end if
                end if
            end for
            for $i \in compounded$ do
                Let $a$ be the action of $subFormulae_i$
                Let $f_k$ be the formula contained in $subFormula_k$ in the list $subFormula$
                if $subFormulae_i$ is a box operator formula then
                    $subFormulae_i \leftarrow [a]\Box_{j \in compounded_i} f_j$
                else if $subFormulae_i$ is a diamond operator formula then
                    $subFormulae_i \leftarrow \langle a \rangle \Box_{j \in compounded_i} f_j$
                end if
            end for
        end while
        Replace $formula$ by $\Box_{f \in subFormula} f$
    else if $\Box$ is $\mu$, $\nu$, $[\cdot]$, or $\langle \cdot \rangle$ then
        Let $f$ be the sub-formula contained in $formula$
        MAKE_COMPACT($f$)
    end if
end procedure
```

Chapter 5

Experimental Results

This chapter describes two experiments with the tool that were developed on the basis of the methodology described in the previous chapter.

For the first step, in the experiments that follow, it is assumed that state 0 of the models corresponding to the properties that are satisfied by the source models (viz. G_ω), is to be matched with κ, that is, $s^\kappa = 0$. For a complete analysis, s^κ can be set to every state of G_ω, and solutions can be obtained for each such assignment.

5.1 First Experiment : A Simple Example

5.1.1 Dataset

This dataset utilizes only two actions that are named a and b. The transformation rule to be considered is the same as the one introduced in the section on model transformations, section 2.4.1, that was described by Figure 2.2.

The formula ω assumed to be satisfied on the source model is as follows:

$$\omega = aUb$$

The Büchi Automaton corresponding to ψ is shown in Figure 5.1.

5.1.2 Results

The three step procedure discussed in the methodology is applied on this dataset.

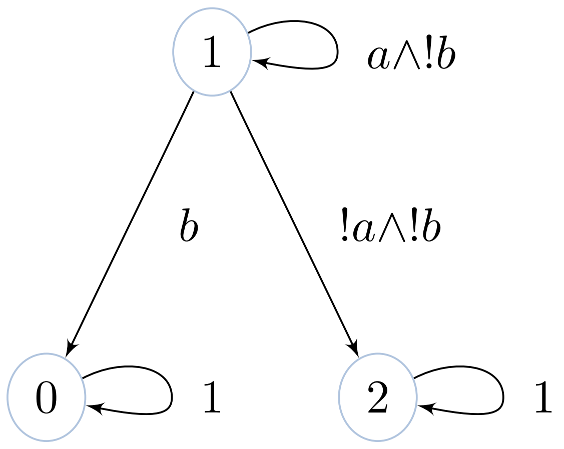

Figure 5.1: G_ψ : Büchi Automata corresponding to ψ, state 0 is a final state

After the first step is applied, 126 refined versions of $\mathcal{L}^\kappa$ are produced. The tool LTScompare [33] was used to test the bisimilarity of these solutions. Since the tool only accepts models with one starting state in the Aldebaran format, which was used for its simplicity, some modifications were made to the models resulting from the first step, which had multiple start states. This transformation went as follows. A dummy starting state was created for each model, and τ-transitions were extended from this state to the original start states, which were then designated as original states. Then, the models resulting from this were compared using *weak* bisimulation. Since, in this case, exactly τ-transitions is present at the start of every trace, these transitions are effectively ignored during the comparison process. It was concluded that all solutions were equivalent modulo weak bisimulation to each other. Arbitrarily, solution 10 was elected for further analysis. The LTS of this solution is displayed in Figure 5.2.

This model is then transformed by applying the rule represented by the pair $(\mathcal{L}^\kappa, \mathcal{R}^\kappa)$ shown in Figure 5.1. The resulting transformed model is shown in Figure 5.3.

$\mathcal{R}^\kappa_\psi$ is then minimized as follows.

1. A new state S is added to a copy of $\mathcal{R}^\kappa_\psi$, named $\mathcal{R}^\kappa_S$.
2. τ-transitions are added from S to the starting states of $\mathcal{R}^\kappa_S$
3. S is now marked as the only starting state of $\mathcal{R}^\kappa_S$. Then, $\mathcal{R}^\kappa_\psi$ and $\mathcal{R}^\kappa_S$ are weakly bisimilar, as the added τ-transitions are ignored when considering weak bisimilarity.
4. $\mathcal{R}^\kappa_S$ is minimized modulo weak bisimulation using the LTSconvert tool of the mCRL2 [33] toolset, producing M. Note that the prior steps had ensured that $\mathcal{R}^\kappa_S$ only had a single starting state, which is required by mCRL2.
5. Steps 1, 2, and 3 are now applied in reverse on M, producing $\mathcal{R}^\kappa_{min}$.

$\mathcal{R}^\kappa_{min}$ (seen in Figure 5.4) is strongly bisimilar to $\mathcal{R}^\kappa_\psi$. This is because the τ-transitions in the minimized version as well as the original one remained in the same relative position — pointing from an artificial starting state to the real starting states of the models.

Hence, it can now be used to obtain the characteristic system of equations corresponding to the necessary and sufficient conditions, as discussed in the third step of the methodology. This system of equations corresponding to the sufficient condition($E_{\leftrightarrow}$) is as follows

$$\begin{aligned}
&U_0 = S_4 \wedge S_8\\
&S_0 = \langle\{b\}\rangle S_3 \wedge \langle\{a\}\rangle\,(S_1 \wedge S_2)\\
&S_1 = \langle\{b\}\rangle\,(S_0 \wedge S_4) \wedge \langle\{a\}\rangle S_1 \wedge \langle\{a,\,b\}\rangle S_3\\
&S_2 = \langle\{b\}\rangle S_1 \wedge \langle\{a\}\rangle S_0\\
&S_3 = \langle\{a\}\rangle\,(S_3 \wedge S_0 \wedge S_1)\\
&S_4 = \langle\{b\}\rangle\,(S_4 \wedge S_3)\\
&S_5 = \langle\{b\}\rangle S_8 \wedge \langle\{a\}\rangle\,(S_6 \wedge S_7)\\
&S_6 = \langle\{b\}\rangle\,(S_5 \wedge S_9) \wedge \langle\{a\}\rangle S_6 \wedge \langle\{a,\,b\}\rangle S_8\\
&S_7 = \langle\{b\}\rangle S_6 \wedge \langle\{a\}\rangle S_5\\
&S_8 = \langle\{a\}\rangle\,(S_8 \wedge S_5 \wedge S_6)\\
&S_9 = \langle\{b\}\rangle\,(S_9 \wedge S_8)
\end{aligned}$$

If one considers the set of equations corresponding to variables S_4, S_5, S_6, and S_7 and replaces those variables(as well as their references) with, respectively, variables S_0, S_1, S_2, and S_3, the latter equations become identical to the former. This replacement can be done without any conflicts because no formulae, other than those that correspond to the former set of variables, refer to those variables in their body. Thus, on evaluating the given system on any model, $\eta(S_5) = \eta(S_1)$. In other words, the equation $W = S_0 \wedge S_5$ can be simplified to $W = S_0 \wedge S_1$. With this modification, after removing redundant equations, the system becomes

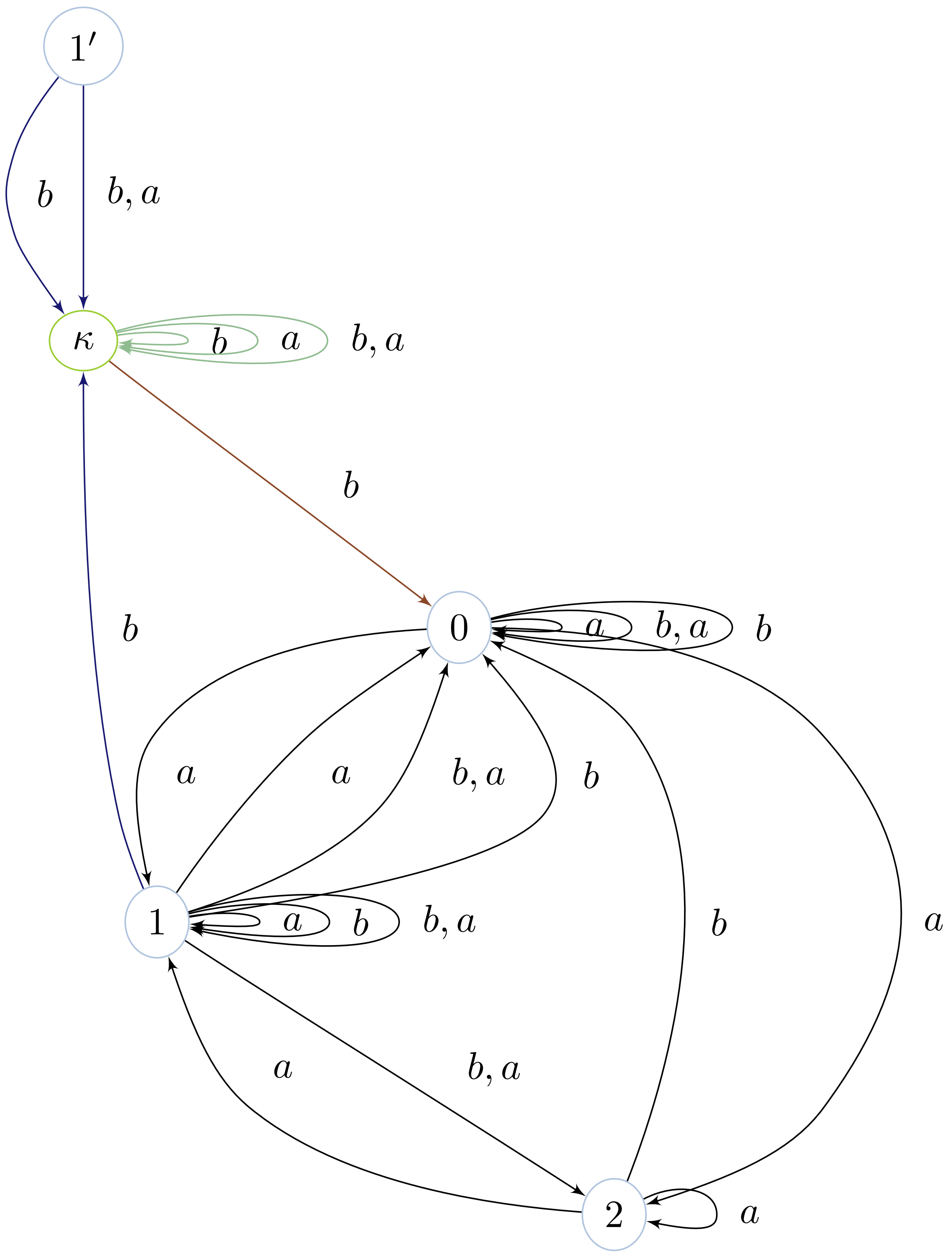

Figure 5.2: $\mathcal{L}^\kappa$ refined with the information in ψ : $\mathcal{L}^\kappa_\psi$

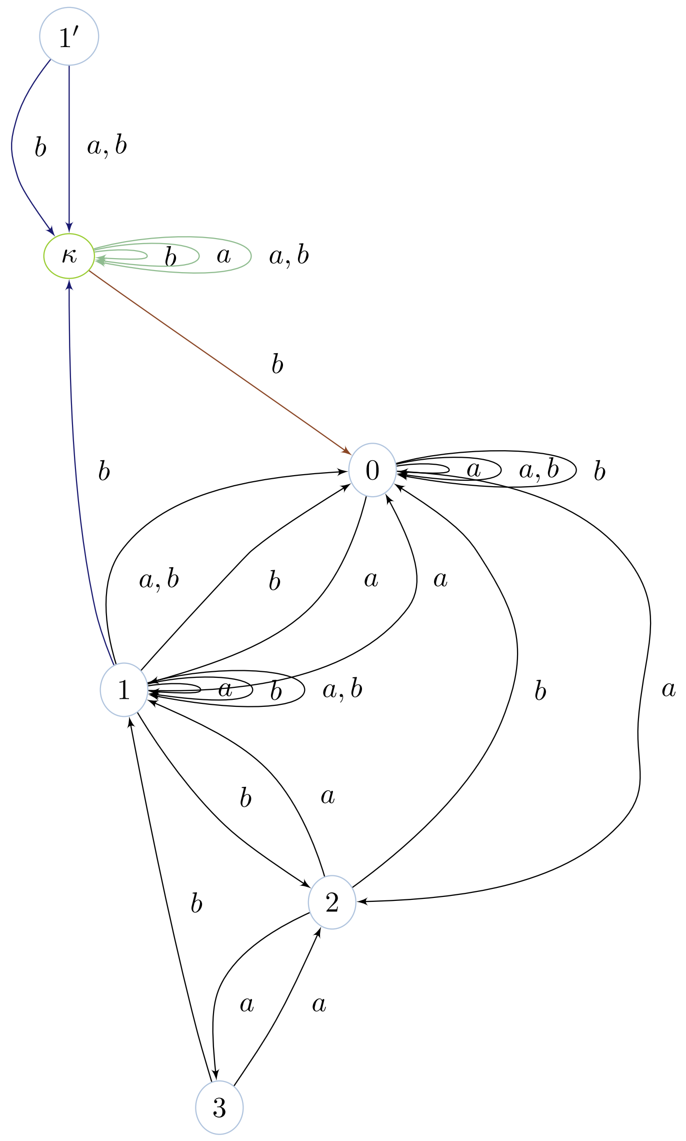

Figure 5.3: $\mathcal{R}^\kappa$ refined with the information in ψ : $\mathcal{R}^\kappa_\psi$

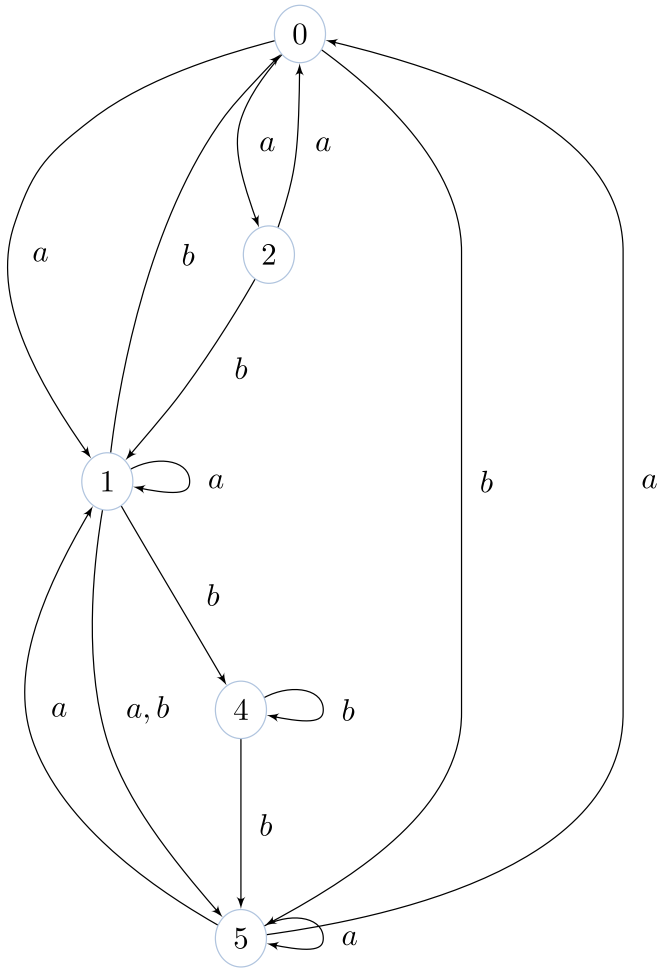

Figure 5.4: $\mathcal{R}_{\psi}^{\kappa}$ after minimization : $\mathcal{R}_{min}^{\kappa}$

$$\begin{aligned}
&W = S_3 \wedge S_4\\
&S_0 = \langle\{a\}\rangle\,(S_1 \wedge S_2) \wedge \langle\{b\}\rangle S_3\\
&S_1 = \langle\{a,\, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\,(S_0 \wedge S_4)\\
&S_2 = \langle\{a\}\rangle S_0 \wedge \langle\{b\}\rangle S_1\\
&S_3 = \langle\{a\}\rangle\,(S_3 \wedge S_0 \wedge S_1)\\
&S_4 = \langle\{b\}\rangle\,(S_4 \wedge S_3)
\end{aligned}$$

This transformation of identifying multiple equation sets and deleting redundant sets has not yet been made automatic for our tool. Hence, this modification was obtained manually and passed as hard-coded input into our system for the Gaussian elimination step.

The system of equations $E_{\rightsquigarrow}$ corresponding to the necessary condition, on the other hand, is

$$\begin{aligned}
&W = S_4 \vee S_8\\
&S_0 = [\{b\}]S_3 \wedge [\{a\}]\,(S_1 \vee S_2) \wedge [\{a,\, b\}]\textsc{false}\\
&S_1 = [\{b\}]\,(S_0 \vee S_4) \wedge [\{a\}]S_1 \wedge [\{a,\, b\}]S_3\\
&S_2 = [\{b\}]S_1 \wedge [\{a\}]S_0 \wedge [\{a,\, b\}]\textsc{false}\\
&S_3 = [b]\textsc{false} \wedge [\{a\}]\,(S_3 \vee S_0 \vee S_1)\\
&S_4 = [\{b\}]\,(S_4 \vee S_3) \wedge [a]\textsc{false}\\
&S_5 = [\{b\}]S_8 \wedge [\{a\}]\,(S_6 \vee S_7) \wedge [\{a,\, b\}]\textsc{false}\\
&S_6 = [\{b\}]\,(S_5 \vee S_9) \wedge [\{a\}]S_6 \wedge [\{a,\, b\}]S_8\\
&S_7 = [\{b\}]S_6 \wedge [\{a\}]S_5 \wedge [\{a,\, b\}]\textsc{false}\\
&S_8 = [b]\textsc{false} \wedge [\{a\}]\,(S_8 \vee S_5 \vee S_6)\\
&S_9 = [\{b\}]\,(S_9 \vee S_8) \wedge [a]\textsc{false}
\end{aligned}$$

By following a process of simplification that is similar to the one described above, one can obtained a simplified version of $E_{\rightsquigarrow}$ given by

$$\begin{aligned}
&W = S_3 \vee S_4\\
&S_0 = [\{a,\, b\}]\textsc{false} \wedge [\{a\}]\,(S_1 \vee S_2) \wedge [\{b\}]S_3\\
&S_1 = [\{a,\, b\}]S_3 \wedge [\{a\}]S_1 \wedge [\{b\}]\,(S_0 \vee S_4)\\
&S_2 = [\{a,\, b\}]\textsc{false} \wedge [\{a\}]S_0 \wedge [\{b\}]S_1\\
&S_3 = [b]\textsc{false} \wedge [\{a\}]\,(S_3 \vee S_0 \vee S_1)\\
&S_4 = [a]\textsc{false} \wedge [\{b\}]\,(S_4 \vee S_3)
\end{aligned}$$

The characteristic formulae corresponding to these systems are described in the appendix, section A.

5.2 Second Experiment - Link layer of P1394

5.2.1 Dataset

The first analysis that is done utilizes a dataset provided by Luttik [35]. This dataset models the link layer protocol of P1394, which was defined as a standard for a “High Performance Serial Bus”. A simplified version of this model is considered in which action labels do not have any parameters. This simplification does not really affect the discussion that follows, since it does not affect the overall structure of the pattern LTSs being analyzed.

In this context, we consider a model transformation that is characterized by κ-extended patterns shown in Figure 5.5. This transformation essentially adds some $e1$-actions after the reception of a signal (denoted by $rPDind$). These are denoted by the path of $e1$-transitions $2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5$. For our purpose, it is sufficient to understand that these $e1$-actions do not represent communication operations for the P1394 system.

The property to be satisfied in this context is that of livelock prevention. This can be represented by the following LTL formula:

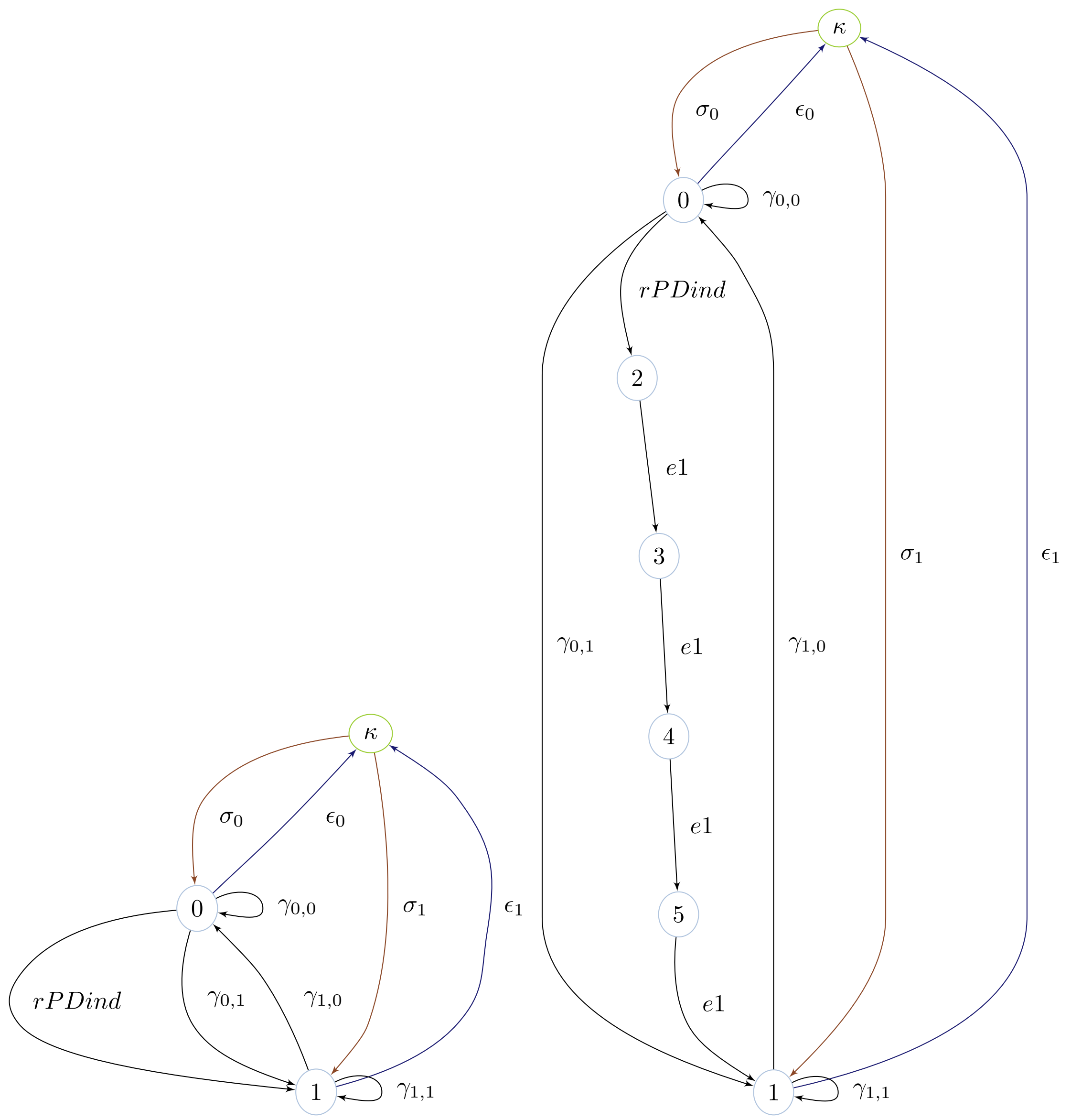

Figure 5.5: $\mathcal{L}^{\kappa}$(Left) and $\mathcal{R}^{\kappa}$(Right) of rule being applied

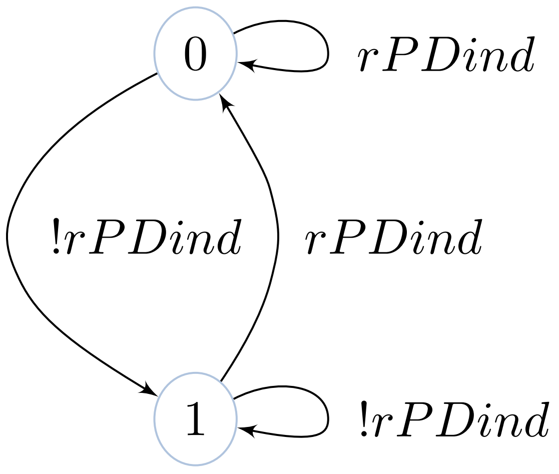

Figure 5.6: G_ω : Büchi Automata corresponding to ω, state 0 is a final state

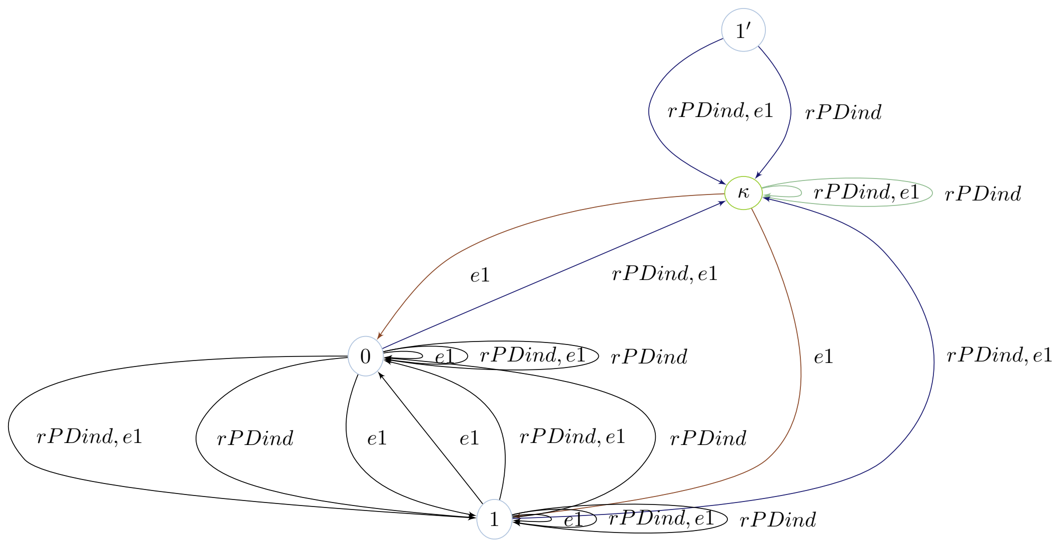

Figure 5.7: $\mathcal{L}^\kappa$ refined with the information in ω : $\mathcal{L}^\kappa_\omega$

$$\omega = \text{G F } rPDind$$

The reason as to why this formula accurately depicts the livelock condition is as follows. In the present context, it is sufficient $rPDind$ is an action that signifies the reception of a signal for a process following the IEEE-1394 specification. Thus, its occurrence indicates that the primary functionality of the system - data transfer, has not stalled out - and thus, there is no livelock. So, we have the condition that *always eventually* $rPDind$ holds.

The Büchi Automaton corresponding to ω is shown in Figure 5.6.

5.2.2 Results

The three step procedure discussed in the methodology is applied on this dataset.

After the first step is applied, 202 refined versions of $\mathcal{L}^\kappa$ are produced. As before, LTScompare [33] was used to test the bisimilarity of these solutions. Arbitrarily, solution 0 was elected for further analysis. The LTS of this solution is displayed in Figure 5.7.

This model is then transformed by applying the rule represented by the pair $(\mathcal{L}^\kappa, \mathcal{R}^\kappa)$ shown in Figure 5.5. The resulting transformed model is shown in Figure 5.8.

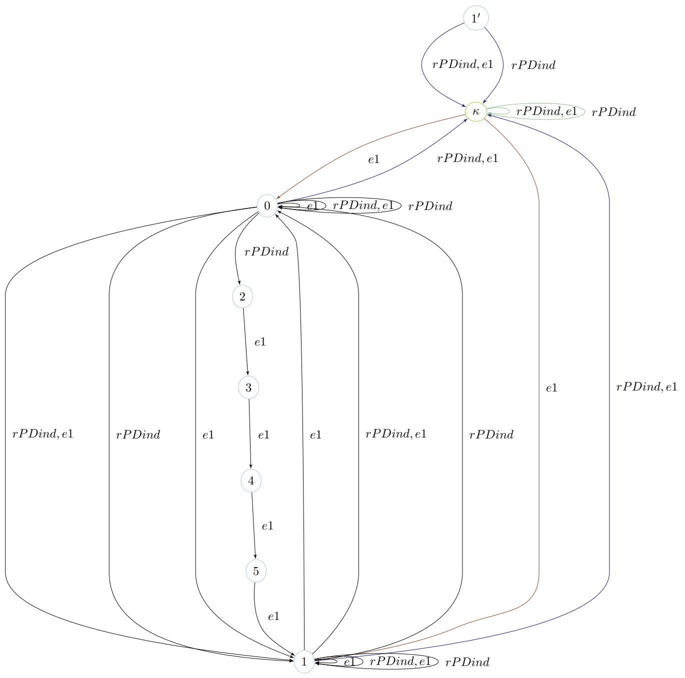

Figure 5.8: $\mathcal{R}^{\kappa}$ refined with the information in ω : $\mathcal{R}^{\kappa}_{\omega}$

Again, as with the previous example, this LTS can be minimized modulo weak bisimulation. After this is done, the resulting LTS that now represents $\mathcal{R}^{\kappa}_{\omega}$ is shown in Figure 5.9.

The system of equations $E_{\rightsquigarrow}$ corresponding to ϕ_n, the necessary condition, can be extracted from $\mathcal{R}^{\kappa}_{\omega}$, and is given by

$$
\begin{aligned}
U_0 &= S_3 \vee S_5 \vee S_6 \\
S_0 &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_1 \\
S_1 &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_2 \\
S_2 &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_3 \\
S_3 &= [\{rPDind\}]\,(S_3 \vee S_5) \wedge [\{e1,\, rPDind\}]S_6 \wedge [\{e1\}]\text{FALSE} \\
S_4 &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_0 \\
S_5 &= [\{rPDind\}]\,(S_3 \vee S_5) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]S_6 \\
S_6 &= [\{rPDind\}]\,(S_5 \vee S_4) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]\,(S_6 \vee S_3) \\
S_7 &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_8 \\
S_8 &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_9 \\
S_9 &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_{10} \\
S_{10} &= [\{rPDind\}]\,(S_{10} \vee S_{12}) \wedge [\{e1,\, rPDind\}]S_{13} \wedge [\{e1\}]\text{FALSE} \\
S_{11} &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_7 \\
S_{12} &= [\{rPDind\}]\,(S_{10} \vee S_{12}) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]S_{13} \\
S_{13} &= [\{rPDind\}]\,(S_{12} \vee S_{11}) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]\,(S_{13} \vee S_{10}) \\
S_{14} &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_{15} \\
S_{15} &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_{16} \\
S_{16} &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_{17} \\
S_{17} &= [\{rPDind\}]\,(S_{17} \vee S_{19}) \wedge [\{e1,\, rPDind\}]S_{20} \wedge [\{e1\}]\text{FALSE} \\
S_{18} &= [rPDind]\text{FALSE} \wedge [\{e1\}]S_{14} \\
S_{19} &= [\{rPDind\}]\,(S_{17} \vee S_{19}) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]S_{20} \\
S_{20} &= [\{rPDind\}]\,(S_{19} \vee S_{18}) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]\,(S_{20} \vee S_{17})
\end{aligned}
$$

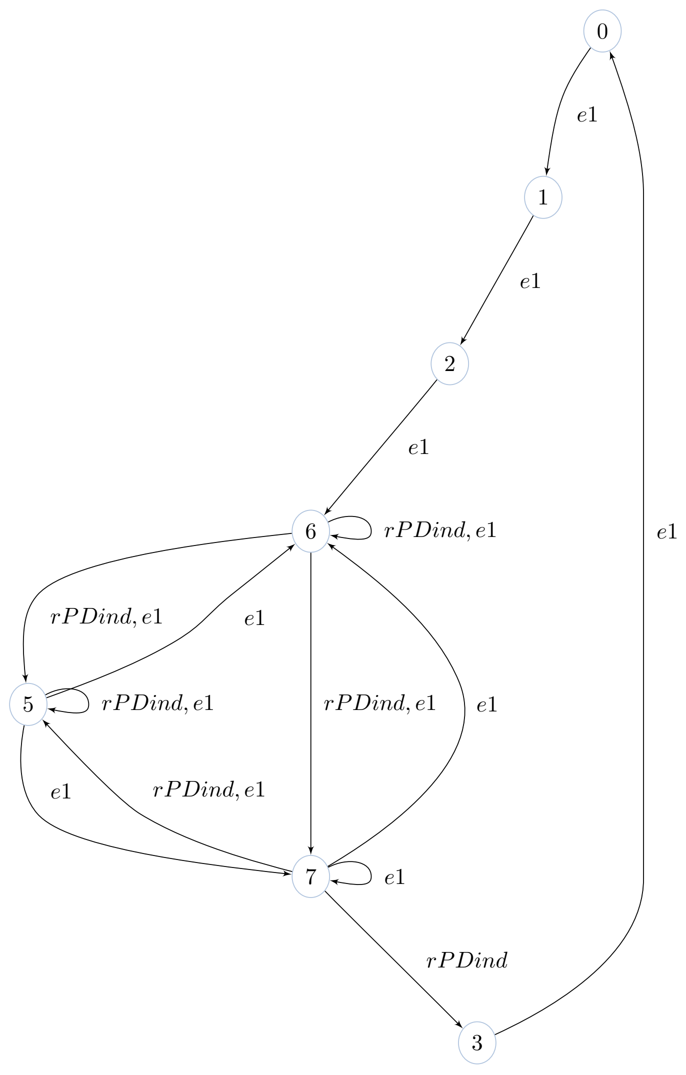

Figure 5.9: $\mathcal{R}^{\kappa}_{\omega}$ minimized modulo weak bisimulation

If one considers the set of equations corresponding to variables S_7, S_8, S_9, S_{10}, S_{11}, S_{12}, and S_{13}, and replaces those variables(as well as their references) with, respectively, variables S_0, S_1, S_2, S_3, S_4, S_5, and S_6 respectively, it can be noted that the latter equations become identical to the former. This replacement can be done without any conflicts because no formulae, other than those that correspond to the former set of variables, refer to those variables in their body. This process can also be repeated for the variables S_{17}, S_{16}, S_{18}, S_{19}, S_{20}, S_{21}, S_{22}, and S_{23}, with the same result. Therefore, on evaluating the given system on any model, $\eta(S_9) = \eta(S_1)$, and $\eta(S_{21}) = \eta(S_5)$ must hold. In other words, the equation $U = S_6 \wedge S_1 \wedge S_5$ can be simplified to $U = S_1 \wedge S_2 \wedge S_0$. With this modification, after removing redundant equations, the system becomes

$$\begin{aligned}
&U_0 = S_3 \vee S_4 \vee S_5\\
&S_0 = [\{e1\}]S_1 \wedge [rPDind]\text{FALSE}\\
&S_1 = [\{e1\}]S_2 \wedge [rPDind]\text{FALSE}\\
&S_2 = [\{e1\}]S_3 \wedge [rPDind]\text{FALSE}\\
&S_3 = [\{e1\}]S_5 \wedge [\{e1,\, rPDind\}]S_4 \wedge [\{rPDind\}]S_3\\
&S_4 = [(e_1 \wedge \neg rPDind) \vee (rPDind \wedge \neg e_1)]\text{FALSE} \wedge [\{e1,\, rPDind\}]\,(S_4 \vee S_5 \vee S_3)\\
&S_5 = [\{e1\}]\text{FALSE} \wedge [\{e1,\, rPDind\}]\,(S_5 \vee S_3 \vee S_4) \wedge [\{rPDind\}]S_6\\
&S_6 = [\{e1\}]S_0 \wedge [rPDind]\text{FALSE}
\end{aligned}$$

The system of equations $E_{\leftsquigarrow}$ corresponding to the sufficient conditions ψ_s is also computed. This system is as follows

$$\begin{aligned}
U_0 &= S_5 \wedge S_{13} \wedge S_{17}\\
S_0 &= \langle\{e1\}\rangle S_1\\
S_1 &= \langle\{e1\}\rangle S_2\\
S_2 &= \langle\{e1\}\rangle S_3\\
S_3 &= \langle\{rPDind\}\rangle\,(S_3 \wedge S_5) \wedge \langle\{e1,\, rPDind\}\rangle S_6\\
S_4 &= \langle\{e1\}\rangle S_0\\
S_5 &= \langle\{rPDind\}\rangle\,(S_3 \wedge S_5) \wedge \langle\{e1\}\rangle S_6\\
S_6 &= \langle\{rPDind\}\rangle\,(S_5 \wedge S_4) \wedge \langle\{e1\}\rangle\,(S_6 \wedge S_3)\\
S_7 &= \langle\{e1\}\rangle S_8\\
S_8 &= \langle\{e1\}\rangle S_9\\
S_9 &= \langle\{e1\}\rangle S_{10}\\
S_{10} &= \langle\{rPDind\}\rangle\,(S_{10} \wedge S_{12}) \wedge \langle\{e1,\, rPDind\}\rangle S_{13}\\
S_{11} &= \langle\{e1\}\rangle S_7\\
S_{12} &= \langle\{rPDind\}\rangle\,(S_{10} \wedge S_{12}) \wedge \langle\{e1\}\rangle S_{13}\\
S_{13} &= \langle\{rPDind\}\rangle\,(S_{12} \wedge S_{11}) \wedge \langle\{e1\}\rangle\,(S_{13} \wedge S_{10})\\
S_{14} &= \langle\{e1\}\rangle S_{15}\\
S_{15} &= \langle\{e1\}\rangle S_{16}\\
S_{16} &= \langle\{e1\}\rangle S_{17}\\
S_{17} &= \langle\{rPDind\}\rangle\,(S_{17} \wedge S_{19}) \wedge \langle\{e1,\, rPDind\}\rangle S_{20}\\
S_{18} &= \langle\{e1\}\rangle S_{14}\\
S_{19} &= \langle\{rPDind\}\rangle\,(S_{17} \wedge S_{19}) \wedge \langle\{e1\}\rangle S_{20}\\
S_{20} &= \langle\{rPDind\}\rangle\,(S_{19} \wedge S_{18}) \wedge \langle\{e1\}\rangle\,(S_{20} \wedge S_{17})
\end{aligned}$$

This system can be simplified by substituting variable names, as was done with the case of the system $E_{\rightsquigarrow}$. The resulting simplified system is

$$\begin{aligned}
&U_0 = S_3 \wedge S_4 \wedge S_5\\
&S_0 = \langle\{e1\}\rangle S_1\\
&S_1 = \langle\{e1\}\rangle S_2\\
&S_2 = \langle\{e1\}\rangle S_3\\
&S_3 = \langle\{e1\}\rangle S_5 \wedge \langle\{e1,\, rPDind\}\rangle S_4 \wedge \langle\{rPDind\}\rangle S_3\\
&S_4 = \langle\{e1,\, rPDind\}\rangle\,(S_4 \wedge S_5 \wedge S_3)\\
&S_5 = \langle\{e1,\, rPDind\}\rangle\,(S_5 \wedge S_3 \wedge S_4) \wedge \langle\{rPDind\}\rangle S_6\\
&S_6 = \langle\{e1\}\rangle S_0
\end{aligned}$$

The characteristic formulae obtained from these systems can be viewed in the appendix, section B.

Chapter 6

Discussions

While analyzing the results obtained in the previous chapter, either the necessary or the sufficient condition is chosen, as needed for further analysis in a particular context.

6.1 First Experiment

6.1.1 Motivation

The purpose of this first experiment is to demonstrate the capabilities of the current tool with a simple minimal example.

6.1.2 Inferences

It is known that the following assertion holds:

If a model M satisfies the system $E_{\leftrightsquigarrow}$ corresponding to this example, then M can simulate M_{trans}.

Let s be a starting state of a model M which satisfies $E_{\leftrightsquigarrow}$. Then, the following chain of logic holds:

$$
\begin{aligned}
s \in \eta(W) &\iff s \in \eta(S_3) \wedge s \in \eta(S_4) \\
&\iff s \in [\![\nu S_3 \,.\, \langle\{a\}\rangle\,(S_3 \wedge S_0 \wedge S_1)]\!]_{M,\eta} \\
&\qquad \wedge s \in [\![\nu S_4 \,.\, \langle\{b\}\rangle\,(S_4 \wedge S_3)]\!]_{M,\eta} \\
&\iff s \in \bigcup\{S \subseteq M_Q \mid S \subseteq [\![\langle\{a\}\rangle\,(S_3 \wedge S_0 \wedge S_1)]\!]_{M,\eta(S_3)=S}\} \\
&\qquad \wedge s \in \bigcup\{S \subseteq M_Q \mid S \subseteq [\![\langle\{b\}\rangle\,(S_4 \wedge S_3)]\!]_{M,\eta(S_4)=S}\} \\
&\iff s \in \text{Largest set such that } \left\{s' \mid \exists_{r\in M_Q}\left(s' \xrightarrow[M_\delta]{a} r \wedge r \in \eta(S_0) \cap \eta(S_1) \cap \eta(S_3)\right)\right\} \\
&\qquad \wedge s \in \text{Largest set such that } \left\{s' \mid \exists_{r\in M_Q}\left(s' \xrightarrow[M_\delta]{b} r \wedge r \in \eta(S_3) \cap \eta(S_4)\right)\right\} \\
&\iff s \in \text{Largest set such that } \Big\{s' \mid \exists_{r_1,r_2\in M_Q}\Big(s' \xrightarrow[M_\delta]{a} r_1 \wedge s' \xrightarrow[M_\delta]{b} r_2 \\
&\qquad\qquad \wedge r_1 \in \eta(S_0) \cap \eta(S_1) \cap \eta(S_3) \wedge r_2 \in \eta(S_3) \cap \eta(S_4)\Big)\Big\}
\end{aligned}
$$

To further decode the meaning of this expression, the formulae corresponding to S_0 and S_1 must be considered. First, consider that the formula for S_0 is given by

$$\langle\{a\}\rangle\,(S_1 \wedge S_2) \wedge \langle\{b\}\rangle S_3$$

Since the largest solution of S_0 is preferred, one can consider, instead, that $\eta(S_0)$ is equal to

$$\nu S_0\,.\,\langle\{a\}\rangle\,(S_1 \wedge S_2) \wedge \langle\{b\}\rangle S_3$$

As S_0 is not in this formula, the ν operator can be ignored.

Hence, for all $s' \in \eta(S_0)$, there exists an a-transition that leads to a state that is in $\eta(S_1)$ and $\eta(S_2)$, and a b-transition to $\eta(S_3)$.

Next, $\eta(S_1)$ is computed. Applying the ν operator on the formula for S_1 yields the formula to be considered for obtaining its value

$$\nu S_1\,.\,\langle\{a,\,b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\,(S_0 \wedge S_4)$$

From this, one can conclude that $\eta(S_1)$ is the largest state X such that for any $s' \in X$

1. There exists a transition on which a and b holds to a state in $\eta(S_3)$.
2. There exists a transition on which only a holds, which leads to a state in $\eta(S_1)$.
3. There exists a transition on which only b holds, which leads to a state that is in both X as well as in $\eta(S_4)$.

Again, to understand this formula, one must consider S_4. Following a similar process as before yields the following formula:

$$\nu S_4\,.\,\langle\{b\}\rangle\,(S_4 \wedge S_3)$$

Thus, $\eta(S_4)$ contains those states which have a b-transition to a state that is in both $\eta(S_3)$ and $\eta(S_4)$.

As mentioned previously, an analysis of this kind can help a requirements engineer in developing a requirement satisfied by a transformed model.

In the case of this example, one can note that in the chain of logic for the sufficient condition, a transition on which a holds was present in the expansion of the formula for every variable. Furthermore, from the starting state, a b-transition that leads to a state in $\eta(S_3) \cap \eta(S_4)$ exists. Thus, the following property can be derived

$$\psi' = GFa \wedge Xb$$

Now, consider a model M that is presented by a system designer to a requirements engineer which also satisfies the given system of equations (or the property ψ_s). The property ψ' is then a specification for M, as it is derivable from ψ_s. Since M can simulate M_{trans}, this specification must hold for M_{trans} as well.

6.2 Second Experiment

6.2.1 Motivation

As mentioned previously, the system on which the given model transformation is applied is that of the IEEE-1394 protocol. This is a standard for *isochronous*, real-time, serial data transfer, commonly referred to by its commercial name - **FireWire**.

One can consider how the rule being analyzed can potentially cause a livelock. Note that the rule adds $e1$-transitions in $\mathcal{R}^{\kappa}$ in Figure 5.5 distinguishes it from $\mathcal{L}^{\kappa}$. These transitions form a loop from state $2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow 1$ which is closed off by a $\gamma_{0,1}$ transition from 1 to 0. Such a loop, including a $\gamma_{0,1}$ action, is also present in $\mathcal{L}^{\kappa}$ (between states 0 and 1). However, in that loop, the only non-$rPDind$ action has a $\gamma_{0,1}$ label, which can potentially also correspond to a communication action in a real source system M on which $(\mathcal{L}, \mathcal{R})$ is applied. In such a scenario, under weak fairness assumption, the model M would not succumb to livelock.

Since the action $e1$ is non-parameterized, a loop with only $e1$ on it can lead to non-progress, under the weak fairness assumption. Thus, the transformation being analyzed can cause an additional source of livelock. For a system like IEEE-1394 whose main aim is to provide a stable serial communication channel at the link layer, a livelock is directly antagonistic to its goal. Therefore, in this context, if one analyzes the formula that was obtained as a final result, one can reach conclusions about the structural peculiarities of models that can arise after the rule is applied on a system on which a livelock property originally holds.

6.2.2 Inferences

Contrary to the previous experiment, in this one, the focus for drawing inferences is shifted to the necessary condition. Suppose that a requirements engineer is presented with a system M that can be simulated by M_{trans}. Then, applying the methodology presented in this work, they can obtain the system of characteristic equations $E_{\rightsquigarrow}$ and the corresponding formula ψ_n that M is guaranteed to satisfy.

It is guaranteed that for all starting states s for such a model M, s is present in $\eta(U)$, where U is the variable in the first equation of $E_{\rightsquigarrow}$, when said system is evaluated on M. Thus, the following chain of logic holds

$$\begin{aligned}
s \in \eta(U) \iff\ & s \in \eta(S_3) \vee s \in \eta(S_5) \wedge s \in \eta(S_6) \\
\iff\ & s \in [\![\nu S_3\,.\,[\{rPDind\}]\,(S_3 \vee S_5) \wedge [\{e1,\, rPDind\}]S_6 \wedge [\{e1\}]\text{FALSE}]\!]_{M,\eta} \\
& \vee s \in \eta(S_5) \vee s \in \eta(S_6) \\
\iff\ & s \in \bigcup\{S \subseteq M_Q \mid S \subseteq [\![[\{rPDind\}]\,(S_3 \vee S_5) \wedge [\{e1,\, rPDind\}]S_6 \wedge [\{e1\}]\text{FALSE}]\!]_{M,\eta(S_3)=S}\} \\
& \vee s \in \eta(S_5) \vee s \in \eta(S_6) \\
\iff\ & s \in \text{Largest set such that } \Bigg\{s' \mid \forall_{r \in M_Q} \Bigg(\Bigg(s' \xrightarrow[M_\delta]{rPDind} r \implies (r \in \eta(S_3) \vee r \in \eta(S_5)) \Bigg) \\
& \wedge \Bigg(s' \xrightarrow[M_\delta]{e1,rPDind} r \implies r \in \eta(S_6) \Bigg) \wedge \nexists_{r \in M_Q} s' \xrightarrow[M_\delta]{e1} r \Bigg) \Bigg\} \vee s \in \eta(S_5) \vee s \in \eta(S_6)
\end{aligned}$$

To further decode the meaning of this expression, the formulae corresponding to S_5 and S_6 must be considered. First, consider that the formula corresponding to S_5 is given by

$$[\{rPDind\}]\,(S_3 \vee S_5) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]S_6$$

Since the largest solution of S_5 is preferred, one can consider, instead, that $\eta(S_5)$ is equal to

$$\nu S_5\,.\,[\{rPDind\}]\,(S_3 \vee S_5) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]S_6$$

Thus, it can be concluded that $\eta(S_5)$ is the largest state X such that for all $s' \in X$, the following hold

1. All transitions on which only $rPDind$ evaluates to true lead to a state either in $\eta(S_3)$ or in X.

2. There is no transition on which both $e1$ and $rPDind$ are true.

3. All transitions on which only $e1$ evaluates to TRUE lead to a state in $\eta(S_6)$.

Next, consider that the formula for S_6 is given by

$$[\{rPDind\}]\,(S_5 \vee S_4) \wedge [\{e1,\, rPDind\}]\text{FALSE} \wedge [\{e1\}]\,(S_6 \vee S_3)$$

Applying ν operator on this formula, one can conclude that $\eta(S_6)$ is the largest set X such that for every $s' \in X$

1. All $rPDind$-transitions from s' lead to states in $\eta(S_4)$ or $\eta(S_5)$.
2. There is no transition on which both $e1$ and $rPDind$ holds.
3. All $e1$-transitions from s' lead to states in X or $\eta(S_3)$

To decode S_4, $\eta(S_4)$ must be calculated. Some insight on the construction of S_4 may be obtained by considering a partial Gaussian elimination of the original equation system, eliminating variables S_0, S_1, and S_2. This process yields the following equation

$$S_4 = [rPDind]\text{FALSE} \wedge [\{e1\}]\Big([rPDind]\text{FALSE} \wedge [\{e1\}]\Big([rPDind]\text{FALSE} \wedge [\{e1\}](\langle\{e1\}\rangle S_3)\Big)\Big)$$

This implies that $\eta(S_4)$ is a set such that all $r' \in \eta(S_4)$ have the following properties:

1. There does not exist any transitions that satisfies the $rPDind$ AP from r'.
2. All $e1$ transitions out from r' lead to $e1$-paths of length at least four. These paths can be illustrated as being of the form

$$r' \xrightarrow[M_\delta]{e1} s_1 \xrightarrow[M_\delta]{e1} s_2 \xrightarrow[M_\delta]{e1} s_3 \xrightarrow[M_\delta]{e1} s_4$$

   All $e1$-transition from s_3 lead to a state in $\eta(S_5)$, and there does not exist transitions that satisfy $rPDind$ from s_1, s_2, or s_3. As there is no $e1$-transition from S_5, so it can be concluded that the length of the path is *exactly* four.

From this analysis, working backwards, one can conclude that the starting state $s \in \eta(U)$ has the following key characteristics

1. All transitions that satisfy $rPDind$, starting from s, lead to a state in $\eta(S_3)$, $\eta(S_4)$, $\eta(S_5)$, or $\eta(S_6)$.
2. All transitions that satisfy only $e1$, starting from s, leads to a state in $\eta(S_6)$, from which an $e1$-path of size four exists, along which no transition that satisfies $rPDind$ exists, except for at the end.
3. If there exists a transition that satisfies both $e1$ and $rPDind$ from some state, then s' must be in $\eta(S_6)$, from which no transition that satisfies both $e1$ and $rPDind$ exists. Further, all $e1$-transitions from $\eta(S_6)$ lead back to $\eta(S_6)$.

These conclusions can also be summarized by the following LTL formula

$$\begin{aligned}\varphi = &G\ \ ((e1 \wedge \neg rPDind)\ \longrightarrow\ F\ (\neg e1))\ \wedge \\ &G\ \Big((e1\ \wedge\ rPDind) \\ &\qquad\qquad \longrightarrow (((e1 \wedge \neg rPDind)\ U\ rPDind) \vee G(e1 \wedge \neg rPDind))\Big)\end{aligned}$$

These inferences help establish the fact that any system that can be simulated by the system M_{trans} obtained by transforming the pattern LTSs $(\mathcal{L}, \mathcal{R})$ from the original system of the IEEE-1394 standard, satisfies φ.

To our knowledge, this is the first work that can obtain conclusions such as these using a system of μ-calculus equations. Therefore, the novelty and scientific contributions of our work have been established in the context of this example.

Chapter 7

Conclusions

The goal of this work was to find a property that provides a summary of two facts. The first of these is that a source system M satisfies a given property ω. The second is that a given model transformation R is applicable on M. The properties that were obtained, in the end, provided both a necessary condition that was satisfied by any model that was simulated by the transformed system, as well as a sufficient condition that guarantees that any model which satisfies it must be able to simulate the transformed version the system model.

To achieve this goal, this report began with a detailed analysis of relevant work in several domains that were related to the present work. It begins by analyzing some research on the topic of model transformations. The purpose of analyzing those works was to contextualize the decision to use LTSs to represent models in the present work. This was done by contrasting the same with other works that utilized various other modeling paradigms, ranging from Families of Deterministic Finite Automata to Boolean Equation Systems. By doing so, the inference that LTSs offer the perfectly suitable balance of expressiveness and ease of use, is determined. Once the modeling approach used - the Labeled Transitions System, is justified, the focus is shifted towards the other component of the research question at hand - the *property* associated with a given system. It is noted that LTL is one of the most common languages used for specification. However, μ-calculus, a more expansive logic that uses fixpoints, is also discussed. Finally, with a broad overview of current research in both the area of model transformations as well as verification, the domain of property-preserving model verification is presented to the reader. This is a relatively new domain, but its richness is exposed with the breadth of works that it enabled. This ranged from works by Wijs et. al. on verifying if a model transformation *preserved* a given property, to a work by Menghi [7] on the subject of verifying models that were incomplete.

The work by Menghi served as inspiration for some ideas during the development of the methodology. In particular, their idea of separating the constraint obtained with respect to a particular replacement into two parts was also used in this work, by splitting the final result into a necessary and sufficient condition. Their work also inspired the choice to use the κ-extended pattern LTSs $\mathcal{L}^\kappa$ and $\mathcal{R}^\kappa$, instead of directly working with (possibly much larger) source and transformed models. A parallel can be easily drawn between the black-box states in their work, and the κ state of these pattern LTS. The main distinction between their work and this one is as follows. Their aim was to verify which refinements against of a given incomplete model (possibly) satisfied a given property. On the other hand, our work is mainly targeted at the cases where a given model's transformation *does not* satisfy the given property, and, in fact, aims to aid in the process of finding a suitable replacement for this property.

With these contributing ideas, the primary task of the report - achieving the stated research objective, was ready to be started. The task was divided into several stages. First, the left side of the transformation rule was augmented by the information that the system(upon which said rule could be applied) satisfied a given LTL formula. The choice of LTL as the specification language ensures that our system would work with a significant fraction of existing model-driven software systems. This augmentation (or merging) step yields several solutions due to non-determinism in the choices that are made during the process of selecting the states of the Büchi Automaton for the property that are to be matched against states of the augmented left side LTS of the transformation rule. In our testing, it was found that these solutions are mutually bisimilar. However, this hypothesis was not proven formally. The benefit of this being the case was that one of the solutions could be chosen arbitrarily for further analysis, without loss of generality. This solution was then transformed by the application of the transformation rule on it. Finally, we used an existing technique to extract a system of characteristic μ-calculus equations from this augmented right-hand side of our transformation rule. These equations could, in principle, be transformed, using Gaussian elimination, into a single μ-calculus formula. However, in practice, it was found that this resulting formula was unwieldy at best (see the Appendix - section B for verification). This system of equations, then, effectively is our novel contribution. In summary, it represents a property associated both with a model transformation and with another property that is satisfied by the source model on which the transformation is to be applied. This knowledge, in practice, can prove to be extremely valuable to a requirements engineer working on a software development team. This was demonstrated with the modified requirement that was found in the first experiment. This is because it is presented in a form that is readily adaptable to the form of a specification.

As described above, the present work sufficiently answers the original research objective. However, this also unfolds a potential for exploring the domain further. One of the major drawbacks of the present setup is that the properties that result from it are quite large. This is a major issue from the standpoint of human-readability of the specification. This was the reason why it was decided that the system of characteristic equations resulting from the third step of the methodology would be used when drawing inferences about the model transformation and the property at hand. The reason for this is the fact that the process of transitioning from a system of characteristic equations to a single characteristic formula using Gaussian elimination is inherently one that makes the resulting formula exponential in the number of equations in said system. A natural extension of this work, therefore, is to reduce the size of this characteristic formula. This may be done in one of two ways. Either the elimination process could be made more efficient, or, a more compact representation of the equations that correspond to the refined right side of the transformation rule being considered ($\mathcal{R}^{\kappa}_{\omega}$) could be found. Further, another idea could be to split $\mathcal{R}^{\kappa}_{\omega}$ by using some process opposite to the merging step, thereby yielding a modified formula Büchi Automaton $G_{\omega'}$ in addition to $\mathcal{R}^{\kappa}$. This model could then be transformed into a property that is essentially co-evolved with the evolution of the system by the transformation rule at hand. The benefit of this approach is that it would improve the clarity of the results that were obtained presently by separating the influence of the transformation rule and the property. The main challenge with this splitting process is that $\mathcal{R}^{\kappa}_{\omega}$ does not often have enough enhancements in comparison to $\mathcal{R}^{\kappa}$ to result in a $G_{\omega'}$ that would be considered as significant. Therefore, there would be a large amount of ambiguity in the structure of $G_{\omega'}$. Finally, a third source of improvement could be to augment the present work by allowing the use of μ-calculus formulae, in addition to LTL, as input. Since the present work is a proof of concept, it was felt that this was not necessary to be done here. However, the added flexibility of supporting multiple logic families would allow for a greater flexibility in terms of the software development systems that this work could be appended to.

Bibliography

[1] Hyejin Han and Ricardo G Sanfelice. Linear temporal logic for hybrid dynamical systems: Characterizations and sufficient conditions. *Nonlinear Analysis: Hybrid Systems*, 36:100865, 2020. vii, 17

[2] Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on ltl on finite traces: Insensitivity to infiniteness. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. vii, 17

[3] Anton Wijs and Luc Engelen. Incremental formal verification for model refining. In *Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation*, pages 29–34, 2012. vii, 1, 18

[4] Anton Wijs and Luc Engelen. Efficient property preservation checking of model refinements. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 565–579. Springer, 2013. vii, ix, 1, 18, 19

[5] Anton Wijs and Luc Engelen. Refiner: towards formal verification of model transformations. In *NASA Formal Methods Symposium*, volume 8430 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2014. vii, 1, 18, 19

[6] Sander de Putter and Anton Wijs. A formal verification technique for behavioural model-to-model transformations. *Formal Aspects of Computing*, 30(1):3–43, 2018. vii, 7, 18, 19

[7] Claudio Menghi, Paola Spoletini, and Carlo Ghezzi. Modeling, refining and analyzing incomplete büchi automata. *arXiv preprint arXiv:1609.00610*, 2016. vii, 20, 55

[8] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017. 1, 15

[9] Sami Beydeda, Matthias Book, Volker Gruhn, et al. *Model-driven software development*, volume 15. Springer, 2005. 1

[10] Thomas Stahl, Markus Völter, and Krzysztof Czarnecki. *Model-driven software development: technology, engineering, management.* John Wiley & Sons, Inc., 2006. 1

[11] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language.* Addison-Wesley Professional, 2004. 1

[12] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language.* Morgan Kaufmann, 2014. 1

[13] Ruth Sara Aguilar-Saven. Business process modelling: Review and framework. *International Journal of production economics*, 90(2):129–149, 2004. 1

[14] Joost-Pieter Katoen. 22 labelled transition systems. *Model-Based Testing of Reactive Systems*, 3472:615, 2005. 1

[15] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Automata theory, languages, and computation. *International Edition*, 24(2), 2006. 1

[16] Claudio Menghi, Paola Spoletini, and Carlo Ghezzi. Dealing with incompleteness in automata-based model checking. In *International Symposium on Formal Methods*, volume 9995 of *Lecture Notes in Computer Science*, pages 531–550. Springer, 2016. 1, 20

[17] S.D.P. Harker, K.D. Eason, and J.E. Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 266–272, 1993. 2

[18] Sharon McGee and Des Greer. A software requirements change source taxonomy. In *2009 Fourth International Conference on Software Engineering Advances*, pages 51–58, 2009. 2

[19] Tom Mens. Introduction and roadmap: History and challenges of software evolution. In *Software evolution*, pages 1–11. Springer, 2008. 2

[20] Neil Ernst, Alexander Borgida, Ivan J Jureta, and John Mylopoulos. An overview of requirements evolution. *Evolving Software Systems*, 4875:3–32, 2014. 2

[21] Tomáš Babiak, Mojmír Křetínskỳ, Vojtěch Řehák, and Jan Strejček. Ltl to büchi automata translation: Fast and more deterministic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012. 5

[22] Dana Angluin, Udi Boker, and Dana Fisman. Families of dfas as acceptors of ω-regular languages. *arXiv preprint arXiv:1612.08154*, 2016. 15

[23] Frédéric Lang and Radu Mateescu. Partial model checking using networks of labelled transition systems and boolean equation systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2012. 15, 16

[24] Valentin Goranko. Temporal logic and state systems, series: Texts in theoretical computer science, 2010. 15

[25] Parosh A Abdulla, Yu-Fang Chen, Lukas Holik, and Toma˘s Vojnar. Mediating for reduction (on minimizing alternating büchi automata). In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009. 15

[26] Josh Mengerink, Ramon RH Schiffelers, Alexander Serebrenik, and Mark van den Brand. Dsl/model co-evolution in industrial emf-based mdse ecosystems. In *ME@ MODELS*, pages 2–7, 2016. 16

[27] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010. 16

[28] Jordi Cabot and Martin Gogolla. Object constraint language (ocl): a definitive guide. In *International school on formal methods for the design of computer, communication and software systems*, volume 7320 of *Lecture Notes in Computer Science*, pages 58–90. Springer, 2012. 16

[29] Abbas Rasoolzadegan and Ahmad Abdollahzadeh Barforoush. Reliable yet flexible software through formal model transformation (rule definition). *Knowledge and information systems*, 40(1):79–126, 2014. 16

[30] LJP Engelen. *From Napkin sketches to reliable software*. PhD thesis, Eindhoven University of Technology, Eindhoven, Netherlands, 2012. 18

[31] Rob J Van Glabbeek and W Peter Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM (JACM)*, 43(3):555–600, 1996. 18

[32] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0—a framework for ltl and ω-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, 2016. 23

[33] Olav Bunte, Jan Friso Groote, Jeroen JA Keiren, Maurice Laveaux, Thomas Neele, Erik P de Vink, Wieger Wesselink, Anton Wijs, and Tim AC Willemse. The mcrl2 toolset for analysing concurrent systems: improvements in expressivity and usability. In *25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems conference series, TACAS 2019 held as part of the 22nd European Joint Conferences on Theory and Practice of Software, ETAPS 2019*, volume 11428 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2019. 29, 39, 45

[34] Markus Müller-Olm. Derivation of characteristic formulae. *Electronic Notes in Theoretical Computer Science*, 18:159–170, 1998. 30, 31

[35] Sebastiaan Pascal Luttik et al. *Description and formal specification of the link layer of P1394*. Citeseer, 1997. 43

Appendix A

Formula Corresponding to First Experiment

The μ-calculus formulae characterizing $\mathcal{R}^{\kappa}_{\psi}$ that were obtained in the first experiment is given as

$$\psi_n = \nu S_4.\{[a]\textsc{false} \wedge [\{b\}](S_4 \vee \nu S_3.\{[b]\textsc{false} \wedge [\{a\}](S_3 \vee \nu S_0.\Bigg\{[\{a,\, b\}]\textsc{false} \wedge [\{b\}]\nu S_3.\Bigg([b]\textsc{false} \wedge$$

$$[\{a\}]\Big\{S_3 \vee S_0 \vee \nu S_1.\Big([\{a,\, b\}]\nu S_3.\Big\{[b]\textsc{false} \wedge [\{a\}]\Big(S_3 \vee S_0 \vee S_1\Big)\Big\} \wedge [\{b\}]\Big\{S_0 \vee S_4\Big\} \wedge [\{a\}]S_1\Big)\Big\}\Bigg) \wedge$$

$$[\{a\}]\Bigg(\nu S_1.\Big\{[\{a,\, b\}]\nu S_3.\Big([b]\textsc{false} \wedge [\{a\}]\Big\{S_3 \vee S_0 \vee S_1\Big\}\Big) \wedge [\{b\}]\Big(S_0 \vee S_4\Big) \wedge [\{a\}]S_1\Big\} \vee \Big\{[\{a,\, b\}]\textsc{false} \wedge$$

$$[\{b\}]\nu S_1.\Big([\{a,\, b\}]\nu S_3.\Big\{[b]\textsc{false} \wedge [\{a\}]\Big(S_3 \vee S_0 \vee S_1\Big)\Big\} \wedge [\{b\}]\Big\{S_0 \vee S_4\Big\} \wedge [\{a\}]S_1\Big) \wedge [\{a\}]S_0\Big\}\Big)\Bigg\} \vee$$

$$\nu S_1.\Bigg\{[\{a,\, b\}]\nu S_3.\Bigg([b]\textsc{false} \wedge [\{a\}]\Big\{S_3 \vee \nu S_0.\Big([\{a,\, b\}]\textsc{false} \wedge [\{b\}]\nu S_3.\Big\{[b]\textsc{false} \wedge [\{a\}]\Big(S_3 \vee S_0 \vee$$

$$\nu S_1.\{[\{a,\, b\}]\nu S_3.([b]\textsc{false} \wedge [\{a\}]\{S_3 \vee S_0 \vee S_1\}) \wedge [\{b\}](S_0 \vee S_4) \wedge [\{a\}]S_1\}\Big)\Big\} \wedge$$

$$[\{a\}]\Big\{\nu S_1.\Big([\{a,\, b\}]\nu S_3.\{[b]\textsc{false} \wedge [\{a\}](S_3 \vee S_0 \vee S_1)\} \wedge [\{b\}]\{S_0 \vee S_4\} \wedge [\{a\}]S_1\Big) \vee \Big([\{a,\, b\}]\textsc{false} \wedge$$

$$[\{b\}]\nu S_1.\{[\{a,\, b\}]\nu S_3.([b]\textsc{false} \wedge [\{a\}]\{S_3 \vee S_0 \vee S_1\}) \wedge [\{b\}](S_0 \vee S_4) \wedge [\{a\}]S_1\} \wedge [\{a\}]S_0\Big)\Big\}\Big) \vee S_1\Big\}\Bigg) \wedge$$

$$[\{b\}]\Bigg(\nu S_0.\Big\{[\{a,\, b\}]\textsc{false} \wedge [\{b\}]\nu S_3.\Big([b]\textsc{false} \wedge [\{a\}]\Big\{S_3 \vee S_0 \vee \nu S_1.\Big([\{a,\, b\}]\nu S_3.\{[b]\textsc{false} \wedge [\{a\}](S_3 \vee S_0 \vee$$

$$S_1)\} \wedge [\{b\}]\{S_0 \vee S_4\} \wedge [\{a\}]S_1\Big)\Big\}\Big) \wedge [\{a\}]\Big(\nu S_1.\Big\{[\{a,\, b\}]\nu S_3.\Big([b]\textsc{false} \wedge [\{a\}]\{S_3 \vee S_0 \vee S_1\}\Big) \wedge [\{b\}]\Big(S_0 \vee$$

$$S_4\Big) \wedge [\{a\}]S_1\Big\} \vee \Big\{[\{a,\, b\}]\textsc{false} \wedge [\{b\}]\nu S_1.\Big([\{a,\, b\}]\nu S_3.\{[b]\textsc{false} \wedge [\{a\}](S_3 \vee S_0 \vee S_1)\} \wedge [\{b\}]\{S_0 \vee S_4\} \wedge$$

$$[\{a\}]S_1\Big) \wedge [\{a\}]S_0\Big\}\Big)\Big\} \vee S_4\Bigg) \wedge [\{a\}]S_1\Bigg\}))\} \vee \nu S_8.\{[b]\textsc{false} \wedge [\{a\}](S_8 \vee \nu S_5.\{[\{a,\, b\}]\textsc{false} \wedge [\{b\}]S_8 \wedge$$

$$[\{a\}](\nu S_6.\Bigg\{[\{a,\, b\}]S_8 \wedge [\{b\}]\Bigg(S_5 \vee \nu S_9.\Big\{[a]\textsc{false} \wedge [\{b\}]\Big(S_9 \vee S_8\Big)\Big\}\Bigg) \wedge [\{a\}]S_6\Bigg\} \vee \Bigg\{[\{a,\, b\}]\textsc{false} \wedge$$

$$[\{b\}]\nu S_6.\Bigg([\{a,\, b\}]S_8 \wedge [\{b\}]\Big\{S_5 \vee \nu S_9.\Big([a]\textsc{false} \wedge [\{b\}]\Big\{S_9 \vee S_8\Big\}\Big)\Big\} \wedge [\{a\}]S_6\Bigg) \wedge [\{a\}]S_5\Bigg\}))\} \vee$$

$$\nu S_6.\{[\{a,\, b\}]S_8 \wedge [\{b\}](\nu S_5.\Bigg\{[\{a,\, b\}]\textsc{false} \wedge [\{b\}]S_8 \wedge [\{a\}]\Bigg(\nu S_6.\Big\{[\{a,\, b\}]S_8 \wedge [\{b\}]\Big(S_5 \vee \nu S_9.\Big\{[a]\textsc{false} \wedge$$

$$[\{b\}]\Big(S_9 \vee S_8\Big)\Big\}\Big) \wedge [\{a\}]S_6\Big\} \vee \Big\{[\{a,\, b\}]\textsc{false} \wedge [\{b\}]\nu S_6.\Big([\{a,\, b\}]S_8 \wedge [\{b\}]\Big\{S_5 \vee \nu S_9.\Big([a]\textsc{false} \wedge$$

$$[\{b\}]\{S_9 \vee S_8\}\Big)\Big\} \wedge [\{a\}]S_6\Big) \wedge [\{a\}]S_5\Big\}\Bigg)\Bigg\} \vee \nu S_9.\Bigg\{[a]\textsc{false} \wedge [\{b\}]\Bigg(S_9 \vee S_8\Bigg)\Bigg\}) \wedge [\{a\}]S_6\})\}$$

$$\psi_s = \nu S_3.\langle\{a\}\rangle(S_3 \wedge \nu S_0.\Bigg\{\langle\{a\}\rangle\Bigg(\nu S_1.\Big\{\langle\{a,\, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\Big(S_0 \wedge \nu S_4.\langle\{b\}\rangle\Big\{S_4 \wedge S_3\Big\}\Big)\Big\} \wedge$$

$$\Big\{\langle\{a\}\rangle S_0 \wedge \langle\{b\}\rangle\nu S_1.\Big(\langle\{a,\, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\Big\{S_0 \wedge \nu S_4.\langle\{b\}\rangle\Big(S_4 \wedge S_3\Big)\Big\}\Big)\Big\}\Bigg) \wedge \langle\{b\}\rangle S_3\Bigg\} \wedge$$

$$\nu S_1.\Bigg\{\langle\{a,\, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\Bigg(\nu S_0.\Big\{\langle\{a\}\rangle\Big(\nu S_1.\Big\{\langle\{a,\, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\Big(S_0 \wedge \nu S_4.\langle\{b\}\rangle\{S_4 \wedge$$

$$S_3\}\bigr)\Bigr\} \wedge \Bigl\{\langle\{a\}\rangle S_0 \wedge \langle\{b\}\rangle\nu S_1.\bigl(\langle\{a, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\{S_0 \wedge \nu S_4.\langle\{b\}\rangle(S_4 \wedge S_3)\}\bigr)\bigr\}\Bigr) \wedge \langle\{b\}\rangle S_3\Bigr\} \wedge$$

$$\nu S_4.\langle\{b\}\rangle\Bigl\{S_4 \wedge S_3\Bigr\}\Biggr)\Biggr\}) \wedge \nu S_4.\langle\{b\}\rangle(S_4 \wedge \nu S_3.\langle\{a\}\rangle\Biggl\{S_3 \wedge \nu S_0.\Biggl(\langle\{a\}\rangle\Bigl\{\nu S_1.\Bigl(\langle\{a, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge$$

$$\langle\{b\}\rangle\Bigl\{S_0 \wedge \nu S_4.\langle\{b\}\rangle\Bigl(S_4 \wedge S_3\Bigr)\Bigr\}\Bigr) \wedge \Biggl(\langle\{a\}\rangle S_0 \wedge \langle\{b\}\rangle\nu S_1.\Bigl\{\langle\{a, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\Bigl(S_0 \wedge$$

$$\nu S_4.\langle\{b\}\rangle\{S_4 \wedge S_3\}\Bigr)\Bigr\}\Biggr)\Biggr\} \wedge \langle\{b\}\rangle S_3\Biggr) \wedge \nu S_1.\Biggl(\langle\{a, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\Bigl\{\nu S_0.\Bigl(\langle\{a\}\rangle\Bigl\{\nu S_1.\Bigl(\langle\{a, b\}\rangle S_3 \wedge$$

$$\langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle\{S_0 \wedge \nu S_4.\langle\{b\}\rangle(S_4 \wedge S_3)\}\Bigr) \wedge \Bigl(\langle\{a\}\rangle S_0 \wedge \langle\{b\}\rangle\nu S_1.\{\langle\{a, b\}\rangle S_3 \wedge \langle\{a\}\rangle S_1 \wedge \langle\{b\}\rangle(S_0 \wedge$$

$$\nu S_4.\langle\{b\}\rangle\{S_4 \wedge S_3\})\}\Bigr)\Bigr\} \wedge \langle\{b\}\rangle S_3\Bigr) \wedge \nu S_4.\langle\{b\}\rangle\Bigl(S_4 \wedge S_3\Bigr)\Bigr\}\Biggr)\Biggr\})$$

Appendix B

Formula Corresponding to Second Experiment

The μ-calculus formulae below characterizes the LTS $\mathcal{R}_{\omega}^{\kappa}$ that was obtained in the second experiment.

$$\begin{aligned}
&\psi_n = \nu S_3.\Big\{[\{rPDind,\ e1\}]S_3 \wedge [\{e1\}]\nu S_5.\Big([\{rPDind,\ e1\}]\nu S_3.\Big\{[\{rPDind,\ e1\}]S_3 \wedge [\{e1\}]S_5 \wedge \\
&[\{rPDind\}]\textsc{false}\Big\} \wedge [\{e1\}]\textsc{false} \wedge [\{rPDind\}]\Big\{\nu S_3.\Big([\{rPDind,\ e1\}]S_3 \wedge [\{e1\}]S_5 \wedge [\{rPDind\}]\textsc{false}\Big) \vee \\
&S_5 \vee \Big([rPDind]\textsc{false} \wedge [\{e1\}]\Big\{[rPDind]\textsc{false} \wedge [\{e1\}]\Big([rPDind]\textsc{false} \wedge [\{e1\}]\{[rPDind]\textsc{false} \wedge \\
&[\{e1\}]\nu S_3.([\{rPDind,\ e1\}]S_3 \wedge [\{e1\}]S_5 \wedge [\{rPDind\}]\textsc{false})\}\Big)\Big\}\Big)\Big\}\Big) \wedge [\{rPDind\}]\textsc{false}\Big\} \vee \\
&\nu S_5.\Big\{[\{rPDind,\ e1\}]\nu S_3.\Big([\{rPDind,\ e1\}]S_3 \wedge [\{e1\}]S_5 \wedge [\{rPDind\}]\textsc{false}\Big) \wedge [\{e1\}]\textsc{false} \wedge \\
&[\{rPDind\}]\Big(\nu S_3.\Big\{[\{rPDind,\ e1\}]S_3 \wedge [\{e1\}]S_5 \wedge [\{rPDind\}]\textsc{false}\Big\} \vee S_5 \vee \Big\{[rPDind]\textsc{false} \wedge \\
&[\{e1\}]\Big([rPDind]\textsc{false} \wedge [\{e1\}]\Big\{[rPDind]\textsc{false} \wedge [\{e1\}]\Big([rPDind]\textsc{false} \wedge [\{e1\}]\nu S_3.\{[\{rPDind,\ e1\}]S_3 \wedge \\
&[\{e1\}]S_5 \wedge [\{rPDind\}]\textsc{false}\}\Big)\Big\}\Big)\Big\}\Big)\Big\} \vee S_6
\end{aligned}$$

$$\begin{aligned}
&\psi_s = \nu S_3.\Big\{\langle\{rPDind,\ e1\}\rangle S_3 \wedge \langle\{e1\}\rangle\nu S_5.\Big(\langle\{rPDind,\ e1\}\rangle S_3 \wedge \langle\{rPDind\}\rangle\Big\{S_3 \wedge S_5 \wedge \\
&\langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle S_3\Big\}\Big)\Big\} \wedge \nu S_5.\Big\{\langle\{rPDind,\ e1\}\rangle\nu S_3.\Big(\langle\{rPDind,\ e1\}\rangle S_3 \wedge \\
&\langle\{e1\}\rangle\nu S_5.\Big\{\langle\{rPDind,\ e1\}\rangle S_3 \wedge \langle\{rPDind\}\rangle\Big(S_3 \wedge S_5 \wedge \langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle S_3\Big)\Big\}\Big) \wedge \\
&\langle\{rPDind\}\rangle\Big(\nu S_3.\Big\{\langle\{rPDind,\ e1\}\rangle S_3 \wedge \langle\{e1\}\rangle\nu S_5.\Big(\langle\{rPDind,\ e1\}\rangle S_3 \wedge \langle\{rPDind\}\rangle\Big\{S_3 \wedge S_5 \wedge \\
&\langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle S_3\Big\}\Big)\Big\} \wedge S_5 \wedge \langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle\nu S_3.\Big\{\langle\{rPDind,\ e1\}\rangle S_3 \wedge \\
&\langle\{e1\}\rangle\nu S_5.\Big(\langle\{rPDind,\ e1\}\rangle S_3 \wedge \langle\{rPDind\}\rangle\Big\{S_3 \wedge S_5 \wedge \langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle\langle\{e1\}\rangle S_3\Big\}\Big)\Big\}\Big)\Big\} \wedge S_6
\end{aligned}$$