EVALUATING REASONING HEURISTICS FOR A HYBRID THEOREM PROVING PLATFORM

by

JACOBUS GIDEON ACKERMANN

submitted in fulfilment of the requirements for the

degree of

MASTER OF SCIENCE

in the subject of

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF. J.A. VAN DER POLL

JUNE 2018

DECLARATION

Name: Jacobus Gideon Ackermann

Student number: 3 2 2 4 7 0 6 0

Degree: Master of Science

Exact wording of the title of the dissertation or thesis as appearing on the copies submitted for examination:

EVALUATING REASONING HEURISTICS FOR A HYBRID THEOREM PROVING PLATFORM

I declare that the above dissertation/thesis is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

SIGNATURE

12 June 2018

DATE

Abstract

The formalisation of first-order logic and axiomatic set theory in the first half of the 20th century — along with the advent of the digital computer — paved the way for the development of automated theorem proving. In the 1950s, the automation of proof developed from proving elementary geometric problems and finding direct proofs for problems in *Principia Mathematica* by means of simple, human-oriented rules of inference. A major advance in the field of automated theorem proving occurred in 1965, with the formulation of the resolution inference mechanism. Today, powerful Satisfiability Modulo Theories (SMT) provers combine SAT solvers with sophisticated knowledge from various problem domains to prove increasingly complex theorems.

The combinatorial explosion of the search space is viewed as one of the major challenges to progress in the field of automated theorem proving. Pioneers from the 1950s and 1960s have already identified the need for heuristics to guide the proof search effort.

Despite theoretical advances in automated reasoning and technological advances in computing, the size of the search space remains problematic when increasingly complex proofs are attempted. Today, heuristics are still useful and necessary to discharge complex proof obligations.

In 2000, a number of heuristics was developed to aid the resolution-based prover OTTER in finding proofs for set-theoretic problems. The applicability of these heuristics to next-generation theorem provers were evaluated in 2009. The provers Vampire and Gandalf required respectively 90% and 80% of the applicable OTTER heuristics.

This dissertation investigates the applicability of the OTTER heuristics to theorem proving in the hybrid theorem proving environment Rodin — a system modelling tool suite for the Event-B formal method. We show that only 2 of the 10 applicable OTTER heuristics were useful when discharging proof obligations in Rodin. Even though we argue that the OTTER heuristics were largely ineffective when applied to Rodin proofs, heuristics were still needed when proof obligations could not be discharged automatically. Therefore, we propose a number of our own heuristics targeted at theorem proving in the Rodin tool suite.

Keywords: first-order logic, automated reasoning, automated theorem proving, Rodin/Event-B, SMT, heuristics, OTTER, resolution, Zermelo-Fraenkel set theory, CVC3, CVC4, veriT, Z3

Opsomming

Die formalisering van eerste-orde-logika en aksiomatiese versamelingsteorie in die eerste helfte van die 20ste eeu, tesame met die koms van die digitale rekenaar, het die weg vir die ontwikkeling van geoutomatiseerde bewysvoering gebaan. Die outomatisering van bewysvoering het in die 1950's ontwikkel vanuit die bewys van elementêre meetkundige probleme en die opspoor van direkte bewyse vir probleme in *Principia Mathematica* deur middel van eenvoudige, mensgerigte inferensiereëls. Vooruitgang is in 1965 op die gebied van geoutomatiseerde bewysvoering gemaak toe die resolusie-inferensie-meganisme geformuleer is. Deesdae kombineer kragtige *Satisfiability Modulo Theories* (SMT) bewysvoerders SAT-oplossers met gesofistikeerde kennis vanuit verskeie probleemdomeine om steeds meer komplekse stellings te bewys.

Die kombinatoriese ontploffing van die soekruimte kan beskou word as een van die grootste uitdagings vir verdere vooruitgang in die veld van geoutomatiseerde bewysvoering. Baanbrekers uit die 1950's en 1960's het reeds bepaal dat daar 'n behoefte is aan heuristieke om die soektog na bewyse te rig.

Ten spyte van die teoretiese vooruitgang in outomatiese bewysvoering en die tegnologiese vooruitgang in die rekenaarbedryf, is die grootte van die soekruimte steeds problematies wanneer toenemend komplekse bewyse aangepak word. Teenswoordig is heuristieke steeds nuttig en noodsaaklik om komplekse bewysverpligtinge uit te voer.

In 2000 is 'n aantal heuristieke ontwikkel om die resolusie-gebaseerde bewysvoerder OTTER te help om bewyse vir versamelingsteoretiese probleme te vind. Die toepaslikheid van hierdie heuristieke vir die volgende generasie bewysvoerders is in 2009 geëvalueer. Die bewysvoerders *Vampire* en *Gandalf* het onderskeidelik 90% en 80% van die toepaslike OTTER-heuristieke nodig gehad.

Hierdie verhandeling ondersoek die toepaslikheid van die OTTER-heuristieke op bewysvoering in die hibriede bewysvoeringsomgewing Rodin — 'n stelselmodelleringsuite vir die formele *Event-B*-metode. Ons toon dat slegs 2 van die 10 toepaslike OTTER-heuristieke van nut was vir die uitvoering van bewysverpligtinge in Rodin. Ons voer aan dat die OTTER-heuristieke grotendeels ondoeltreffend was toe dit op Rodin-bewyse toegepas is. Desnieteenstaande is heuristieke steeds nodig as bewysverpligtinge nie outomaties uitgevoer kon word nie. Daarom stel ons 'n aantal van ons eie heuristieke voor wat in die Rodin-suite aangewend kan word.

Sleutelwoorde: eerste-orde-logika, outomatiese redenering, outomatiese bewysvoering, Rodin/Event-B, SMT, heuristieke, OTTER, resolusie, Zermelo-Fraenkel-versamelingsteorie, CVC3, CVC4, veriT, Z3

Kafushane ngocwaningo

Ukwenziwa semthethweni kwe-*first-order logic* kanye ne-*axiomatic set theory* ngesigamu sokuqala sekhulunyaka lama-20—kanye nokufika kwekhompyutha esebenza ngobuxhakaxhaka bedijithali—kwavula indlela ebheke ekuthuthukisweni kwenqubo-kusebenza yokufakazela amathiyoremu ngekhomyutha. Ngeminyaka yawo-1950, ukuqinisekiswa kobufakazi kwasuselwa ekufakazelweni kwezinkinga zejiyomethri eziyisisekelo kanye nasekutholakaleni kobufakazi-ngqo bezinkinga eziphathelene ne-*Principia Mathematica* ngokuthi kusetshenziswe imithetho yokuqagula-sakucabangela elula, egxile kubantu. Impumelelo enkulu emkhakheni wokufakazela amathiyoremu ngekhompyutha yenzeka ngowe-1965, ngokwenziwa semthethweni kwe-*resolution inference mechanism*. Namuhla, abafakazeli abanohlonze bamathiyori abizwa nge-*Satisfiability Modulo Theories* (SMT) bahlanganisa ama-*SAT solvers* nolwazi lobungcweti oluvela kwizizinda zezinkinga ezihlukahlukene ukuze bakwazi ukufakazela amathiyoremu okungelula neze ukuwafakazela.

Ukukhula ngesivinini kobunzima nobunkimbinkimbi benkinga esizindeni esithile kubonwa njengenye yezinselelo ezinkulu okudingeka ukuthi zixazululwe ukuze kube nenqubekela phambili ekufakazelweni kwamathiyoremu ngekhompyutha. Amavulandlela eminyaka yawo-1950 nawo-1960 asesihlonzile kakade isidingo sokuthi amahuristikhi (*heuristics*) kube yiwona ahola umzamo wokuthola ubufakazi.

Nakuba ikhona impumelelo esiyenziwe kumathiyori ezokucabangela okujulile kusetshenziswa amakhompyutha kanye nempumelelo yobuchwepheshe bamakhompyutha, usayizi wesizinda usalokhu uyinkinga uma kwenziwa imizamo yokuthola ubufakazi obuyinkimbinkimbi futhi obunobunzima obukhudlwana. Namuhla imbala, amahuristikhi asewuziso futhi ayadingeka ekufezekiseni izibopho zobufakazi obuyinkimbinkimbi.

Ngowezi-2000, kwathuthukiswa amahuristikhi amaningana impela ukuze kulekelelwe uhlelo-kusebenza olungumfakazeli osekelwe phezu kwesixazululo, olubizwa nge-OTTER, ekutholeni ubufakazi bama-*set-theoretic problems*. Ukusebenziseka kwalawa mahuristikhi kwizinhlelo-kusebenza ezingabafakazeli bamathiyoremu besimanjemanje kwahlolwa ngowezi-2009. Uhlelo-kusebenza olungumfakazeli, olubizwa nge-Vampire kanye nalolo olubizwa nge-Gandalf zadinga ama-90% kanye nama-80%, ngokulandelana kwazo, maqondana nama-*OTTER heuristics* afanelekile.

Lolu cwaningo luphenya futhi lucubungule ukusebenziseka kwama-*OTTER heuristics* ekufakazelweni kwamathiyoremu esimweni esiyinhlanganisela sokufakazela amathiyoremu esibizwa nge-Rodin—okuyi-*system mod-*

elling tool suite eqondene ne-*Event-B formal method*. Kulolu cwaningo siyabonisa ukuthi mabili kuphela kwayi-10 ama-*OTTER heuristics* aba wusizo ngenkathi kufezekiswa isibopho sobufakazi ku-*Rodin*. Nakuba sibeka umbono wokuthi esikhathini esiningi ama-*OTTER heuristics* awazange abe wusizo uma esetshenziswa kuma-*Rodin proofs*, amahuristikhi asadingeka ezimweni lapho izibopho zobufakazi zingazenzekelanga ngokwazo ngokulawulwa yizinhlelo-kusebenza zekhompyutha. Ngakho-ke, siphakamisa amahuristikhi ethu amaningana angasetshenziswa ekufakazeleni amathiyoremu ku-*Rodin tool suite*.

Amagama asemqoka: *first-order logic*, ukucabangela okujulile okwenzeka ngekhompyutha, ukufakazela amathiyoremu ngekhompyutha, *Rodin/Event-B*, *SMT*, amahuristikhi (*heuristics*), OTTER, isixazululo, *Zermelo-Fraenkel set theory*, *CVC3*, *CVC4*, *veriT*, *Z3*

Contents

List of Tables

List of Figures

List of Definitions

Chapter 1

Introduction

This is a dissertation on evaluating the usefulness and applicability of a set of reasoning heuristics that were developed for the resolution-based theorem prover OTTER. The evaluation was conducted in a hybrid theorem proving environment. Our definition of a hybrid theorem proving environment is the following:

> **Definition 1.1** (Hybrid Theorem Proving Environment)**.** A theorem proving environment is a *hybrid* environment when the theorem proving tool applies at least two different inference mechanisms while discharging proof obligations.

Our discussion focuses on heuristics for discharging set-theoretic proof obligations formulated in first-order logic.

1.1 Motivation

A number of industrially successful formal specification and modelling languages are based on first-order logic and set theory, e.g., Z (Lightfoot 1991), the B-method (Abrial 1996) and Event-B (Abrial 2010). Formal specifications and mathematical models offer the advantage of formally reasoning about system properties. However, writing down and proving these formal properties by hand is "foolish (and error prone)... for the simple reason that it is common to have thousands of such proofs" (Abrial 2010, p. xviii).

Instead, the proof obligations that arise from such specification and modelling activities can be discharged by automated theorem provers. However, automated theorem proving for first-order logic is a hard problem. The complexity is a consequence of the large — even infinite — number of distinct, derivable formulas and the number of ways in which formulas can be derived from the initial axioms. Both the number of derivable formulas and the number of ways in which they can be derived accelerate with the proof search depth (Lusk 1992).

The problem is further compounded by the hierarchical nature of set theory when attempting set-theoretic

proofs: Every set S is a member of some *larger* set, e.g., $S \in \mathbb{P}(S)$, $\mathbb{P}(S) \in \mathbb{P}(\mathbb{P}(S))$ etc. The definitions of sets and set-theoretic constructs must therefore be unfolded judiciously to avoid referencing unnecessary sets.

It is generally accepted that heuristics are needed to prove interesting and complex theorems with automated theorem provers, since proofs are often not discovered without the pruning of inference steps or guiding the search through the space of possible inference steps (Bundy 1999, p. 156). The need for heuristics was recognized as early as 1955 with the development of the Logic Theory Machine (Newell and Simon 1956; Mackenzie 1995, p. 12). In 1960, Hao Wang predicted that "strategies in the search for proofs, or what are often called heuristic methods, will also gradually become part of the subject matter of inferential analysis" (Wang 1960, p. 220).

Labuschagne and Van der Poll (1999) and Van der Poll (2000) have formulated a set of reasoning heuristics to guide and support the resolution-based automated reasoner OTTER (McCune 2003) in discharging set-theoretic proof obligations. Hereafter we refer to the Van der Poll-Labuschagne heuristics as the OTTER heuristics. Since the OTTER heuristics were specifically formulated for resolution-based theorem proving, Steyn (2009) evaluated the utility of these heuristics for the newer, state-of-the-art resolution-based theorem provers Vampire (Riazanov and Voronkov 2002) and Gandalf (Tammet 1997). Steyn was able to demonstrate that the OTTER heuristics were still applicable nearly a decade after their formulation: Vampire needed 10 and Gandalf needed 9 of the 11 OTTER heuristics that were evaluated.

1.2 Research Question

Steyn (2009) was able to demonstrate that the OTTER heuristics — developed for the resolution-based theorem prover OTTER — have wider applicability. However, Steyn evaluated the utility of these heuristics for additional *resolution-based* theorem provers. The question therefore arises whether the OTTER heuristics can be applied in a hybrid theorem proving environment.

Rodin/Event-B[1] was identified as a suitable hybrid theorem proving environment to evaluate the applicability of the OTTER heuristics because:

- The two basic Event-B mathematical theories are first-order logic and (typed) set theory; and
- The theorem proving environment provides integrated access to a number of different inference mechanisms, i.e. Rodin/Event-B is a hybrid theorem proving platform.

For example, since Rodin version 3, the following inference mechanisms were available via theorem prover extension plug-ins[2]: SMT (CVC3, CVC4, veriT and Z3), model checking (ProB), term rewriting (ML) and

[1] `http://www.event-b.org/`

[2] The SMT prover plug-in for Rodin is a joint work between Systerel and *Universidade Federal do Rio Grande do Norte* (`http://www.systerel.fr/`). The predicate prover (PP) and mono-lemma (ML) prover are part of the Atelier B plug-in developed by

resolution (PP, NewPP).

Our primary aim in this dissertation was to evaluate to what extent the OTTER heuristics were applicable to the hybrid theorem proving environment Rodin. Our research addresses the following questions:

- How are proof obligations discharged in Rodin?
- Can Rodin prove the collection of problems formulated in (Van der Poll 2000) and (Steyn 2009) automatically?
- If not, does the application of the OTTER heuristics increase the number of automatically discharged proof obligations?
- Are additional heuristics required to increase the number of automatically discharged proof obligations in Rodin?

1.3 Hypotheses

The OTTER heuristics formulated for resolution-based automated reasoning with set theory are not applicable when discharging set-theoretic proof obligations with the Rodin/Event-B hybrid theorem proving platform.

However, our preferred automated theorem proving strategy only yielded a 83.1% success rate when applied to the proof obligations that arise from the Rodin contexts in Appendix C. This leads to our secondary hypothesis: additional Rodin specific heuristics are necessary to discharge all the proof obligations that arise from the Rodin contexts in Appendix C.

1.4 Approach

Each of the heuristics in Van der Poll (2000) were explained in light of a specific set-theoretic problem. Steyn (2009) presented additional examples where the complexities of Van der Poll's set-theoretic problems were increased.

These (and numerous other) set-theoretic problems were attempted in Rodin and when proofs could not be generated automatically the OTTER heuristics were applied. When a proof obligation could not be discharged even after the application of suitable OTTER heuristics, the proof was attempted interactively in Rodin. From these interactive attempts we were able to derive additional heuristics targeted specifically at theorem proving in Rodin.

ClearSy (`http://www.clearsy.com/en/`). ProB is provided by the HHU Düsseldorf STUPS Group (`https://www3.hhu.de/stups/prob/index.php/Main_Page`) and NewPP is the only prover that is included in the Rodin installer.

1.5 Dissertation Layout

In the current chapter we have presented our motivation for the study of automated reasoners and the need for heuristics to guide automated theorem provers in their search for proofs. We have also proposed a number of research questions around the applicability of a set of previously defined heuristics for the resolution-based reasoner OTTER to the *hybrid theorem proving environment* Rodin/Event-B.

In Chapter 2 we outline the historical developments of the theoretical foundations of our research, namely *first-order logic*, *set theory* and *automated theorem proving*. We see that first-order logic was not considered the logical system of choice until the 1940s (especially after World War II). Developments in logic and deduction often occurred in parallel with developments in mathematics and the discovery of contradictions in set theory (one of the foundational branches of mathematics) provided a major impetus for establishing the boundaries of formal systems. An important consequence was the development of axiomatic set theory which lends itself naturally to *mechanised* theorem proving in deductive systems. With the advent of digital computing, the confluence of these developments has *inter alia* resulted in the discovery and development of automated theorem proving.

Chapter 3 expands on Chapter 2 to provide a technical introduction to deductive systems and inference mechanisms. The Gentzen system $\mathcal{G}$ is introduced to illustrate the process of deriving valid conclusions in a deductive system. Automated theorem provers embed deductive systems as part of their inference mechanism and the remainder of the chapter introduces the inference mechanisms implemented by the theorem provers that were used in our experimental work.

In Chapter 4 we present the Zermelo-Fraenkel axiomatisation of set theory in first-order logic. The definitions of set-theoretic constructs and operators in first-order logic are necessary to formulate input for theorem provers that cannot accept the highly evolved notation of set theory. The chapter is concluded with an example that illustrates the approach we have taken to rewrite formulas in set-theoretic notation to equivalent formulas in first-order logic.

The resolution-based theorem provers OTTER, Vampire and Gandalf are introduced in Chapter 5. Van der Poll (2000) used OTTER to develop and test the OTTER heuristics, and Steyn (2009) investigated the applicability of these heuristics to the next generation theorem provers Vampire and Gandalf — their findings are included in Chapter 7. The remainder of the chapter presents the Rodin theorem provers NewPP, PP, ML, ProB, and the SMT theorem provers CVC3, CVC4, veriT and Z3.

The hybrid theorem proving environment used in our research, Rodin/Event-B, is introduced in Chapter 6. A short overview of the historical development of the Rodin tool suite is presented, followed by a discussion of the Event-B syntax. The remainder of the chapter introduces the Rodin/Event-B inference mechanism, tactic profiles and the integration of automated reasoners.

In Chapter 7 we evaluate the utility of the OTTER heuristics for Rodin. For each heuristic the sample problems in (Van der Poll 2000) and (Steyn 2009) are discussed to illustrate the utility of the heuristic for OTTER, Vampire and Gandalf. The same problems were attempted in Rodin and the relevant heuristic applied where a proof obligation could not be discharged automatically. We concluded that the OTTER heuristics were largely ineffective when applied to Rodin proofs.

Chapter 8 addresses the need for additional Rodin specific heuristics to discharge all the proof obligations that arise from the Rodin contexts in Appendix C. A total of eight heuristics are presented to aid with discharging proof obligations in Rodin.

In the first part of Chapter 9 we present Morgan's Z specification for a telephone system (Morgan 1993). We show the implementation of the system properties in Event-B and attempt a number of proof obligations that arise from the specification. In the following section, we discuss the implementation of the group axioms in Rodin and show how the Rodin heuristics can be applied interactively when a proof obligation is not discharged automatically. Finally we show how the Rodin provers can be used to evaluate intersection points for polynomials with integer coefficients. A number of proof obligations could only be discharged after applying several Rodin heuristics.

Chapter 10 summarises the conclusions to be drawn from the research reported on in this dissertation. We close with some thoughts on directions for future research.

Chapter 2

Literature Review

We have presented our motivation for studying automated reasoning heuristics in Chapter 1. However, we have not yet presented a historical context to put our choice of theoretical frameworks and technologies in perspective.

In this chapter we present an overview of the development of mathematical logic, foundational mathematics and proof theory. We narrow the discussion to first-order logic, axiomatic set theory and automated theorem proving, since these fields constitute the theoretical framework for our experimental results.

In Section 2.1 we discuss the historical development of first-order logic. First-order logic is considered the standard for the formalisation of axiomatic mathematics. For example, Peano arithmetic and Zermelo-Fraenkel set theory are axiomatisations of number theory and set theory, respectively, into first-order logic.

Section 2.2 presents an overview of the development of axiomatic set theory. The axiomatic treatment of set theory was not isolated from developments in logic and pioneers involved in the development of first-order logic (such as Frege and Russell) had a part to play in the development of axiomatic set theory too.

The origins of the field of automated theorem proving are discussed in Section 2.3. During the early years, significant challenges (e.g., the combinatorial explosion of the search space and the limited proof search abilities of early provers) threatened to derail interest in automated theorem proving. However, the field was revitalised with the formulation and successful implementation of the resolution mechanism in 1965. Today the field of automated theorem proving has grown into a sophisticated and actively expanding branch of proof theory.

Our discussion spans several centuries and a large number of historically significant figures are mentioned. We have therefore summarized Sections 2.1 to 2.3 in two timelines. Figure 2.1 presents a timeline of the historical figures who made important contributions to the fields of mathematical logic, axiomatic set theory or automated theorem proving. Similarly, Figure 2.2 contains a timeline of the significant historical events that shaped the landscape forming the background to the remainder of this work.

The summary in Section 2.4 concludes the chapter.

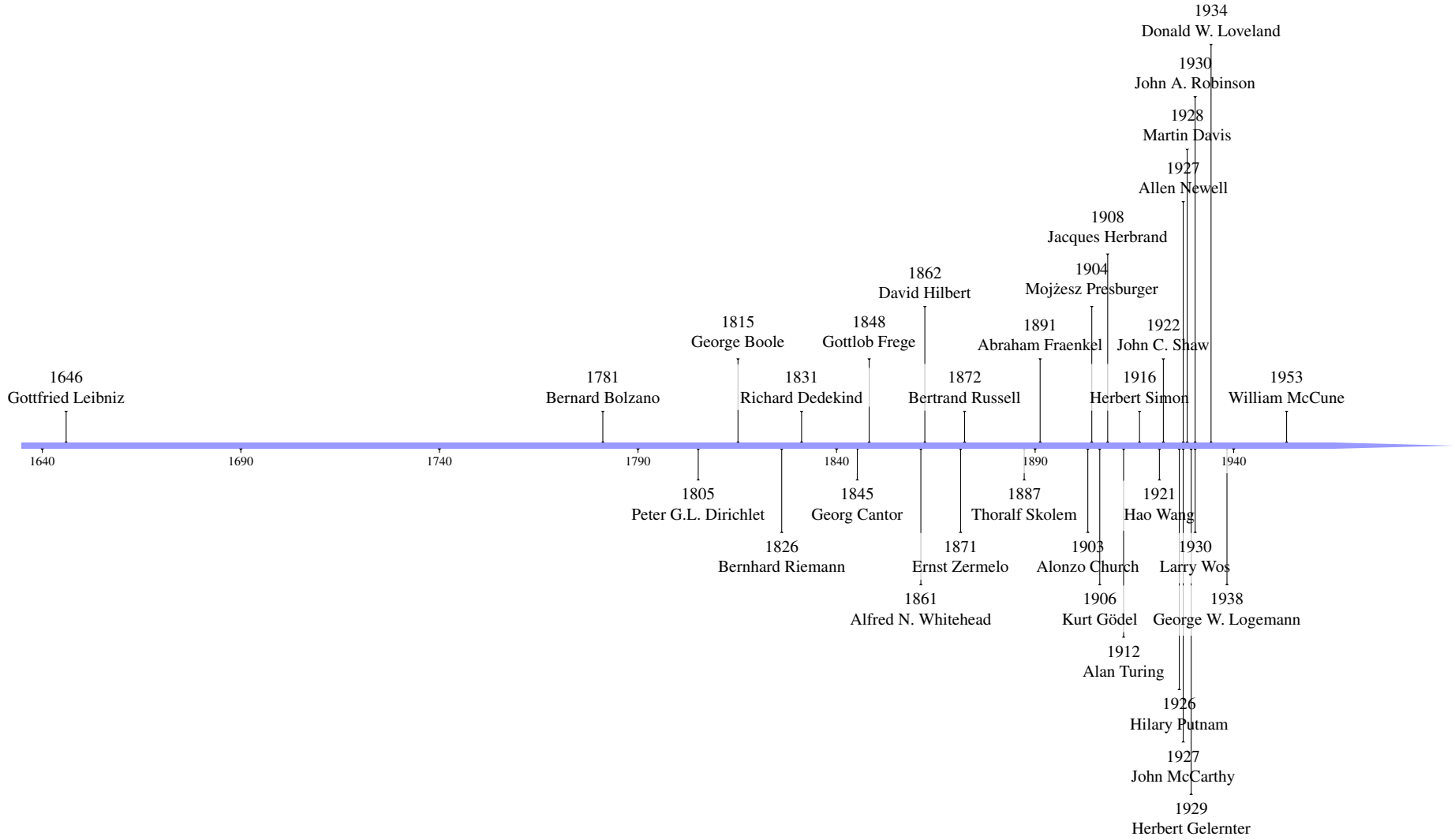


Figure 2.1: Timeline of Historical People (synthesised by the researcher)

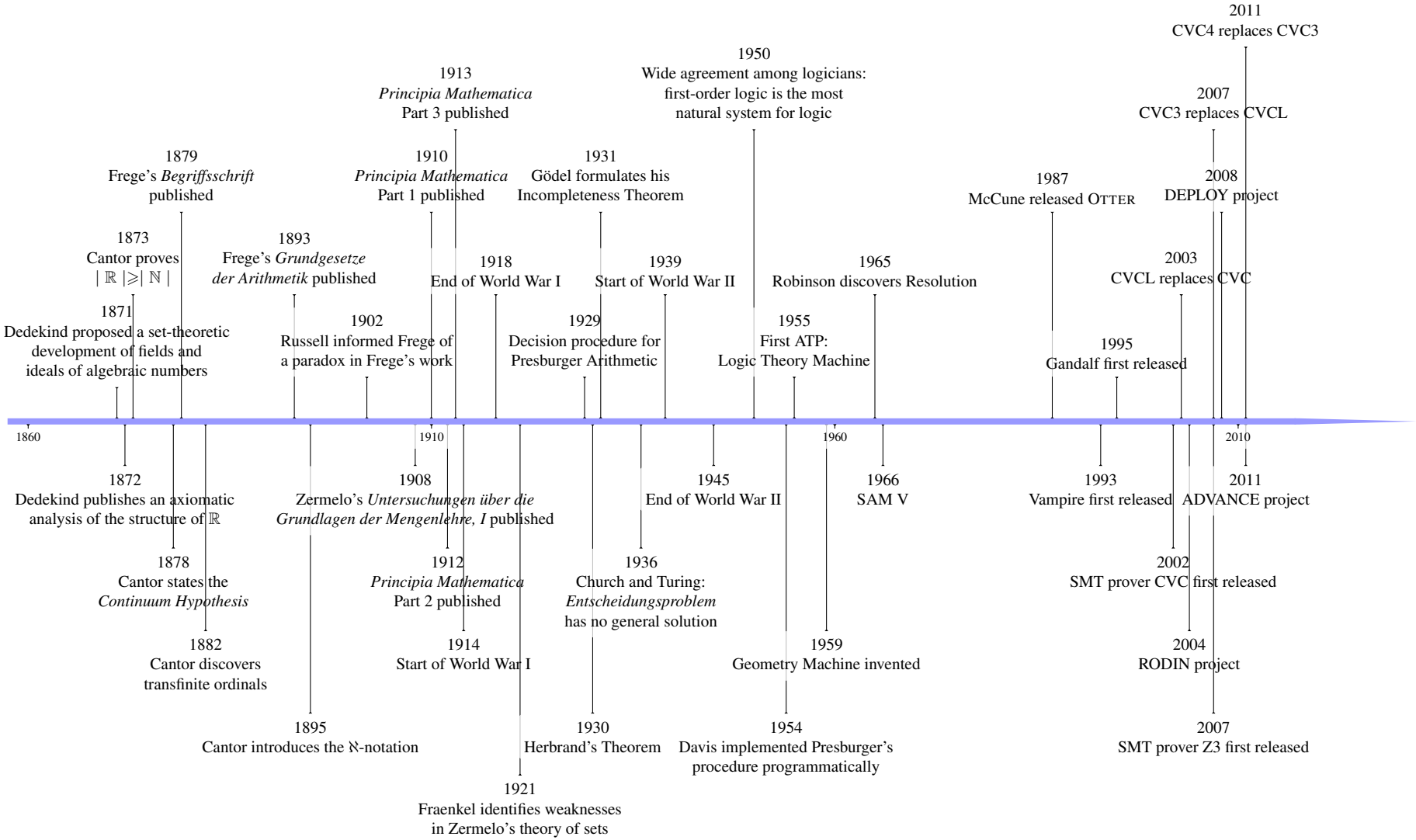


Figure 2.2: Timeline of Historical Developments (synthesised by the researcher)

2.1 Developments in Mathematical Logic

Our discussion of developments in mathematical logic is presented in two sections. In Section 2.1.1 we consider the origin of classical logic or first-order logic. Next, we consider some of the fundamental properties of deductive reasoning and deductive systems for first-order logic in Section 2.1.2.

2.1.1 The Origin of First-Order Logic

The study of logic dates back to the ancient Greek educational system which emphasised philosophy and rhetoric. Rhetoric (public debate) included the study of logic that placed all sides on an equal footing in terms of the rules of deduction (Ben-Ari 2004). Logic formalises the deductive process, i.e. the process of deriving true statements (conclusions) from statements assumed to be true (premises). An example of a well-known rule of logic is *syllogism* (Ben-Ari 2004, p. 1):

> **Premise** All men are mortal
> **Premise** X is a man
> **Conclusion** X is mortal.

Another rule known since antiquity is *modus ponens* (Mackenzie 1995, p. 9):

> **Premise** P
> **Premise** $P \Rightarrow Q$
> **Conclusion** Q.

But for centuries logic remained mainly a philosophical tool, rather than a tool for mathematics or science. In fact, for nearly two millennia works such as Euclid's *Elements* provided paradigms of valid proof without any attempt to formalise the notion of proof *per se* (Mackenzie 1995, p. 8).

One of the early attempts at defining a calculus for logic and proof was by Gottfried Leibniz (1646-1716). Leibniz was interested in reducing proof to the manipulation of symbols alone, but his approach was not widely adopted. More than a century later, George Boole (1815-1864) developed an algebraic representation of propositional logic (more specifically, the logical connectives common in deduction such as AND, OR and NOT) with lasting significance. An argument stated in propositional logic could now be verified formally, irrespective of the subject matter (Mackenzie 1995, p. 8).

The development of modern mathematical logic has its roots in the nineteenth century. Around 1850, the classical philosophical tradition of logic converged with developments in mathematics and resulted in a "huge thematic expansion" as well as changing conceptions of the scope and content of logic. By the late nineteenth century, the logic of mathematics included the theory of sets and the theory of relations, combined with elementary propositional and predicate logic (Ferreirós 2001, p. 443-444).

The formal systems developed by Gottlob Frege (1848-1925) and Bertrand Russell (1872-1970) already included set theory, but their systems were still different from modern first-order logic and were, in fact, higher-order logic. Not withstanding, Frege is credited with laying the foundation for the development of first-order logic in his 1879 *Begriffsschrift* (Frege 1879; Mackenzie 1995; Ferreirós 2001), and the introduction of a theory of classes (sets) in his 1893 *Grundgesetze der Arithmetik* (Ferreirós 2001, p. 444). We shall see in the next section that Frege's theory of sets, based on the principle of unrestricted comprehension, was proven contradictory by Russell.

In his formal system, Frege considered predicates like "is prime" as signifying a function $f()$ of one variable that maps its argument x to a truth value $f(x)$, i.e. a proposition that is either true or false. Although his diagrammatic notation has been abandoned, Frege's system was the archetype for establishing first-order logic as a formal system consisting of the logical connectives, terms (constants, variables and functions), predicates and quantifiers (Plaisted 2015, p. 4). Quantification over variables distinguishes first-order logic from propositional logic (since the latter lacks quantification). However, quantification ranges over variables only (not over predicates or functions), which distinguishes first-order logic from higher-order logic.

In 1903 Russell published *The Principles of Mathematics*, expressing the view that all of mathematics is symbolic logic[1]. At the time of expressing this sentiment, first-order logic was still not established as the most natural system for logic and many logicians continued to argue in favour of higher-order logic. In fact, only around 1940-1950 (especially after World War II) did the community of logicians come to agree that first-order logic is "formal logic *par excellence*" and "the paradigm logical system" (Ferreirós 2001, p. 448).

2.1.2 Deductive Systems for First-order Logic

One of the goals of formal logic (from Leibniz to Frege) had been the development of a symbolic system that could guarantee correct reasoning independent of any understanding of the domain of discourse (Mackenzie 1995, p. 12). Deductive systems and proof procedures were developed that treated a proof as a finite sequence of formulas: the last formula is the theorem to be proved, and every formula in the sequence is either an axiom or is derived from a previous formula by the rules of inference. These rules operate syntactically on formulas and do not depend on the meaning of symbols (Mackenzie 1995, p. 9).

Even when inference is purely syntactical, the set of axioms must be mutually consistent. Consistency is a precondition of sound inference, otherwise it might be possible to derive both a proposition and its negation from the same set of axioms. Every deductive system must therefore be subjected to a proof of its soundness: every syntactically derived formula must be a theorem of the system and must be true in all interpretations in the semantic theory of the system. In other words, when starting from consistent axioms, the proof rules cannot

[1] "The fact that all Mathematics is Symbolic Logic is one of the greatest discoveries of our age; and when this fact has been established, the remainder of the principles of mathematics consists in the analysis of Symbolic Logic itself" (Russell 1903, p. 5).

allow a false inference.

There is also the question of whether it is always possible to decide if an arbitrary formula is a theorem. The problem was first formulated by David Hilbert (1862-1943) and is known as the *Entscheidungsproblem*: is there a mechanical procedure (algorithm) that could, in a finite number of steps, determine if an arbitrary mathematical proposition is provable? In 1936, Alonzo Church (1903-1995) and Alan Turing (1912-1954) independently proved that the *Entscheidungsproblem* has no general solution. Church formulated the λ-calculus and established that λ-definable functions are undecidable . These conclusions where applied to first-order logic in his paper "A Note on the *Entscheidungsproblem*" to prove that the *Entscheidungsproblem* cannot be solved in general. Turing made the notion of "mechanical procedure" precise by defining the "Turing machine". Turing proved that no such algorithm exists by showing that the question of whether a Turing machine will halt on arbitrary input is undecidable.

2.2 Developments in Set Theory

Set theory is a remarkable achievement of modern mathematics, since it serves as a foundation for all of mathematics: all mathematical concepts can be expressed within axiomatic set theory. However, set theory was not always considered an independent branch of mathematics and, much like the development of first-order logic, axiomatic set theory needed many decades to mature into the sophisticated modelling tool it eventually turned out to be. Our discussion in this section is based on the detailed analysis of the emergence of set theory in Ferreirós (2016).

2.2.1 The Origin of Set Theory

Despite exerting little influence on later developments in set theory, Bernard Bolzano (1781-1848) was one of the first nineteenth century mathematicians preceding Georg Cantor (1845-1918) who argued in favour of the acceptance of the infinite in mathematics. Bolzano recognized the possibility of creating a bijection between two real intervals, e.g., $[0, 1]$ can be mapped to $[6, 10]$ by the bijective function $f : [0, 1] \to [6, 10]$ given by $f(x) = (10 - 6) * x + 6 = 4x + 6$. However, he did not draw the conclusion that these sets must have the same cardinality and failed to discover the concept of cardinality altogether. The strong influences of geometry and topology may explain why Bolzano resisted these conclusions (Ferreirós 2016).

Peter Gustav Lejeune Dirichlet (1805-1859) proposed the novel idea that a function was an injective correspondence between numerical values, irrespective of whether or not the correspondence could be represented by a formula. Dirichlet was offering a novel alternative to viewing a function in analytic terms only.

Dirichlet had a strong influence on the work of Bernhard Riemann (1826-1866), particularly the former's views on functions. Riemann became a proponent of a new style of mathematics that sought to base all of

mathematics on the notion of a set or class. His vision for the role of sets in mathematics had a profound effect on the work of Richard Dedekind (1831-1916) and Cantor. In 1868, Dedekind published Riemann's paper on trigonometric series (presenting the Riemann integral). The paper advanced the field of real analysis significantly by stimulating interest in the study of discontinuous functions.

In 1871, Dedekind proposed an essentially set-theoretic development of fields and ideals of algebraic numbers. His ideas were presented in terms of set operations and contained a precise treatment of concepts such as isomorphisms (structure-preserving mappings), equivalence relations, partition sets, homomorphisms, and automorphisms.

Dedekind published another paper in 1872 that contained an axiomatic analysis of the structure of the set of real numbers, $\mathbb{R}$. $\mathbb{R}$ was defined as a complete (defined in terms of Dedekind-cuts), ordered field. Cantor provided another definition of $\mathbb{R}$ in the same year, but analysed $\mathbb{R}$ in terms of Cauchy sequences of rational numbers. Both mathematicians relied heavily, albeit implicitly, on set theory: their work relied on the assumption of a Power Set principle and both tacitly accepted the existence of the set of rational numbers, $\mathbb{Q}$, and their definitions of $\mathbb{R}$ relied on infinite sets of rational numbers.

In correspondence between Cantor and Dedekind in 1873, Cantor asked the question whether a bijection exists between the set of natural numbers, $\mathbb{N}$, and the set $\mathbb{R}$. Dedekind's reply contained a proof that the set of algebraic numbers, $\mathbb{A}$, is countable. Shortly thereafter, Cantor was able to prove that assuming $\mathbb{R}$ denumerable, leads to a contradiction. Hence, $\mathbb{R}$ is not countable. Cantor had discovered that the cardinality of $\mathbb{R}$ is strictly greater than the cardinality of $\mathbb{N}$. Years later, in 1878, Cantor conjectured that subsets of $\mathbb{R}$ are either denumerable or equal in cardinality to all of $\mathbb{R}$ — the first form of the *Continuum Hypothesis*. The discovery that uncountable sets exist "provided an impetus for the development of set theory and became a source of its depth and richness" (Hrbacek and Jech 1999, p. 90).

2.2.2 The Discovery of Transfinite Numbers

Between 1878 and 1885, Cantor continued to publish several works that helped to establish set theory as a separate branch of mathematics. In 1882, his work on point sets led Cantor to discover the transfinite ordinals. His definition contained two principles for generating transfinite ordinals:

- If a is an ordinal number, then the successor $a + 1$ is also an ordinal number;
- If γ is an infinite sequence of ordinals without a maximal element, then there exists a least ordinal b such that b immediately follows after γ.

The first transfinite ordinal is $\omega = \mathbb{N} = \{0,\ 1,\ 2,\ \ldots\}$, so we have (including the finite ordinals): $0,\ 1,\ 2,\ 3,\ \ldots,\ \omega,\ \omega{+}1,\ \omega{+}2,\ \ldots, \omega{+}\omega = \omega{\cdot}2,\ \omega{\cdot}2{+}1,\ \ldots,\ \omega{\cdot}\boldsymbol{n},\ \ldots,\ \omega{\cdot}\omega = \omega^2,\ \omega^2{+}1,\ \ldots,\ \omega^\omega,\ \ldots$

Cantor classified the transfinite ordinals into various classes: the first being the set of finite ordinals (the natural numbers $\mathbb{N}$) and the second class consisting of ω and all numbers following ω with a denumerable set of predecessors. Based on this distinction, Cantor was able to show that the cardinality of the latter class was strictly greater than the cardinality of $\mathbb{N}$, and that no other cardinality is simultaneously strictly greater than that of $\mathbb{N}$ and strictly less than the cardinality of the second ordinal class. Hence, if the cardinality of $\mathbb{N}$ is denoted by $\aleph_0$, then the cardinality of the second ordinal class is $\aleph_1$ — the $\aleph$-notation was first used by Cantor in 1895.

By continuing this line of thought, the third ordinal class consists of all ordinals whose set of predecessors has cardinality $\aleph_1$. The cardinality of the new class itself is $\aleph_2$. This process can be continued indefinitely. Hence, the transfinite ordinals served to establish an increasing sequence of transfinite cardinalities. The *Continuum Hypothesis* could now be stated very elegantly: the cardinality of $\mathbb{R} = \aleph_1$.

Finally, the study of transfinite ordinals turned Cantor's attention to the study of well-ordered sets and he conjectured that every set can be well-ordered. Hilbert included both the *Continuum Hypothesis* and the well-ordering conjecture as Problem 1 in his list of open mathematical problems published in 1900.

2.2.3 The Paradoxes

By the late nineteenth century, Hilbert and Dedekind proposed and supported the idea that all of pure mathematics can be grounded in set theory. However, the discovery of paradoxes in set theory (first by Cantor, but later also by Russell) posed a major obstacle in the path to attaining full rigour in mathematics.

For example, Cantor discovered a paradox while studying the transfinite numbers. Each transfinite ordinal is the order type of the set of its predecessors. So ω is the order type of $\{0,\ 1,\ 2,\ \ldots\}$, $\omega + 1$ is the order type of $\{0,\ 1,\ 2,\ \ldots,\ \omega\}$ etc. Hence, to each initial segment S of ordinals there corresponds a least ordinal greater than the maximal ordinal in S. But then the set T of all transfinite ordinals forms a well-ordered set and consequently there must exist an ordinal Ω greater than every ordinal in T. However, this implies $\Omega < \Omega$.

A similar argument can be applied to the set of all cardinals: the set of all cardinals must have a power set of cardinality $\aleph$. But then Cantor's Theorem[2] implies $\aleph_0 < \aleph_0$.

Cantor realised that such logical approaches to set theory, such as the systems developed by Frege and Dedekind, incorrectly assumed that any well-defined collection is also a consistent system. The essence of the matter was their use of unrestricted comprehension.

Aware of Cantor's paradoxes, Hilbert and Dedekind realised that the foundations of set theory were not unshakable. Hilbert formulated another paradox which he discussed with mathematicians in Göttingen. Among them was Ernst Zermelo (1871-1953) who was thus led to the discovery of Russell's paradox. In contemporary set-theoretic notation, the paradox can be expressed succinctly as follows: $R = \{x \mid x \notin x\}$. From the definition

[2]Cantor's Theorem states that every set X has cardinality strictly less than the cardinality of $\mathbb{P}(X)$ (Hrbacek and Jech 1999, p. 96).

of R it follows that both $R \in R$ and $R \notin R$, i.e. R is the "set" of all sets that are not members of themselves.

Russell later discovered the paradox independently whilst studying Cantor's Theorem and Frege's *Begriffsschrift*. In 1902, Russell explained the paradox to Frege in a letter as follows:

> "You state ... that a function, too, can act as the indeterminate element[3]. This I formerly believed, but now this view seems doubtful to me because of the following contradiction. Let w be the predicate: to be a predicate that cannot be predicated of itself. Can w be predicated of itself? From each answer its opposite follows. Therefore we must conclude that w is not a predicate" (Russell quoted in Van Heijenoort 1967, p. 125).

Frege clearly grasped the impact of the paradox on his own work:

> "Your discovery of the contradiction caused me the greatest surprise and, I would almost say, consternation, since it has shaken the basis on which I intended to build arithmetic... It is all the more serious since, with the loss of my Rule V, not only the foundations of my arithmetic, but also the sole possible foundations of arithmetic, seems to vanish. Yet, I should think, it must be possible to set up conditions... such that the essentials of my proofs remain intact. In any case your discovery is very remarkable and will perhaps result in a great advance in logic, unwelcome as it may seem at first glance" (Frege quoted in Van Heijenoort 1967, p. 127–8).

2.2.4 *Principia Mathematica*

In an attempt to address these paradoxes, Russell sought a new logical system that would prevent paradoxes. In a joint effort with Alfred North Whitehead (1861-1947), they produced the three volume work *Principia Mathematica*[4] (published between 1910 and 1913) which contained the Theory of Types (Ferreirós 2001, p. 445). Their goal was to establish a logical system for set theory (consisting of axioms and inference rules in symbolic logic) from which *all* mathematical truths could be derived.

Kurt Gödel (1906-1978) proved definitively that neither *Principia Mathematica* nor any other system can achieve this goal. Gödel was the first to prove the completeness of systems of first-order logic that are not extended to include arithmetic in 1930. However, his 1931 incompleteness theorem stated that every finite formal system S rich enough to express arithmetic is incomplete. S is either inconsistent or there are mathematical truths that cannot be deduced from S. In other words, if S is a rich enough formal system, it contains some well-formed and meaningful proposition p such that neither p nor $\neg\, p$ can be proven in S. As a corollary Gödel also showed that S cannot be proven consistent (sound) within S itself, but only in some larger system T which can in turn not be shown consistent within T itself.

[3] "On the other hand, it may also be that the argument is determinate and the function indeterminate" (Frege 1879, p. 23).

[4] The Modern Library placed *Principia Mathematica* twenty third in its list of top 100 English-language non-fiction books of the twentieth century. `http://www.modernlibrary.com/top-100/100-best-nonfiction/`

2.2.5 Axiomatic Set Theory

Zermelo turned his attention to axiomatic set theory. His first axiomatisation of set theory was given in the 1908 paper "*Untersuchungen über die Grundlagen der Mengenlehre, I*" which has become the basis of the modern theory of sets (Hallett 2016). Foremost among the difficulties to be addressed by Zermelo was the use of unrestricted comprehension (the source of contradictions in, for example, Frege's work). Zermelo identified a single fundamental relation, namely set membership, '$\in$', to express the idea that an object a is included in a set b, denoted by $a \in b$. Seven axioms were formulated:

1. **Extensionality** A set is determined by its elements.
2. **Elementary Sets** Three sets were postulated to exist, namely (i) the empty set $\varnothing$; (ii) the singleton set $\{a\}$ for any object a; and (iii) for any two objects a and b, the unordered pair $\{a, b\}$.
3. **Separation** For every set a and every definite property of elements in the domain of objects, there exists a set b that contains the (separated out) elements of a which satisfy the given property.
4. **Power Set** For every set a, the collection of subsets of a is also a set.
5. **Union** For every set a, the collection of elements of the elements of a forms a set.
6. **Choice** For every set a of pair-wise disjoint, non-empty sets, there exists a set that contains one element of each element of a.
7. **Infinity** There exists an infinite set I that contains $\varnothing$, and for each set $a \in I$ we have $\{a\} \in I$. For example, $\{\varnothing,\ \{\varnothing\},\ \{\{\varnothing\}\},\ \dots\} \subseteq I$.

In 1921, Abraham Fraenkel (1891-1965) pointed out some weaknesses in Zermelo's theory of sets. For example, Zermelo's theory was incapable of proving the existence of the set

$$\{I,\ \mathbb{P}(I),\ \mathbb{P}(\mathbb{P}(I)),\ \dots\}$$

Fraenkel and Thoralf Skolem (1887-1963) proposed a number of modifications to Zermelo's theory. For example, instead of the ill-defined "definite property" in the Axiom Schema of Separation, they proposed replacing the "definite property" by a predicate in first-order logic whose atomic formulas are limited to set membership and identity.

They also independently proposed replacing the Axiom Schema of Separation with the Axiom Schema of Replacement. Appending the Axiom Schema of Replacement and the Axiom of Foundation to Zermelo's set theory without the Axiom of Choice, yields Zermelo-Fraenkel set theory or simply ZF. When the Axiom of Choice (or an equivalent statement) is included, the resulting theory is known as Zermelo-Fraenkel set theory with Choice, denoted by ZFC (Hallett 2016).

2.3 Developments in Automated Theorem Proving

Deductive systems have great potential for mechanisation. But "[s]ustained efforts to automate mathematical reasoning...came only with the advent of digital computers" (Mackenzie 1995, p. 11). Our discussion of the development of automated theorem proving is divided into three sections. In Section 2.3.1 we consider the earliest known attempts at mechanised — specifically *computerised* — theorem proving. We also discuss important contributions by logicians such as Presburger, Davies and Herbrand, since their work was a precursor to Robinson's discovery of the resolution proof procedure in 1965. Section 2.3.2 discusses resolution, the challenge presented by the combinatorial explosion of the search space and finally McCune's powerful, resolution-based theorem prover OTTER. In the final part of this section, Section 2.3.3, we give a brief overview of developments in interactive theorem proving. We introduce the interactive theorem provers SAM — a series of interactive theorem provers from the 1960s — and Rodin/Event-B — the hybrid theorem proving environment used in our research.

2.3.1 Pre-Resolution

The first generally accepted automated theorem prover was the Logic Theory Machine developed in 1955 by three pioneers in the field of Artificial Intelligence, namely Allen Newell (1927-1992), Herbert Simon (1916-2001) and John Clifford Shaw (1922-1991). The Logic Theory Machine proved theorems in the propositional calculus and included the five basic axioms of propositional logic given in *Principia Mathematica* together with the following rules of inference (Mackenzie 1995, p. 11):

1. **Substitution** Every occurrence of a variable may be substituted by any expression in a theorem;
2. **Replacement** A logical connective and its definition are interchangeable; and
3. **Detachment** *Modus ponens*.

The Logic Theory Machine was able to prove 38 out of 52 theorems from Chapter 2 in *Principia Mathematica*. Russell, one of the authors of *Principia Mathematica*, responded to these results by saying that he is "delighted to know that *Principia Mathematica* can now be done by machinery... I am quite willing to believe that everything in deductive logic can be done by machinery" (quoted in Mackenzie 1995).

The approach taken to generating a proof was to start with the theorem and search for a direct proof. If no proof was found, a list of formulas (subgoals) were derived from which the theorem could be proved in one step. Proofs for these formulas were then searched for. This iterative approach terminated when

1. One of the subgoals was proved;
2. No additional subgoals could be derived; or
3. Computing resources were exhausted.

In practice, the Logic Theory Machine could only discover very short proofs, because subgoals were evaluated in the order in which they were generated. In other words, the search was neither selective nor guided and computing resources were easily exhausted. The explosive growth of the search space is known as the "combinatorial explosion" of the search space and has remained as one of the central challenges of automated theorem proving.

Another early invention was the Geometry Machine by Herbert Gelernter (1929-2015) in 1959. A crucial difference between the Geometry Machine and the Logic Theory Machine was the use of numerically encoded diagrams to represent geometric problems. These diagrams were used to guide the search effort, because subgoals could be discarded when they contradicted any of the diagrams. Pruning the search space allowed the Geometry Machine to solve a number of interesting problems; it serves as an early example of the utility of a heuristic to address the combinatorial explosion of the search space (Gelernter 1959; Mackenzie 1995).

In light of the importance of heuristics in our own work, we note that heuristics have played a central part in the development of automated theorem provers since at least the early 1960s. For example, Gelernter concludes the discussion of the Geometry Machine with the following remarks:

> "It is well at this point in our discussion to reemphasize the fact that the object of this research has not been the design of a machine capable of proving theorems in Euclidean plane geometry, or even one able to prove theorems in some undecidable system such as number theory. We are, rather, interested in understanding the use of heuristic methods (or strategies) by machines for the solution of problems that would otherwise be inaccessible to them... Both Gilmore and Wang find that for more complex formal systems, heuristics are required... to make their algorithms sufficiently selective to produce, within acceptable bounds on space and time, proofs of any great interest" (Gelernter 1959, pp. 145-6).

These early theorem provers did not make a significant contribution to mathematical logic (Mackenzie 1995, p. 13), since their development and goals were aligned with early developments in the field of Artificial Intelligence (AI). Human problem solving, intelligent behaviour and the automation of common sense reasoning was paramount in the work of the early AI pioneers (Bundy 1999; Mackenzie 1995). For example, John McCarthy (1927-2011) explicitly included a definition of common sense reasoning in the discussion of his proposed Advice Taker:

> "One will be able to assume that the *advice taker* will have available to it a fairly wide class of immediate logical consequences of anything it is told and its previous knowledge. This property is expected to have much in common with what makes us describe certain humans as having *common sense*. We shall therefore say that *A program has common sense if it automatically deduces for*

itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows" (McCarthy 1960, p. 78).

The contribution of logicians to automated theorem proving followed a different line of development. One of the areas of active research in the 1950s was the demonstration of completeness of particular proof procedures. Completeness can be proven by defining an algorithm for finding the proof of a proposition (or finding a contradiction resulting from its negation), and showing that the algorithm must terminate if the proposition is true. Several logicians realised that these algorithms could be implemented on a computer to create automated theorem provers for first-order logic (Mackenzie 1995, p. 13).

Initial attempts concentrated on automating decision procedures for various restrictions of mathematics and logic. For example, in 1929 Mojżesz Presburger[5] (1904–1943?) had shown the existence of a decision procedure for Presburger arithmetic, i.e. the first-order theory of natural numbers with addition. Martin Davis (born 1928) succeeded in implementing Presburger's procedure programmatically in 1954 at the Princeton Institute for Advanced Study (Mackenzie 1995, p. 13).

By 1960, Hao Wang (1921-1995) had implemented decision procedures for both propositional calculus and a decidable fragment of first-order logic (Mackenzie 1995, p. 13). Wang's program was able to prove all the theorems from *Principia Mathematica* that the Logic Theory Machine could prove as well as many additional theorems beyond the Logic Theory Machine's reach.

Jacques Herbrand (1908-1931) contributed an important refutation procedure in 1930 to decide the unsatisfiability of a set of clauses. According to Herbrand's theorem, the substitutions that need to be tried when searching for a contradiction are exactly the terms from the Herbrand universe: a finite or countably infinite sequence of variable-free, propositional terms formed from the constants and functions in the set of clauses (Mackenzie 1995, p. 10). In other words, Herbrand's theorem provides a method to test a first-order logic formula for validity by successively testing propositional formulas for validity (Plaisted 2015, p. 4–5).

Paul Gilmore attempted to construct a theorem prover for the full first-order logic by implementing Herbrand's theorem. However, his prover had very limited capabilities and Gilmore discussed the crippling effect of the combinatorial explosion on his prover in (Gilmore 1960).

2.3.2 Resolution

A breakthrough finally occurred with the work of John Alan Robinson (1930-2016) when he discovered the resolution proof procedure in 1965 (J. A. Robinson 1965). Robinson was influenced by a 1960 article written by Davis and Hilary Putnam (1926-2016) in which Davis and Putnam outlined a procedure where a proof is sought by repeatedly dissolving occurrences of the atomic formulas p and $\neg p$. The search terminates when an

[5] Mojżesz Presburger was a Polish Jew and he died in the Holocaust, probably in 1943.

empty clause is generated (Davis and Putnam 1960, p. 209-10).

However, when Robinson implemented their procedure, it was inefficient. In fact, in a subsequent article by Davis, George Wahl Logemann (1938-2012) and Donald W. Loveland (born 1934) describing the implementation of their algorithm for proving theorems in first-order logic, the authors admitted their disappointment with the performance of their theorem prover (Davis, Logemann, and Loveland 1962). Analysis of a particular theorem (namely that in a group, a left inverse is also a right inverse) which could not be proven, showed that their prover was also severely hampered by the combinatorial explosion in the search space.

Robinson subsequently realised that the combinatorial explosion was, in part, caused by the desire to keep the inference steps as small as possible – a decidedly human-oriented approach to automated theorem proving. Resolution on the other hand, introduced a powerful machine-oriented proof procedure: a single inference could well be too complex for a human to grasp directly. Robinson was able to show that first-order logic — with resolution as the only inference rule — is a *complete* inference mechanism (Mackenzie 1995, p. 14).

Unfortunately in practice, even resolution got bogged down by the combinatorial explosion of the proof search space. One reason is that the proof procedure does not make use of domain-specific knowledge. For example, problems in specific areas of mathematics are represented through the inclusion of axioms describing a particular field (e.g., set theory or group theory). Interest in resolution declined during the 1970s and early 1980s. However, at the Argonne National Laboratory (where Robinson was first introduced to automated reasoning), Larry Wos (born 1930) and George Robinson continued theoretical work on the resolution proof procedure and invented powerful rules such as demodulation and paramodulation (Mackenzie 1995, p. 16).

The Argonne National Laboratory group devoted significant effort to implementing Robinson's resolution mechanism (Plaisted 2015, p. 5). Although their initial prover was very slow, William McCune (1953 - 2011) rewrote the entire prover in 1987 and produced the resolution-based prover OTTER[6] (Organized Techniques for Theorem-proving and Effective Research). OTTER reached a level of sophistication that allowed McCune to attempt some open conjectures in mathematics. For example, EQP[7] (Equational Prover, also by McCune) is a variant of OTTER and was used to prove the open conjecture that all Robbins algebras are Boolean algebras (McCune 1997). Prover9[8] is the successor of OTTER and the latter — OTTER — is no longer actively developed.

2.3.3 Interactive Theorem Proving

The poor performance of automated theorem provers during the 1950s and 1960s also led to interest in interactive[9] theorem provers. Some of the earliest interactive provers, called SAM (Semi-Automated Mathematics),

[6] `http://www.cs.unm.edu/~mccune/otter/`

[7] `http://www.cs.unm.edu/~mccune/eqp/`

[8] `http://www.cs.unm.edu/~mccune/prover9/`

[9] Note that a theorem prover such as OTTER allows the user to control the proof search in a number of ways, e.g., by setting weights, changing the order or structure of input, or selecting different proof rules. However, we do not consider this explicitly interactive since the user has no further control over the proof search or proof generation once the search starts. Instead, explicitly interactive provers

are a series of provers developed by the Applied Logic Corporation. In 1966, SAM V was used to prove theorems from an article by Bumcrot on lattice theory. The mathematician using the system was able to prove a conjecture posited by Bumcrot as an immediate consequence of one of the intermediate results. This result came to be known as SAM's lemma and is considered the first contribution of an automated reasoner to mathematics (Mackenzie 1995, p. 17).

Despite the name, automated theorem proving covers a wide spectrum of automation. For example, while human guidance through configurable options may be possible, a fully automated prover will attempt to find a proof without further human intervention during the search space traversal. Other provers will only check detailed proofs produced by a human (Harrison, Urban, and Wiedijk 2014). Another option is theorem provers capable of executing proof steps automatically until there is no more progress, accepting human intervention and repeating this process until a proof is found (assuming a proof exists). The hybrid theorem proving environment Rodin — introduced below — follows this pattern of interaction.

Of particular interest to our research are the European Commission funded projects RODIN[10,11] (2004–2007), DEPLOY[12] (2008–2012) and ADVANCE[13] (2011-2014). A key aspect of the RODIN project "was the development of an open source extensible toolset to support refinement-based formal development" (Jastram 2012, p. 9). The tool set, also known as Rodin, is a modelling and theorem proving tool for Event-B (Butler and Hallerstede 2007). Event-B is a formal method for "system-level modelling and analysis" (Abrial, Butler, Hallerstede, Hoang, et al. 2010). The basic mathematical theories supported by Event-B are first-order logic and (typed) set theory.

After the RODIN project concluded and the Rodin tool set was successfully established, the DEPLOY and ADVANCE projects "addressed further development of the Rodin core and associated plug-ins in parallel with industrial-scale deployment of the Rodin tools" (Jastram 2012, p. 9).

Built on the Eclipse[14] framework, the Rodin platform can potentially support an unlimited number of theorem provers as plug-ins. The Rodin tool suite is distributed with a single (unsound) prover named NewPP and the Rodin User's Handbook (Jastram 2012) recommends the installation of the third-party Atelier B provers in order to add the PP and ML provers (Jastram 2012, p. 162). Other theorem provers are available, for example, the model checker ProB and the SMT provers CVC3, CVC4, veriT and Z3.

Rodin/Event-B is discussed in depth in Chapter 6.

allow the user to direct the search while the prover is traversing the search space.

[10] European Commission funded project IST-511599 (Jastram 2012, p. 170).

[11] `http://rodin.cs.ncl.ac.uk/index.htm`

[12] `http://www.deploy-project.eu/`

[13] `http://www.advance-ict.eu/`

[14] `http://www.eclipse.org/`

2.4 Summary

The first two sections of this chapter gave an overview of significant historical figures and events in the development of mathematical logic, foundational mathematics and proof theory during the 19^{th} and 20^{th} centuries. These developments have contributed to the inception and growth of the field of automated reasoning. Notable consequences of the developments in logic and mathematics included the establishment of first-order logic as the logical system of choice and the formulation of axiomatic set theory.

In the final section of this chapter we outlined the origin of the field of automated theorem proving, the discovery of the resolution proof procedure and the advent of interactive theorem provers. Finally, the integrated development environment (IDE) Rodin — the hybrid theorem proving environment used in our research to evaluate the OTTER heuristics — was introduced.

Chapter 3

Deductive Systems

We have seen in Chapter 2 that the idea of mechanised proof can be traced back to the work of Gottfried Leibniz in the seventeenth century. However, interest in mechanised proof remained dormant for approximately 200 years before it was revived in the latter half of the nineteenth century. Developments in mathematics and formal logic contributed to new fields of enquiry such as proof theory and deductive reasoning. Numerous deductive systems were devised and studied for both propositional logic and first-order logic. These systems were a necessary precursor of automated reasoning which gained prominence with the advent of digital computing.

The current chapter is the first of two chapters expanding on Chapter 2 to provide a technical introduction to deductive systems, and axiomatic set theory (Chapter 4). In Section 3.1 we present a brief overview of decision procedures and the reasons for preferring deductive proof over e.g., the evaluation of truth value assignments (a semantic approach). The section is concluded with a definition of deductive systems and some remarks on problems that arise from mechanised deduction.

The Gentzen system $\mathcal{G}$ is presented in Section 3.2 as an example of a simple deductive system. Example 3.1 shows a derivation within the Gentzen system $\mathcal{G}$ and illustrates the basic principle that underpins all of deductive reasoning: if all premises are true and the rules of the deductive system applied, then every conclusion that is reached must necessarily be true too.

When a deductive system is implemented in an automated reasoner, it becomes part of the reasoner's inference mechanism. Two inference mechanisms are discussed in subsequent sections. Arguably the most significant inference mechanism is *resolution* - at least in terms of the historical development of the field (Plaisted 2015). Resolution (as well as a number of adaptations) is discussed in Section 3.3 and *term rewriting* is discussed in Section 3.4.

Finally, the chapter is concluded with a summary.

3.1 Decision Procedures versus Deductive Systems

A *decision problem* is a question in some formal system with a yes-or-no answer. The answer will typically depend on the values of some input parameters. An example of a decision problem in mathematics is the question, "Given the integers x and y, does x leave a remainder of 1 when dividing y?"

An algorithm that solves a particular decision problem is called a *decision procedure*. By definition, a decision procedure must (1) terminate after a finite number of steps and (2) produce a yes-or-no answer for the decision problem. In our example, an effective decision procedure to decide if x divides y with a remainder of 1 is long division.

In mathematical logic, typical decision problems include:

1. Is formula A satisfiable?
2. Is formula A valid?
3. Given a set of formulas U and an arbitrary formula A, is $A \in U$?
4. Given a set of formulas U and an arbitrary formula A, is A a logical consequence of U?

We now turn our attention to decision problem 4. In order to proceed with the discussion, consider the following definitions of *theory* from Ben-Ari (2004, p. 27):

> **Definition 3.1** (Theory)**.** A set of formulas $\mathcal{T}$ is a *theory* iff it is closed under logical consequence. $\mathcal{T}$ is closed under logical consequence iff for all formulas A, if $\mathcal{T} \vDash A$ then $A \in \mathcal{T}$. The elements of $\mathcal{T}$ are called theorems. ■
>
> **Definition 3.2** (Theory of U)**.** Let U be a set of formulas. $\mathcal{T}(U) = \{A \mid U \vDash A\}$ is called the *theory of* U. The formulas of U are called *axioms* and the theory $\mathcal{T}(U)$ is axiomatisable. ■

From these definitions we see that every element of $\mathcal{T}(U)$ is a logical consequence of the set of axioms U. Decision problem (4) can therefore be stated as follows: Given a set of formulas U and an arbitrary formula A, is $A \in \mathcal{T}(U)$?

Assume $U = \{A_1, A_2, \ldots, A_n\}$. Then $A \in \mathcal{T}(U)$ iff a decision procedure for validity answers 'yes' on the formula $A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow A$.

For finite U, the *method of truth tables* is a decision procedure for the propositional calculus[1]. For example, let $U = \{p \Rightarrow q\}$ and $A = \neg p \vee q$. Then we see that $(p \Rightarrow q) \Rightarrow (\neg p \vee q)$ is valid, i.e. $A \in \mathcal{T}(U)$, because every row of its truth table evaluates to T:

[1] A truth table for some formula A requires a row for each combination of possible assignments of truth values to the atoms in A. This results in exponential growth in the number of rows, because n atoms can be assigned truth values in 2^n different ways. Better performant decision procedures had been proposed by 1960, e.g., see Davis and Putnam (1960, p. 201).

p	q	$p \Rightarrow q$	$\neg p \vee q$	$(p \Rightarrow q) \Rightarrow (\neg p \vee q)$
T	T	T	T	T
T	F	F	F	T
F	T	T	T	T
F	F	T	T	T

However, there are several drawbacks when relying on decision procedures (Ben-Ari 2004, p. 43):

1. The set U can be infinite. For example, in the axiomatisation of set theory, the Axiom Schema of Comprehension specifies that an infinite collection of subset formulas are axioms (see Section 4.3 below);
2. Some theories do not have decision procedures like the propositional calculus, e.g., a first-order theory that includes the Peano axioms; and
3. Since a decision procedure produces only a yes-or-no answer, no intermediate results can be obtained.

Deductive proof is an alternative to decision procedures for *semantic* concepts such as interpretation and consequence. Instead of dealing with semantics, one can choose a set of axioms and a set of *syntactic* rules for deducing new formulas from the axioms. Since these rules can be applied mechanically, deductive systems can be implemented in computer programs and the rules of the deductive system included in an inference mechanism. In essence, an automated theorem prover is an automated deductive system.

Deductive systems are therefore central to the research reported on in this dissertation. We give the definition of a deductive system from (Ben-Ari 2004, p. 43):

> **Definition 3.3** (Deductive system)**.** A *deductive system* is a set of axioms and a set of rules of inference. A proof in a deductive system is a sequence of sets of formulas such that each element is either an axiom or it can be derived from previous elements of the sequence using a rule of inference.
>
> If $\{A\}$ is the last element of the sequence, A is a *theorem*, the sequence is a proof of A, and A is provable, denoted by $\vdash A$. ■

Seeing that deduction is purely syntactical, some of the problems that arise from relying on decision procedures disappear (Ben-Ari 2004, p. 44):

1. Even if the set of axioms is infinite, only a finite number of axioms will appear in any proof;
2. Every proof consists of a finite sequence of sets of formulas; and
3. Once a theorem is proved, it can be used in other proofs just like an axiom.

Unfortunately, deduction itself introduces new problems. For instance, since deduction is defined in terms of syntactic formula manipulation, efficient systematic search procedures may not exist. In fact, in most deduc-

tive systems, any axiom can be used at any point in the proof sequence. Deductive proof attempts are, therefore, often hindered by the combinatorial explosion in the search space (see Section 2.3).

In the next section we introduce the Gentzen system $\mathcal{G}$ to illustrate the notion of proof in a deductive system.

3.2 The Gentzen System $\mathcal{G}$

The Gentzen system $\mathcal{G}$ is a deductive system where each rule of inference eliminates, when applied to a formula, a connective from the formula. The elimination of connectives produces simpler subgoals from more complex goals (Plaisted 1993, p. 281). A simplified definition of the Gentzen system $\mathcal{G}$ appears in (Ben-Ari 2004, p. 45):

Definition 3.4 (Gentzen system $\mathcal{G}$)**.** The *Gentzen system* $\mathcal{G}$ is a deductive system. Axioms are sets of formulas containing a pair of complementary literals. The rules of inference are:

$$\frac{\vdash U_1 \cup \{\alpha_1, \alpha_2\}}{\vdash U_1 \cup \{\alpha\}} \qquad\qquad \frac{\vdash U_1 \cup \{\beta_1\} \qquad \vdash U_2 \cup \{\beta_2\}}{\vdash U_1 \cup U_2 \cup \{\beta\}}$$

where the set or sets of formulas above the line are called *premises* and the set of formulas below the line is called the *conclusion*.

The classification of α- and β-formulas are as follows:

α	α_1	α_2
A	$\neg\neg A$	
$\neg(A_1 \wedge A_2)$	$\neg A_1$	$\neg A_2$
$A_1 \vee A_2$	A_1	A_2
$A_1 \Rightarrow A_2$	$\neg A_1$	A_2
$\neg(A_1 \Leftrightarrow A_2)$	$\neg(A_1 \Rightarrow A_2)$	$\neg(A_2 \Rightarrow A_1)$

β	β_1	β_2
$B_1 \wedge B_2$	B_1	B_2
$\neg(B_1 \vee B_2)$	$\neg B_1$	$\neg B_2$
$\neg(B_1 \Rightarrow B_2)$	B_1	$\neg B_2$
$B_1 \Leftrightarrow B_2$	$B_1 \Rightarrow B_2$	$B_2 \Rightarrow B_1$

Table 3.1: Rules of Inference for α- and β-formulas

■

Each set of formulas in Definition 3.4 is an implicit disjunction and the disjunction of the formulas is valid since it contains a complementary pair of literals. The α-rules are merely a formalisation of a set of formulas as a disjunction. The β-rules are inferred using the distribution of disjunction over conjunction.

We conclude this section with an example that illustrates the derivation of a proof for the tautology

$$(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$$

The choice of axioms is no coincidence: applying the rules in the *backwards direction* yields a *context sensitive* inference mechanism that progresses from the goal to the axioms. A proof is obtained when each subgoal is

reduced to a set of formulas containing a pair of complementary literals. A similar approach is followed in term rewriting systems (Section 3.4.3).

Example 3.1. We show that $\vdash (p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$ in $\mathcal{G}$:

1	$\vdash p, q, \neg p$	axiom
2	$\vdash \neg q, q, \neg p$	axiom
3	$\vdash \neg (p \Rightarrow q), q, \neg p$	$\beta\Rightarrow, 1, 2$
4	$\vdash \neg (p \Rightarrow q), \neg \neg q, \neg p$	$\alpha\neg, 3$
5	$\vdash \neg (p \Rightarrow q), (\neg q \Rightarrow \neg p)$	$\alpha\Rightarrow, 4$
6	$\vdash (p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$	$\alpha\Rightarrow, 5$
7	$\vdash \neg q, \neg p, q$	axiom
8	$\vdash p, \neg p, q$	axiom
9	$\vdash \neg \neg p, \neg p, q$	$\alpha\neg, 8$
10	$\vdash \neg (\neg q \Rightarrow \neg p), \neg p, q$	$\beta\Rightarrow, 7, 9$
11	$\vdash \neg (\neg q \Rightarrow \neg p), (p \Rightarrow q)$	$\alpha\Rightarrow, 10$
12	$\vdash (\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$	$\alpha\Rightarrow, 11$
13	$\vdash (p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$	$\beta\Rightarrow, 6, 12$

■

3.3 Resolution

Resolution is another inference mechanism for deductive proof searching and was developed by J. A. Robinson in 1965. The method enjoys a property that, although not necessary, is desirable for a deductive system, namely that it is "easy to mechanize an efficient proof search" (Ben-Ari 2004, p. 67). Plaisted considers Robinson's work on unification and resolution to be a turning point in the history of automated theorem proving and cites Robinson's 1965 article *A machine-oriented logic based on the resolution principle* (J. A. Robinson 1965) as the dawn of the modern era of theorem proving (Plaisted 2015, p. 5).

3.3.1 Ground Resolution

Resolution is based on the notion of a set of clauses. A clause can be regarded as any finite set of literals. An important result by Skolem states that for every closed predicate formula A there exists a formula A' in clausal form such that A is satisfiable if and only if A' is satisfiable (Ben-Ari 2004).

Now, assume L, K_1, K_2, …, K_k, M_1, M_2, …, M_m are literals, so that $L \vee K_1 \vee K_2 \vee \cdots \vee K_k$ and $\neg L \vee M_1 \vee M_2 \vee \cdots \vee M_m$ are clauses. Note that the disjunction sign is often replaced by a comma in the

literature, hence these clauses can also be written as $L, K_1, K_2, \ldots, K_k$ and $\neg L, M_1, M_2, \ldots, M_m$. The ground resolution rule can then be stated as follows:

Premise $L, K_1, K_2, \ldots, K_k$

Premise $\neg L, M_1, M_2, \ldots, M_m$

Conclusion $K_1, K_2, \ldots, K_k, M_1, M_2, \ldots, M_m$

The conclusion is also known as the *resolvent* of the premises and the premises are called the *parent clauses* of the resolvent clause. The literals L and $\neg L$ are the *resolution literals*. Any interpretation that satisfies the parent clauses will also satisfy the resolvent, because the resolvent is a logical consequence of its parent clauses. Stated differently, the resolvent is satisfiable if and only if the parent clauses are (mutually) satisfiable (Ben-Ari 2004, p. 153).

The ground resolution rule can, however, only be applied to *ground literals*. Due to this limitation, "ground resolution is hardly a useful refutation procedure for the predicate calculus since the number of ground terms is both unbounded and unstructured" (Ben-Ari 2004, p. 153).

3.3.2 General Resolution

When two clauses contain literals that are not complementary as they stand, but can be made complementary after substitution for variables, resolution can still be applied. For example, the two literals $P(x)$ and $\neg P(y)$ are not complementary since the variables x and y are different. They can be made complementary by instantiating x and y with a. This is called *substitution* and can be defined in terms of the mapping $\sigma = \{x \mapsto a, y \mapsto a\}$.

The *unifier* of a set of atoms is a substitution that makes the atoms of the set identical (Ben-Ari 2004, p. 155). For example, applying σ to either of $P(x)$ or $P(y)$ results in $P(a)$. There are infinitely many substitutions that result in $P(x)$ and $P(y)$ being complementary. If μ is a unifier of a set of atoms, then μ is a *most general unifier* (MGU) if any other unifier θ can be obtained from μ by a further substitution λ such that $\theta = \mu\lambda$ (Ben-Ari 2004, p. 155). One such MGU in terms of x and y is $\{x \mapsto y\}$.

Now the resolution rule can be applied to non–ground clauses by performing unification as an integral part of the rule – this is called the *general resolution* rule. The general resolution rule can be stated as follows:

Premise $L, K_1, K_2, \ldots, K_k$

Premise $\neg L', M_1, M_2, \ldots, M_m$

Conclusion $\sigma K_1, \sigma K_2, \ldots, \sigma K_k, \sigma M_1, \sigma M_2, \ldots, \sigma M_m$

where σ is the MGU of L and $\neg L'$, and the parent clauses do not share the same variables[2].

[2]This is easily achieved by standardising apart, which is to rename all the variables in one of the clauses before it is used in the resolution rule. Since variables in a clause are implicitly universally quantified, renaming does not change satisfiability (Ben-Ari 2004, p. 164).

3.3.3 Soundness and Completeness

The resolution theorem proving mechanism is often applied to derive a proof by contradiction. If the clause C can be derived from the set of clauses $P_1, P_2, \ldots, P_n$, then the set $S = \{P_1, P_2, \ldots, P_n, \neg C\}$ is unsatisfiable. That is, $S \vDash \square$ (the empty clause). The resolution principle is sound and refutation complete, because it can always generate the empty clause from an unsatisfiable set of clauses[3] (Van der Poll 2000, p. 56). Ben-Ari presents proofs of both the soundness and completeness of resolution in (Ben-Ari 2004, p. 169).

3.4 Term Rewriting

3.4.1 Terms

Term rewriting deals with terms. For example, in the mathematical formula $f(x) = x + 1$, each side of the formula is a term. Terms are built from variables and functions, and each function has an associated natural number (the *arity* of the function) indicating how many parameters the function takes. A *term* can then be defined as follows (Jolk 2005): (1) Variables are terms; and (2) A function symbol of *arity n* applied to *n* terms yields another term.

Example 3.2 (Terms)**.** Let $V = \{x, y\}$ be a set of variables and let $F = \{f, g\}$ be a set of function symbols. Let f and g have *arity* 1 and 2 respectively.

Then the following are terms generated from the symbols in V and F:

$$x, y,$$

$$f(x),\ f(y),\ g(x,x),\ g(x,y),\ g(y,x),\ g(y,y),$$

$$f(f(x)),\ f(f(f(x))),\ f(f(f(f(x)))), \ldots,$$

$$f(f(y)),\ f(f(f(y))),\ f(f(f(f(y)))), \ldots,$$

$$g(f(x),x),\ g(x,f(x)),\ g(f(x),f(x)),$$

$$f(g(f(x),x)),\ f(g(x,f(x))),\ f(g(f(x),f(x))),$$

$$g(g(x,y),x),\ g(g(x,y),g(x,x)),\ g(g(x,y),g(g(x,x),f(x))), \ldots$$

■

It is clear from Example 3.2 that terms of any arbitrary depth can be constructed when the sets of variables and functions are non-empty, and at least one function has *arity* > 0.

[3] One should bear in mind that this statement assumes unlimited resources. In reality, it is remarkably simple to construct proofs that exhaust the available computing resources.

3.4.2 Equational Systems

An equation is an expression of the form $s = t$ and a set of equations is called an *equational system*.

Example 3.3 (Group Axioms)**.** Let the operators e, i and $*$ have *arity* 0, 1 and 2 respectively. Consider the equational system E consisting of the following equations:

$$x * e = x$$

$$x * i(x) = e$$

$$(x * y) * z = x * (y * z)$$

If the equations in E hold for all x, y, z in a set G, then $\langle G, * \rangle$ is a group with identity e and inverse of $x \in G$ given by $i(x)$.

An equational proof of $e * x = x$ can then be constructed (Hsiang et al. 1992):

$$\begin{aligned} e * x &= e * (x * e) = e * (x * (i(x) * i(i(x)))) = e * ((x * i(x)) * i(i(x))) \\ &= e * (e * i(i(x))) = (e * e) * i(i(x)) = e * i(i(x)) \\ &= (x * i(x)) * i(i(x)) = x * (i(x) * i(i(x))) = x * e = x \end{aligned}$$

■

Each equation in an equational system can be applied bidirectionally when constructing proofs. Equational systems form the basis of term rewriting systems, but rewriting systems direct an equation $r = s$ into the rewrite rule $r \to s$. The rule indicates that instances of r may be replaced by instances of s, but replacements of s by r are not allowed (Plaisted 1993, p. 280).

3.4.3 Term Rewriting Systems

Not only does a term rewriting system have unidirectional rewrite rules derived from equations, the system is also expected to replace terms by *simpler* terms. For example, one could orient rewrite rules towards smaller terms. So, an equation such as $a \times 1 = a$ could be oriented as $a \times 1 \to a$ to avoid the generation of terms such as $(a \times 1) \times 1$, $((a \times 1) \times 1) \times 1, \ldots$

A term rewriter becomes a theorem prover when an inference rule is associated with each rewrite rule (Plaisted 1993, p. 280-281). Let $s \to t$ be a rewrite rule, then the associated inference rule is:

$$\frac{A[s\rho] \quad t \to s}{A[t\rho]}$$

The rewrite rule is specified in the backwards direction so that the search for a proof can progress from the goal. In other words, a proof obligation is discharged when the goal can be transformed into one of the

equations (i.e. one of the axioms) of the associated equational system through the application of a chain of inference rules applied in the backwards direction.

3.5 Summary

The current chapter was the first of two chapters expanding on Chapter 2 to provide a technical introduction to deductive systems, and axiomatic set theory (Chapter 4). In this chapter we have introduced decision procedures and deductive systems. The Gentzen system $\mathcal{G}$ was presented to illustrate how a proof is derived in a deductive system. The final sections of the chapter gave an overview of the inference mechanisms *resolution* and *term rewriting*.

Chapter 4

Axiomatic Set Theory

The current chapter is the second of two chapters expanding on Chapter 2 to provide a technical introduction to deductive systems and axiomatic set theory.

In Chapter 3, we discussed deductive systems and presented the inference mechanisms resolution and term rewriting. In the current chapter we present an introduction to the axiomatisation of set theory known as Zermelo-Fraenkel set theory.

Every branch of contemporary mathematics (with the exception of category theory) can be defined in terms of set theory and their properties proved from the Zermelo-Fraenkel axioms with Choice (ZFC). For example, the fundamental objects of topology, algebra or functional analysis (topological spaces, vector spaces, groups, rings, Banach spaces etc.) are typically defined as sets of a specific kind. Topologic, algebraic and analytic properties are derived from the properties of these sets, which can be obtained as consequences of the axioms of ZFC (Hrbacek and Jech 1999, p. 268).

The Zermelo-Fraenkel axiomatisation of set theory allows for the first-order logic representation of set-theoretic problems. Hence, set-theoretic problems can be proved with automated reasoners that accept input in either first-order logic, or the more evolved notation of set theory itself. The ZFC axioms ensure that the universe of discourse is not empty and contains only sets.

We introduce the ZFC axioms following the discussions in Hrbacek and Jech (1999) and Enderton (1977). Section 4.11 presents two examples that illustrate how a set-theoretic problem can be rewritten in first-order logic and finally the chapter is concluded with a summary in Section 4.12.

4.1 Axiom of Existence

There exists a set called the empty set that has no elements. The empty set is denoted by $\varnothing$.

$$\exists \varnothing \, \forall x (x \notin \varnothing)$$

We note that it does not yet follow that $\varnothing$ is unique. The Axiom of Extensionality is required to show the uniqueness of $\varnothing$.

4.2 Axiom of Extensionality

If all the elements of the set X are in set Y and all the elements of Y are in X, then $X = Y$.

$$\forall A \, \forall B (\forall x (x \in A \Leftrightarrow x \in B) \Rightarrow A = B)$$

4.3 Axiom Schema of Comprehension

The Axiom Schema of Comprehension is also known as the Axiom Schema of Separation in keeping with Zermelo's use of the German term "aussonderung", from the German for "sonderung" (to separate) and "aus" (out). Enderton calls this axiom schema the Subset Axioms (Enderton 1977, p. 21).

Let $P(x)$ be a property (not containing B) of x. Then, for any set A, there exists a set B such that $x \in B$ iff $x \in A$ and $P(x)$ holds.

$$\forall A \, \exists B \, \forall x (x \in B \Leftrightarrow x \in A \wedge P(x)) \tag{4.1}$$

Before axiomatic set theory was developed, sets were constructed by the axiom schema of unrestricted comprehension. Simply put, the restriction in the Axiom Schema of Comprehension, namely $x \in A$, was omitted. Unrestricted comprehension takes the form:

$$\forall A \, \exists B \, \forall x (x \in B \Leftrightarrow P(x)).$$

A number of paradoxes are now easily obtained:

1. Let $P(x)$ be the property $x = x$ and we have $\forall A \, \exists B \, \forall x (x \in B \Leftrightarrow x = x)$ which defines B as a universal set. This is known as Cantor's second antinomy.
2. Let $P(x)$ be the property $x \notin x$ and we have $\forall A \, \exists B \, \forall x (x \in B \Leftrightarrow x \notin x)$ which defines B as the set of all sets that do not contain themselves as elements. This is known as Russell's paradox.

The restriction $x \in A$ in the Axiom Schema of Comprehension removes these and similar paradoxes.

Next we define a number of instances of the Axiom Schema of Comprehension.

4.3.1 Subset

It follows from (4.1) that B is a *subset* of A: Let $P(x)$ be the property $x \in B \Rightarrow x \in A$. Then P yields exactly the definition of *subset* as an instance of the Axiom Schema of Comprehension:

$$\forall A \, \exists B \, \forall x (x \in B \Leftrightarrow x \in A \wedge (x \in B \Rightarrow x \in A))$$

which can be abbreviated to

$$\forall A \exists B(B \subseteq A \Leftrightarrow \forall x(x \in B \Rightarrow x \in A))$$

4.3.2 Intersection

For sets A and B, there exists a set C such that $x \in C$ iff $x \in A \wedge x \in B$.

$$\forall A \forall B \exists C \forall x(x \in C \Leftrightarrow x \in A \wedge x \in B)$$

When C is the *intersection* of A and B, we write $A \cap B = C$.

4.3.3 Arbitrary Intersection

For a non-empty system of sets S, the *arbitrary intersection* of S is the set U that contains every x that belongs to every member of S.

Stated as an instance of the Axiom Schema of Comprehension, let $C \in S$, where S is a non-empty system of sets. Then there exists a set U such that $x \in U$ iff $x \in C$ and x belongs to every member of S.

$$\forall S \exists U \forall x(x \in U \Leftrightarrow x \in C \wedge \forall T(T \in S \Rightarrow x \in T))$$

Since C is arbitrary, we arrive at the following definition of U (Enderton 1977, p. 25):

$$\forall S \exists U \forall x(x \in U \Leftrightarrow \forall T(T \in S \Rightarrow x \in T))$$

U is called the *arbitrary intersection* of S and is denoted by $U = \bigcap S$. For example, if $S = \{\{0,1\},\{1\},\{1,2\}\}$, then $U = \bigcap\{\{0,1\},\{1\},\{1,2\}\} = \{0,1\} \cap \{1\} \cap \{1,2\} = \{1\}$.

Finally, we note that $\bigcap\{X,Y\} = X \cap Y$:

$$\begin{aligned} x \in \bigcap\{X,Y\} &\Leftrightarrow x \in X \wedge x \in Y \\ &\Leftrightarrow x \in X \cap Y \end{aligned}$$

4.3.4 Difference (Relative complement)

For sets A and B, there exists a set C such that $x \in C$ iff $x \in A \wedge x \notin B$.

$$\forall A \forall B \exists C \forall x(x \in C \Leftrightarrow x \in A \wedge x \notin B)$$

When C is the *difference* of A and B, we write $A - B = C$.

4.3.5 Symmetric Difference

For sets A and B, there exists a set C such that $x \in C$ iff $x \in A - B \cup x \in B - A$.

When C is the *symmetric difference* of A and B, we write $A + B = C$.

4.4 Axiom of Pair

For any sets A and B, there exists a set C such that $x \in C$ iff $x = A$ or $x = B$.

$$\forall A \forall B \exists C \forall x (x \in C \Leftrightarrow (x = A \vee x = B))$$

4.5 Axiom of Union

For any set S, there exists a set U such that $x \in U$ iff $x \in A$ for some $A \in S$.

$$\forall S \exists U \forall x (x \in U \Leftrightarrow \exists A (x \in A \wedge A \in S))$$

U is called the *arbitrary union* of S and is denoted by $U = \bigcup S$. For example, if $S = \{\{0\}, \{1\}, \{2\}\}$, then $U = \bigcup \{\{0\}, \{1\}, \{2\}\} = \{0\} \cup \{1\} \cup \{2\} = \{0, 1, 2\}$.

When $S = \{s, t\}$ contains only two elements, we have

$$\bigcup S = \bigcup \{s, t\} = s \cup t$$

4.6 Axiom of Power Set

For any set S, there exists a set P such that $X \in P$ iff $X \subseteq S$.

$$\forall S \exists P \forall X (X \in P \Leftrightarrow X \subseteq S)$$

4.7 Axiom of Infinity

An inductive set exists. A set S is inductive when $\varnothing \in S$ and $x \in S \Rightarrow x \cup \{x\} \in S$, i.e. when S contains x, then S also contains the successor $x \cup \{x\}$ of x.

$$\exists S (\varnothing \in S \wedge (x \in S \Rightarrow x \cup \{x\} \in S))$$

One such infinite set is the set of natural numbers. The following representation of the natural numbers by Von Neumann has become standard (Enderton 1977, p. 67):

$$\begin{aligned} 0 &= \varnothing \\ 1 &= \{0\} = \{\varnothing\} \\ 2 &= \{0, 1\} = \{\varnothing, \{\varnothing\}\} \\ 3 &= \{0, 1, 2\} = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\} \\ 4 &= \ldots \end{aligned}$$

4.8 Axiom Schema of Replacement

Let $P(x, y)$ be a property such that for every x there is a unique y for which $P(x, y)$ holds. Then for every set A, there exists a set B such that for every $x \in A$ there exists $y \in B$ for which $P(x, y)$ holds. In other words, if P is functional, then there exists a set B that is the functional image of A under P.

$$[\forall x \forall y_1 \forall y_2 (P(x, y_1) = P(x, y_2) \Rightarrow y_1 = y_2)]$$
$$\Rightarrow \quad [\forall A \exists B \forall x (x \in A \Rightarrow \exists y (y \in B \wedge P(x, y)))]$$

4.9 Axiom of Foundation

All sets are well-founded. A set S is well-founded iff for every $X \subseteq S$, $X \neq \varnothing$, there exists $a \in X$ such that $a \cap X = \varnothing$.

The Axiom of Foundation is also known as the Axiom of Regularity. An important consequence of this axiom is that no set can be a member of itself. We conclude the presentation of the Axiom of Foundation with a remark by Hrbacek and Jech (1999, p. 259):

> "It should be stressed that, whether or not one accepts the Axiom of Foundation, makes no difference as far as the development of ordinary mathematics in set theory is concerned. Natural numbers, integers, real numbers and functions on them, and even cardinal numbers and ordinal numbers have been defined, and their properties proved... without any use of the Axiom of Foundation."

4.10 Axiom of Choice

Every set of non-empty sets S has a choice function g. A function g defined on a system of sets S is a choice function for S if $g(X) \in X$ for all non-empty $X \in S$.

$$\forall S[\forall X(X \in A \Rightarrow X \neq \varnothing) \Rightarrow \exists g(\mathit{func}(g) \wedge \mathit{dom}(g) = S \wedge \forall X(X \in A \Rightarrow g(X) \in X))]$$

Zermelo had formulated the Axiom of Choice in 1904, but it was not until 1963 that Paul Cohen was able to show that the axiom cannot be proved from the other ZF axioms. Hence, the Axiom of Choice is a new principle of set formation. It differs from the other set formation principles in that it is not effective: the axiom asserts the existence of certain sets (the choice functions) without describing these sets (Hrbacek and Jech 1999, p. 139).

4.11 Examples

From the discussion above, we see that a problem stated in set-theoretic notation can be rewritten in first-order logic. This approach is necessary whenever an automated theorem prover does not accept the highly evolved notation of set theory. In our research, the provers OTTER/Prover9, Vampire and Gandalf all accept input in first-order logic only. The Event-B input language, on the other hand, includes set-theoretic symbols and set-theoretic problems can be stated more naturally.

Let us convert

$$\{a,b\} \cup \{c\} = \{a,b,c\} \tag{4.2}$$

to a list of first-order logic formulas, where "*elem*" is used for "$\in$":

$$\begin{aligned}
& [\forall A \forall B(\forall x(x \in A \Leftrightarrow x \in B) \Rightarrow A = B) \\
\wedge \quad & \forall x(elem(x,P) \Leftrightarrow x = a \vee x = b) \\
\wedge \quad & \forall x(elem(x,Q) \Leftrightarrow x = c) \\
\wedge \quad & \forall x(elem(x,R) \Leftrightarrow x = a \vee x = b \vee x = c) \\
\wedge \quad & \forall S \forall T \forall x(elem(x, union(S,T)) \Leftrightarrow (elem(x,S) \vee elem(x,T)))] \\
\Rightarrow \quad & union(P,Q) = R.
\end{aligned} \tag{4.3}$$

Then (4.3) is a first-order logic representation of (4.2).

A final example will suffice to illustrate the convenience of set-theoretic notation. For $A \subseteq S$ and $B \subseteq S$, let us convert the following example from Enderton (1977, p. 28)

$$[A \subseteq S \wedge B \subseteq S] \Rightarrow [S - (A \cap B) = (S - A) \cup (S - B)] \tag{4.4}$$

to a list of first-order logic formulas:

$$\begin{aligned}
& [\forall A \forall B(\forall x(x \in A \Leftrightarrow x \in B) \Rightarrow A = B) \\
\wedge \quad & subset(A,S) \\
\wedge \quad & subset(B,S) \\
\wedge \quad & \forall A \forall x(subset(x,A) \Leftrightarrow (\forall y(elem(y,x) \Rightarrow elem(y,A)))) \\
\wedge \quad & \forall S \forall T \forall x(elem(x, minus(S,T)) \Leftrightarrow (elem(x,S) \wedge \neg elem(x,T))) \\
\wedge \quad & \forall S \forall T \forall x(elem(x, union(S,T)) \Leftrightarrow (elem(x,S) \vee elem(x,T))) \\
\wedge \quad & \forall S \forall T \forall x(elem(x, intersect(S,T)) \Leftrightarrow (elem(x,S) \wedge elem(x,T)))] \\
\Rightarrow \quad & minus(S, intersect(A,B)) = union(minus(S,A), minus(S,B))
\end{aligned} \tag{4.5}$$

Again, (4.5) is a first-order logic representation of (4.4).

4.12 Summary

In this chapter — the second of two chapters expanding on Chapter 2 — we gave an overview of the axioms of Zermelo-Fraenkel set theory with Choice. The Zermelo-Fraenkel axiomatisation of set theory is crucial to the research reported on in this dissertation, because it allows for the representation of set-theoretic problems in first-order logic. Without such an axiomatisation of set theory, automated reasoners cannot discharge set-theoretic proof obligations unless they accept set-theoretic notation in their input. The chapter concluded with two examples that illustrate the conversion of formulas containing set-theoretic notation to formulas in first-order logic.

Chapter 5

Automated Theorem Provers

Deductive systems and axiomatic set theory — discussed in Chapters 3 and 4 — form the theoretical backdrop for the automated reasoners that were used in our experimental work. In this chapter we introduce the automated theorem provers that were used in preparing the experimental results we report on in Chapters 7 to 9.

Competitions have for many years served as a catalyst for automated theorem prover research, since international competitions provide a platform for the evaluation of the capabilities of automated theorem provers. The two competitions discussed in Section 5.1 are of particular interest, because many of the theorem provers discussed in this chapter have participated in one or both of these competitions. For example, the theorem provers OTTER, Vampire and Z3 have all emerged as winners in various divisions over the years.

The resolution-based prover OTTER is introduced in Section 5.2. The reasoning heuristics discussed in Chapter 7 were originally developed to guide and support OTTER while discharging set-theoretic proof obligations. Steyn (2009) evaluated the utility of these heuristics for the newer resolution-based theorem provers Vampire and Gandalf, introduced in Sections 5.3 and 5.4 respectively.

Section 5.5 introduces the Rodin theorem provers that were used to evaluate the OTTER heuristics. Rodin is an Eclipse-based[1] theorem proving environment which can be extended with plug-ins. The tool suite can therefore include any theorem prover for which a suitable plug-in has been created. We have made use of the provers PP, ML, ProB, and the SMT provers CVC3, CVC4, veriT and Z3 in our experimental work.

The chapter is concluded with a summary.

5.1 Automated Theorem Proving (ATP) Competitions

Several international competitions serve as a platform for the evaluation of the capabilities of fully automated theorem provers. Two of these competition are of interest, because many of the theorem provers discussed in this chapter have participated in one or both of these competitions. These competition are the *CADE ATP System*

[1] http://www.eclipse.org/

Competition (CASC) for first-order logic reasoners and the *Satisfiability Modulo Theories (SMT) Competition (SMT-COMP)* for Satisfiability Modulo Theories (SMT) provers.

These competitions both rely on and contribute benchmarks to collections of problems, namely the *Thousands of Problems for Theorem Provers (TPTP) Library* and the *Satisfiability Modulo Theories (SMT) Library (SMT-LIB)* respectively.

The discussion below provides an overview of CASC and the associcated TPTP library in Sections 5.1.1 and 5.1.2 respectively, followed by an overview of SMT-COMP and SMT-LIB in Sections 5.1.3 and 5.1.4 respectively.

5.1.1 The CADE ATP System Competition (CASC)

CASC, the "World Championship for Automated Theorem Proving (ATP) systems", is an annual competition for fully automated theorem proving systems for classical logic and is organized by Geoff Sutcliffe (2016, p. 99). The competition is hosted at either the International Conference on Automated Deduction (CADE) or the International Joint Conference on Automated Reasoning (IJCAR), which replaces CADE on alternate years.

The first CADE was held in 1974 and the most recent, the 26th CADE, was held in August 2017. The CADE website can be accessed online at `http://www.cadeinc.org/`.

The first IJCAR was held in 2001 and the most recent, the eighth IJCAR, was held in June 2016. The IJCAR website can be accessed online at `http://www.ijcar.org/`.

While new research into automated deduction is presented at these conferences, CASC provides an opportunity for the public evaluation of the capabilities of participating automated theorem provers. According to Sutcliffe (2016, p. 99), CASC aims to both stimulate automated theorem proving research, and to provide motivation for the development of fast, efficient and reliable automated theorem proving systems. Interaction between participating researchers is also a stimulus for the exchange of ideas and exposure to novel implementation techniques.

The first CASC was held at CADE-13 in Nancy, France, in 1996. The most recent competition was held at Gothenburg, Sweden, in 2017. Details of individual competitions can be accessed online at `http://www.cs.miami.edu/~tptp/CASC/`.

Each CASC is divided into various divisions dedicated to specific problem and system characteristics. Division winners of the previous CASC are automatically entered the following year to provide benchmarks against which progress can be measured. Furthermore, OTTER was automatically entered in every CASC from 2002 to 2011 as a fixed point of comparison. OTTER was replaced by its successor, Prover9, in 2012 when it became apparent that OTTER was no longer competitive (Sutcliffe 2016, p. 100).

Division	Description	Winner
THF	Typed higher-order form theorems	–
THN	Typed higher-order form non-theorems	–
TFA	Typed first-order with arithmetic theorems	CVC4 (2014), VampireZ3 (2015), Vampire (2016-2017)
TFN	Typed first-order with arithmetic non-theorems	CVC4 (2015)
FOF	First-order form theorems	VampireFOF (2000), Vampire (2002-2017)
FNT	First-order form non-theorems	Vampire (2015-2017)
CNF	Clause normal form theorems	Gandalf (1997-1998), Vampire (1999), VampireJC (2001), Vampire (2002-2010)
SAT	Clause normal form non-theorems (satisfiable clause sets)	OtterMACE (1999), GandalfSat (2000-2004)
EPR	Effectively propositional theorems and nontheorems	Vampire (2015)
SLH	Typed first-order theorems without arithmetic	Vampire (2017)
UEQ	Unit equality theorems	OTTER(1996)
SEM	FOF theorems based on a specified axiomatisation of a specified semantic domain	–
LTB	First-order form theorems from large theories	Vampire-LTB (2009-2011), Vampire (2012, 2015-2017)

Table 5.1: CASC Divisions and Winners (Sutcliffe 2016)

Table 5.1 shows the divisions that have existed over the years as well as the division winners[2] (Sutcliffe 2016; Sutcliffe and Urban 2016; Sutcliffe 2017a).

Problems for CASC are taken from the TPTP problem library which we discuss in the next section.

5.1.2 The Thousands of Problems for Theorem Provers (TPTP) Library

The TPTP library (a library of automated theorem proving problems) is part of the infrastructure known as the TPTP World. The latter is aimed at supporting "research, development, and deployment of Automated Theorem Proving (ATP) systems for classical logics" (Sutcliffe 2010, p. 1). Other components of the TPTP World include the Thousands of Solutions from Theorem Provers (TSTP) solution library, standards for writing automated theorem proving problems and solutions (including the TPTP language), and tools for processing automated theorem proving problems and solutions.

The problems in the TPTP library are classified into fields and domains. The seven main fields of TPTP

[2]The complete list of winners can be viewed online at `http://www.cs.miami.edu/~tptp/CASC/`.

v6.4.0 are logic, mathematics, computer science, science & engineering, social sciences, arts & humanities, and other. The complete list of domains and problem counts for TPTP v6.4.0 are included in (Sutcliffe 2017b, p. 491).

An alternative classification system divides the TPTP problems into classes based on syntactic similarities. These are called the Specialist Problem Classes (SPCs). An SPC is associated with (1) automated theorem proving techniques that are especially well suited to the problems in the class, and (2) automated theorem proving systems that can solve all the problems in the class (at least in principle). Problem ratings (i.e. a measure of the difficulty of a problem for automated theorem proving systems) are calculated for each SPC according to the number of automated theorem provers capable of solving the problem. The problem rating algorithm is discussed in (Sutcliffe 2009, p. 350).

The problems used in CASC are selected from the TPTP problem library. The CASC divisions and problem categories are similar to the SPCs used in the TPTP library and problem rating algorithm. New versions of the TPTP library are released annually after the CASC competition to prevent automated theorem proving systems from over-tuning to the TPTP in order to perform well in the competition.

The TPTP library website can be accessed online at `http://www.cs.miami.edu/~tptp/`.

5.1.3 The Satisfiability Modulo Theories Competition (SMT-COMP)

The purpose of the Satisfiability Modulo Theories (SMT) Competition (SMT-COMP) is in essence similar to that of the CADE ATP System Competition (CASC): to stimulate research and development in the SMT community, and to encourage advances in SMT solver technology (Cok, Stump, and Weber 2015).

The establishment of a common input language for SMT solvers was not trivial, because it has to accommodate a wide variety of disparate background theories. The task was undertaken as part of the SMT-LIB initiative. The SMT-COMP was instrumental in the early adoption of the common input language for SMT solvers, the SMT-LIB and the collection of performance benchmarks used for the competition. According to Barrett, Deters, et al. (2013), from the release of SMT-LIB v1.0 in 2004 to the first SMT-COMP in 2005, "the SMT community went from having its brand new standard input language but no compliant solvers and no benchmarks, to 12 compliant solvers and over 1300 benchmarks."

The first SMT-COMP was hosted in 2005 at the 17th International Conference on Computer-Aided Verification (CAV). The most recent, the twelfth SMT-COMP, was held in July 2017. The SMT-COMP website can be accessed online at `http://www.smtcomp.org/`.

Between 2006 and 2010, CVC3 and Z3 frequently ranked among the winning SMT provers in a wide range of competition divisions. By 2017, the list of winning SMT provers included Vampire, veriT, CVC4 and Z3. Results for competitions held between 2005 and 2016 can be accessed online at `http://smtcomp.`

Abbreviation	Logic Name
QF_	Quantifier-Free
A	Arrays
UF	Uninterpreted Functions
BV	Bit-Vector
L	Linear (arithmetic)
N	Non-Linear (arithmetic)
IA/RA/IRA	Integer/Real/Mixed arithmetic
IDL/RDL	Integer/Real Difference Logic

Table 5.2: SMT-COMP Abbreviations Used in Logic Names (Cok, Déharbe, and Weber 2016)

`sourceforge.net/2017/previous.shtml` and SMT-COMP-2017 results can be accessed online at `http://smtcomp.sourceforge.net/2017/results-toc.shtml`.

5.1.4 The Satisfiability Modulo Theories Library (SMT-LIB)

Similar to the TPTP library, the SMT-LIB is an international initiative that encompasses more than a large collection of problems and benchmarks for SMT solvers. Since the inception of the SMT-LIB initiative in 2003, a number of goals were identified to facilitate research and development in SMT.

According to Barrett, Fontaine, and Tinelli (2015, p. 13), the initiative's efforts were focused primarily on the following goals: providing standardized descriptions of background theories (with respect to some underlying logic) used in SMT provers, standardizing input and output languages for SMT solvers, and collecting and distributing a large library of benchmarks for SMT solvers.

The SMT-COMP was and remains instrumental in the collection of benchmarks, since additional benchmarks are collected annually to be included in the competition. Furthermore, adoption of specific SMT-LIB standards is driven by the SMT-COMP, since solvers participating in the competition must read benchmarks in the specified SMT-LIB format.

SMT-LIB released the first version of the standard input language, named SMT-LIB v1.0, in July 2004 (Barrett, Deters, et al. 2013, p. 244). The current version of the standard input language is SMT-LIB v2.6, released in June 2015 (Barrett, Fontaine, and Tinelli 2015). The names of the logics used in the SMT-LIB (along with their abbreviations) are shown in Table 5.2.

The SMT-LIB website can be accessed online at `http://smtlib.cs.uiowa.edu/`.

5.1.5 Conclusion

Standardization of input and output languages, large collections of benchmarks and annual competitions have been instrumental in advancing the state of the art in automated theorem prover implementation and develop-

ment for both classical logic theorem provers and SMT solvers. We have seen that these needs were met by CASC and the TPTP library for first-order theorem provers. Similarly, SMT-COMP and SMT-LIB were able to meet these needs for SMT solvers.

In the following sections we introduce the theorem provers and solvers used to obtain our experimental results (see Chapters 7 to 9). Many of the provers discussed below have ranked among the winning contestants at either CASC or SMT-COMP, namely OTTER, Vampire, Gandalf, CVC3, CVC4, veriT and Z3.

5.2 OTTER

OTTER (Organized Techniques for Theorem-proving and Effective Research) was the product of many years of theorem prover development at the Argonne National Laboratory and the Northern Illinois University (NIU). Many of the features and inference rules included in OTTER were developed as part of earlier theorem provers.

5.2.1 The Development of OTTER

The first phase of theorem prover development at Argonne started in 1963, approximately two years before Robinson published his pioneering paper defining the resolution inference mechanism (see Section 2.3.2). By 1970 two major systems (P1 and RW1) had been developed that contained the first attempts to implement and experiment with resolution (Lusk 1992).

Daniel Carson and Larry Wos built the first resolution theorem prover called P1 (for "Program 1"). P1 implemented binary resolution with factoring as well as forward subsumption . The set-of-support strategy and demodulation were developed during experimentation with P1 (Wos et al. 1967).

Wos, in collaboration with George Robinson, continued experimentation with demodulation. They also formulated the paramodulation inference mechanism in their 1968 article *Paramodulation and Theorem-proving in First-order Theories with Equality* (G. A. Robinson and Wos 2000). Together they built RW1 (for "Robinson-Wos 1") in the late 1960's which was capable of performing equality proofs of the problems P1 could solve.

In 1971 Ross Overbeek independently built the theorem prover FTP (for "Functionless Theorem Prover" since it did not make use of functions). He introduced a number of features that remained important components of the Argonne family of theorem provers for years to come (Lusk 1992). For example, Overbeek introduced hyperresolution, forward and backward subsumption, and weighting.

The algorithm for the basic loop of Overbeek's prover remained unchanged for several generations of Argonne provers. The search for a proof starts with a set of clauses divided into two sets: the *axioms* and the *set-of-support*. The algorithm is as follows:

While (the null clause has not been produced and the set of support is not empty)
  Choose a clause from the set of support,
    call it the given clause, and move it to the axioms
  Generate all clauses that can be deduced from the given clause
    and other clauses in the axiom set
  For each new generated clause
    Process it (rewrite to canonical form, merge literals, etc.)
    Test whether it is subsumed by any existing clause
    Perform any other desired filtering (e.g., weighting)
    If the new clause survives
      Add it to the set of support
    end if
    Perform back subsumption with new clause
    Rewrite existing clauses with [new clause] if it is a demodulator
  end for
end while

Table 5.3: The *closure algorithm* (Lusk 1992, p. 99)

Lusk (1992, p. 99) refers to Overbeek's algorithm as the *closure algorithm*. The algorithm computes the closure of a set of clauses under the operation of a set of inference rules. Two characteristics of the algorithm are significant:

- At each execution of the while loop, all consequences of the given clause are generated; and
- All generated clauses are stored to avoid having to repeat processing clauses.

The behaviour of the algorithm could be altered through user control options and heuristics, for example by

- Specifying the initial set-of-support;
- Specifying the set of inference rules;
- Specifying the given clause at the start of each iteration; and
- Specifying the choice of kept clauses during post-processing.

In 1971, Overbeek joined Northern Illinois University and met Wos. They merged the functionality and features of RW1 and FTP to produce WOS1 which was also based on the *closure algorithm*. Features and inference rules included function symbols, paramodulation, hyperresolution and weighting. A paper containing the results of experiments carried out with WOS1 was presented at the Argonne Workshop on Automated Theorem Proving. The workshop was later renamed CADE-1, i.e. the first CADE[3] (Lusk 1992).

[3] See Section 5.1.

By the late 1970's Ewing Lusk, William (Bill) McCune and Overbeek created LMA (Logic Machine Architecture) to incorporate new insights from computer science regarding the structuring of large computer systems and to provide some support for the interactive use of the prover. LMA was still powerful enough to solve all problems that the earlier Argonne and Northern Illinois University theorem provers could solve, but it was significantly slower.

McCune joined Argonne in 1984 and he created OTTER during 1987-1988. Developed in ANSI C, OTTER's first release (starting at version 0.9) was at CADE-9 in May 1988 (see Section 5.1.1). OTTER is a general-purpose, resolution-based automated reasoner for first-order logic with equality (McCune 2003; McCune and Wos 1997; Wos 1998). One of the primary objectives was to regain the speed of the systems preceding the LMA theorem prover. A secondary goal was to create a flexible tool that could easily absorb new techniques and inference rules for experimentation (by recompiling the application from modified source code).

OTTER was applied successfully to solve numerous problems in mathematics and logic. Of particular interest was the open conjecture regarding Robbins algebras (McCune 1997; Wos 1998). Many additional examples are discussed in (McCune and Wos 1997). In 2004 development of OTTER was suspended in favour of Prover9 (McCune 2005; McCune 2013).

This concludes our overview of the development of OTTER. In the remaining sections we discuss OTTER's input and syntax, and implementation of resolution.

5.2.2 Input

5.2.2.1 Parts of the Input File

OTTER input files consist of lists of commands. A command is either a flag or a parameter. Flags (boolean-valued options) are changed with the `set` and `clear` commands. Parameters (integer-valued options) are changed with the `assign` command. Table 5.4 shows the commands accepted by OTTER.

`include` (file_name).	`% read input from another file`
`op` (precedence, type, name(s)).	`% declare operator(s)`
`make_evaluable` (sym, eval-sym).	`% make a symbol evaluable`
`set` (flag_name).	`% set a flag`
`clear` (flag_name).	`% clear a flag`
`assign` (parameter_name, integer).	`% assign to a parameter`
`list` (list_name).	`% read a list of clauses`
`formula_list` (list_name).	`% read a list formulas`
`weight_list` (weight_list_name).	`% read weight templates`
`lex` (symbol_list).	`% assign an ordering on symbols`
`skolem` (symbol_list).	`% identify skolem functions`

```
lrpo_multiset_status (symbol_list).  % status for LRPO
```

Table 5.4: OTTER's Input Commands (McCune 2003, p. 12)

The `list` and `formula_list` commands are discussed below in Section 5.2.2.2 . This section is concluded with Example 5.1: an excerpt from an input file showing the commands `set`, `clear` and `assign`.

Example 5.1 (OTTER Commands)**.** The list of commands below instructs OTTER to apply unit-resulting resolution, perform no *arity* checks on symbols and to terminate the search once 1000 clauses were generated (McCune 2003, p. 13):

```
set(ur_res).           % Apply UR-resolution
                       %   to all generated clauses
clear(check_arity).    % Do not warn if symbols
                       %   have variable arities
assign(max_gen, 1000). % Terminate search after
                       %   generating 1000 clauses
```

■

5.2.2.2 Formula lists

Formulas can be divided into several lists in the input file. Four types of lists can be specified: *usable*, *set-of-support (sos)*, *demodulators* and *passive*. The usable list contains clauses that are actively used to make inferences. The clauses in the set-of-support list are not used to make inferences, but are used during the search. Clauses in the passive list are used for forward subsumption and unit conflict resolution. The passive list remains unchanged during search. Finally, the list of demodulators are used to rewrite newly inferred clauses (McCune 2003, p. 3).

Example 5.2 (Usable List)**.** The following list of formulas is an example of a usable list in OTTER input. Usable clauses are actively used to make inferences (McCune 2003, p. 3).

```
list(usable).
      x = x.  % reflexivity
      f(e, x) = x.  % left identity
      f(x, e) = x.  % right identity
end_of_list.
```

■

5.2.3 Syntax

OTTER accepts two types of statements in first-order logic:

- **Clauses** - disjunctions of terms and literals with implicitly universally quantified variables; and
- **Formulas** - with all variables explicitly quantified.

All formulas in the input are converted to clauses when not already in clausal form, since OTTER's search mechanism only operates on clauses (McCune 2003, p. 5).

5.2.4 Implementation of Resolution in OTTER

OTTER implements the general resolution rule discussed in Section 3.3.2 and allows the user to set flags in the input that result in the application of various extensions to the binary resolution rule[4]. OTTER attempts to derive the empty clause ("$F" in the output) from the input when the input contains a non-empty goal list and a possibly empty set-of-support as follows:

1. The formulas in the sos list are clausified;
2. The formulas in the goal list are negated and clausified;
3. All resulting clauses are included in the initial internal sos (as opposed to the set-of-support list in the input);
4. While the sos is not empty and $F has not been derived:
   (a) The lightest clause is selected from the set-of-support;
   (b) The clause is moved to the usable list;
   (c) OTTER then attempts to generate and process new clauses based on the inference rule (all new clauses must have the selected clause and clauses from the usable list as parents);
   (d) Clauses that pass various tests (Van der Poll (2000, p. 66) refers to retention tests such as paramodulation, subsumption and weighting) are appended to the set-of-support.

The process terminates when a refutation is found, the set-of-support list is empty, or the user interrupts the proof attempt (Van der Poll 2000, p. 67).

The set-of-support list in OTTER input is related to the set-of-support strategy. The strategy involves partitioning of the set S of input clauses, by taking a subset T that contains one or more clauses to be the set-of-support. The reasoning process then progresses by requiring one of the parent clauses of a resolvent to come from the set-of-support (in this case, T) and the other parent(s) from $S - T$ (Van der Poll 2000).

5.3 Vampire

The first version of Vampire was created by Andrei Voronkov and Alexandre Riazanov in 1993 (Kovács and Voronkov 2013).

[4]Numerous extensions and options are discussed in the OTTER User Manual (McCune 2003).

Vampire is a resolution-based theorem prover for first-order logic. Vampire has a number of features in common with OTTER, but the prover also boasts a number of novel techniques and algorithms. Since version 1.1, two inference mechanisms are supported, namely (1) binary resolution with superposition and negative selection; and (2) positive and negative hyperresolution without equality (Riazanov and Voronkov 2001).

All inference rules use a combination of compiling and indexing. Indexes always retrieve all elements to which an operation is applicable and the results of the operation on all these elements are always computed. This is known as "perfect filtering".

Additional implementation features include the use of data structures such as code trees (for forward subsumption), path indexing (for backward subsumption) and a variation on OTTER's discrimination trees for resolution (Riazanov and Voronkov 1999; Riazanov and Voronkov 2001).

Finally, Vampire includes a *limited resource strategy* for best performance when a time limit is set. When a limit is set, Vampire attempts to estimate which clauses can be processed before the end of the time limit. These clauses are called the *currently eligible clauses*. Every generated clause is inspected for current eligibility and is discarded if the test fails. The estimation algorithm is based on the selected strategy as well as a prediction of how clause generation will develop by the time limit (Riazanov and Voronkov 1999).

Vampire can be downloaded from *Vampire's Home Page* at `http://www.vprover.org/`. We conclude this section with a remark by Voronkov on his home page[5] (dated August 2015):

> "Vampire is winning CASC yet again! Vampire has won the world cup in theorem proving CASC held at the 25th International Conference on Automated Deduction (CADE-25). This time Vampire was the winner of five out of eight divisions, including the main division of the competition FOF (first-order formulas). All together [*sic*] Vampire won 35 division titles in CASC since 1999: more than any other theorem prover in the history of the competition. Vampire won all the FOF divisions since 2002. In four out of five divisions Vampire solved more problems than all other systems together."

5.4 Gandalf

Gandalf is the name of a family of interdependent, code-sharing, resolution-based automated theorem provers developed by Tanel Tammet since 1995. The specific Gandalf prover discussed here is Tammet's resolution prover for first-order logic[6] (Tammet 1997).

A number of standard resolution strategies are implemented, e.g., binary resolution, hyperresolution, set-of-support strategy, paramodulation and demodulation with automated ordering of equalities. Additional strate-

[5] `http://www.voronkov.com/index.cgi`

[6] Specifically, for first-order *classical* logic as opposed to another of the Gandalf provers which is a prover for first-order *intuitionistic* logic.

gies include tautology deletion, full forward subsumption, back subsumption limited to already used clauses, paramodulation with ordering of equality and full forward demodulation (Tammet 1997).

Gandalf has participated in CASC since 1995 and was the winner of the MIX division for two years, and the winner of the SAT division for the following four years. Several versions of the prover can be downloaded at `http://deepthought.ttu.ee/it/gandalf/`.

5.5 Rodin Provers

The Rodin tool suite installation includes only the NewPP prover. All other provers are third-party plug-ins from providers such as ClearSy[7] and Systerel[8]. The Rodin theorem proving environment can therefore include any theorem prover for which a suitable plug-in has been created.

With the exception of NewPP, all the provers discussed in this section were provided by third-party plug-in providers. PP and ML are part of the Atelier B tool suite developed by ClearSy, and the SMT prover plug-in (CVC3, CVC4, veriT and Z3) is a joint work between Systerel and *Universidade Federal do Rio Grande do Norte*. The model checker ProB is provided by the HHU Düsseldorf STUPS Group. All these provers (except NewPP) were used in the experimental work reported on in subsequent chapters.

5.5.1 NewPP

NewPP applies a combination of unit resolution and the Davis-Putnam algorithm (see Section 2.3.2) to discharge proof obligations. Function and predicate symbols in the input that are different from '$\in$' and are not related to arithmetic, are "translated away" (Jastram 2012, p. 164). For example, $A \subseteq B$ is converted to

$$\forall x \cdot x \in A \Rightarrow x \in B$$

The NewPP prover has three configurations, namely *restricted*, *with lasso* and *unrestricted*. In the *restricted* configuration, selected hypotheses[9] along with the goal are passed to the prover.

The *with lasso* configuration first applies the *lasso* operation and then passes the selected hypotheses and the goal to NewPP. The *lasso* operation adds additional, unselected hypotheses with free identifiers common to the goal or the selected hypotheses to the list of selected hypotheses.

In the *unrestricted* configuration, all available hypotheses are passed to NewPP.

The NewPP prover was not used in our experimental work, because it is not currently considered sound, does not support arithmetic and is unaware of the set-theoretic axioms (Jastram 2012, p. 163).

[7] `http://www.clearsy.com/en/`

[8] `http://www.systerel.fr/`

[9] The Rodin theorem proving user interface allows the user to explicitly change the list of selected hypotheses for a pending proof tree node. See Figure B.2 in Appendix B.1.

5.5.2 PP

The PP has the same configurations as the NewPP prover, called “P0”, “P1” and “PP”. The prover proceeds by translating the input sequent to classical B. PP’s inference mechanism is similar to that of NewPP, but PP supports equational and arithmetic reasoning (Jastram 2012, p. 162).

5.5.3 ML

The ML prover accepts all visible hypotheses as input. ML is a term rewriter (see Section 3.4) and applies a mix of forward, backward and rewriting rules in order to either discharge the goal or detect a contradiction among the hypothesis. Finally, ML has limited support for equational and arithmetic reasoning, but does not contain all the set-theoretic axioms (Jastram 2012, p. 163).

5.5.4 ProB

ProB was originally developed as an animation and model checking tool for the B-method (Abrial, Butler, Hallerstede, Hoang, et al. 2010; Leuschel and Butler 2003, p. 855). The intended purpose was consistency checking of B machines via model checking. However, the prover also includes a constraint solver that attempts to find counter-examples for a given proof obligation. In the case of proof obligations that only include finite sets, the absence of counter-examples constitute a proof and ProB can therefore also be used as a prover (Krings, Bendisposto, and Leuschel 2015).

The user manual can be downloaded from `https://www3.hhu.de/stups/prob/index.php/User_Manual`.

5.5.5 SMT Solvers

Propositional satisfiability (SAT) is a constraint satisfaction problem for propositional logic. The problem consists of finding an assignment of true/false values to the variables in a formula such that the formula is true (i.e. the formula is satisfiable), or determining that no such assignment exists. Satisfiability is one of the NP-complete problems (Moskewicz et al. 2001, p. 530).

However, propositional logic cannot naturally express, for example, the languages of arithmetic or set theory (De Moura and Bjørner 2011, p. 69). Satisfiability Modulo Theories (SMT) is the area of automated deduction that combines propositional satisfiability checking with satisfiability checking of first-order formulas with respect to some logical theories. SMT generalises satisfiability by adding equality reasoning, arithmetic, bit-vectors, arrays, quantifiers and numerous other useful theories (De Moura and Bjørner 2008, p. 337).

SMT differs from general automated deduction, because the background theories need not be finitely or even first-order axiomatisable. Specialised inference methods are used for each theory. Typically, these spe-

cialised methods are more effective than general-purpose theorem provers (Barrett, Fontaine, and Tinelli 2015, p. 13).

A driving force in the development of SMT provers is the annual competitions CASC and SMT-COMP discussed in Sections 5.1.1 and 5.1.3 respectively. Another factor contributing to the continued improvement of SMT provers, is the common interchange formats of the TPTP and SMT-LIB problem libraries discussed in Sections 5.1.2 and 5.1.4 respectively.

A number of powerful SMT provers are available as plug-ins in Rodin, namely the CVC provers (CVC3 and CVC4), veriT and Z3. Next we provide a brief overview of each of these provers. Section 6.2.3 provides additional information on when these provers were made available for Rodin.

5.5.5.1 CVC provers

The Standford Validity Checker (SVC) was the first of the CVC family of provers and was first released in 1996. Its line of successors currently terminates with the SMT solvers CVC3 and CVC4 (*CVC4 - The SMT Solver* 2017). Both CVC3 and CVC4 are included in Systerel's SMT prover plug-in for Rodin.

5.5.5.1.1 SVC and CVC

The Standford Validity Checker (SVC) is the predecessor of the CVC provers CVC, CVCL, CVC3 and CVC4.

The Cooperating Validity Checker (CVC) is a SMT solver built around 2002 incorporating the Chaff SAT solver for propositional reasoning[10] (Stump, Barrett, and Dill 2002, p. 500-502). The name of the prover comes from the cooperation between decision procedures in Nelson-Oppen[11] theory combination, which relies on sharing equalities amongst decision procedures for common terms (Barrett, Conway, et al. 2011, p. 171).

CVC is a high-performance system for checking validity of formulas in a rich decidable logic. Atomic formulas are applications of predicate symbols like $\leqslant$ and $=$ to first-order terms like $1 + 2 * x$. Formulas are the usual boolean combinations (using $\wedge$, $\vee$, $\neg$, etc.) of atomic formulas (Stump, Barrett, and Dill 2002, p. 500-502).

A variant of the Nelson–Oppen approach was implemented in CVC to achieve cooperating, independent decision procedures for arrays, data types such as lists and trees, and linear real arithmetic (Stump, Barrett, and Dill 2002, p. 501).

5.5.5.1.2 CVCL

[10] For a discussion of the Chaff SAT solver, see Moskewicz et al. (2001).

[11] Nelson and Oppen (1979) showed that independent decision procedures for quantifier-free logical theories in first-order logic with equality can be combined to yield a decision procedure for the union of the theories. The theories are considered independent if they share no function and predicate symbols other than the equality symbol.

CVC Lite (CVCL) replaced the CVC solver in August 2003 (Barrett and Berezin 2004). The theories supported by CVCL included:

1. **Equality with uninterpreted functions**

   The theory includes an arbitrary number of predicate and function symbols without containing any further information about them.

2. **Arrays**

   The theory of abstract arrays supporting the operations *read* and *write*.

3. **Records and tuples**

   The theory supports aggregate data types like records and tuples with operations similar to the array operations to create, read from and write to these data types.

4. **Arithmetic**

   The theory of arithmetic was extended to also deal with arithmetic over integers (not just real numbers) as well as linear expressions of any combination of real and integer variables. Limited ability to simplify some nonlinear expressions was also added, e.g., simple identities like $(a + b)(a - b) = a^2 - b^2$ can be verified.

5. **Set theory**

   A decision procedure for a subset of set theory was in development in 2004.

Another significant new feature of CVCL was native support of quantifiers, which necessarily makes the logic undecidable. However, Barrett and Berezin (2004, p. 517) report that in many practical examples, simple heuristics for quantifier instantiation sufficed.

5.5.5.1.3 CVC3

In 2007, CVC3 replaced CVCL and was the first SMT-LIB compliant CVC prover (Barrett and Tinelli 2007, p. 298). Other than improving a number of theories, CVC3 treated quantified formulas as if they belonged to a quantifier theory. Treating quantified formulas as such allowed CVC3 to apply a special strategy: "[E]xistential formulas are skolemized and then passed back to the main theory solver for additional processing; universal formulas are accumulated and a set of heuristics is used to instantiate the formulas with ground terms from other literals known to the theory solver" (Barrett and Tinelli 2007, p. 300).

Furthermore, CVC3 contains a powerful translation module in order to support SMT-LIB. The module is capable of translating benchmarks to and from the SMT-LIB format. In fact, at the time, CVC3 became the standard for checking the syntax and categorization of new benchmarks submitted to the library (Barrett and Tinelli 2007, p. 300).

5.5.5.1.4 CVC4

CVC4 is the fifth and latest version of the Cooperating Validity Checker and replaced CVC3 in 2011. The CVC4 solver supports the useful feature set of CVC3 and SMT-LIBv2, but is a complete re-evaluation and rewrite of the CVC3 implementation to take advantage of engineering and algorithmic advances (Barrett, Conway, et al. 2011, p. 171).

No additional theories are supported, but CVC4 generally performs better than CVC3: in SMT-COMP 2010, both solvers competed in the QF_LRA[12] division and CVC4 solved more than twice as many benchmarks as CVC3. Also, CVC4 was almost always faster for benchmarks solved by both (Barrett, Conway, et al. 2011, p. 176).

5.5.5.2 veriT

The Rodin theorem prover extensions also include the SMT solver veriT, a joint effort of the University of Nancy (Nancy, France) and Federal University of Rio Grande do Norte (Natal, Brazil). veriT provides decision procedures for unquantified formulas over uninterpreted symbols, difference logic over integer and real numbers, and the combination of these procedures. Quantifier reasoning capabilities are supported through the integration of a first-order logic prover and quantifier instantiation (Bouton et al. 2009, p. 151).

A satisfiability solver produces models of the boolean abstraction of a formula and passes the propositional assignments to a *theory reasoner* responsible for checking if the assignments are models in the relevant background theories. The theory reasoner is a Nelson-Oppen combination of decision procedures with equality propagation (Bouton et al. 2009, p. 152).

5.5.5.3 Z3

Z3 is an efficient SMT solver from Microsoft Research. A Z3 prototype participated in SMT-COMP 2007 and won four first places, and seven second places. The success can be attributed to factors such as new algorithms for quantifier instantiation and theory combination. Z3 was first released externally in September 2007. The implementation of Z3 is discussed in (De Moura and Bjørner 2008).

5.6 Summary

In this chapter we introduced the theorem provers that were used to obtain our experimental results. We have also highlighted two automated theorem proving competitions — the CADE ATP System Competition (CASC) and the Satisfiability Modulo Theories Competition (SMT-COMP) — both because of their significance as cata-

[12] See Table 5.2 in Section 5.1.4: *QF_LRA* is an abbreviation of *Quantifier-Free, Linear Real Arithmetic*.

lysts for automated theorem prover research, and because many of the theorem provers discussed in this chapter have participated in one or both of these competitions. The resolution-based theorem provers OTTER, Vampire and Gandalf were introduced. Finally, we introduced the Rodin theorem provers NewPP, ML, PP, and the SMT provers CVC3, CVC4, veriT and Z3.

Chapter 6

Rodin/Event-B

In Chapter 5 we have introduced the theorem provers that were used in our experimental work. OTTER, Vampire and Gandalf are standalone theorem provers, but the remaining provers are part of the Rodin/Event-B[1] tool suite. The variety of inference mechanisms implemented by these provers motivated our selection of Rodin as a hybrid theorem proving environment.

In the current chapter, we provide an in-depth discussion of Rodin/Event-B, especially with a view to understanding the modelling language and theorem proving mechanism.

In Sections 6.1 and 6.2 we present an introduction to the tool suite and modelling language along with an overview of the historical development of Rodin/Event-B.

The Event-B modelling components and mathematical notation are discussed in Sections 6.3 and 6.4. The Event-B data types are introduced as well as the concept of well-definedness.

Section 6.5 describes the Event-B Core, i.e. the tool chain consisting of three major components: the static checker, the proof obligation generator and the proof obligation manager.

In Section 6.6 the Event-B language is presented in terms of proof rules and rewrite rules in order to understand the Event-B deductive mechanism. The various parts of the theorem proving environment responsible for discharging proof obligations are discussed in Sections 6.7 to 6.11.

Finally, the chapter is concluded with a summary in Section 6.12.

6.1 Overview of Rodin/Event-B

Rodin/Event-B refers to the combination of the formal method Event-B and the Rodin integrated development environment. Specifically, Rodin is a platform for formal modelling in Event-B.

The Rodin tool suite was designed to support the construction and verification of Event-B models (which can also serve as formal specifications) and refinement of these models towards implementation. Modelling,

[1] http://www.event-b.org/

refinement and proving are seamlessly integrated in Rodin (Abrial, Butler, Hallerstede, Hoang, et al. 2010). The Rodin platform is an open source, Eclipse-based[2] integrated development environment providing "core functionality for syntax analysis and proof-based verification of Event-B models" (Jastram 2012, p. 9).

Event-B is a state-based formal method for system-level modelling and analysis developed by Jean-Raymond Abrial. The basic mathematical theories of Event-B are first-order logic, typed set theory and integer arithmetic (Abrial 2010; Déharbe et al. 2012, p. 194).

Furthermore, Event-B is closely related to the B-method and Action Systems (Lemor Jr., Da Costa Cavalheiro, and Foss 2015, p. 193). In fact, Event-B is an evolution of the B-Method (also developed by Abrial in the early 1990s) to support reactive system development (Leuschel and Butler 2008, p. 185). Abrial (2010) describes Event-B as "a simplification as well as an extension of the B formalism". While the primary focus of the B-Method is the formal development of software, Event-B is intended for modelling and reasoning about systems that may include components other than software, e.g., physical components and electronics (Jastram 2012, p. 9).

6.2 The Development of Rodin/Event-B

Three research projects running between 2004 and 2014 provided much of the needed resources and impetus to drive the development of Rodin/Event-B. These projects, known as RODIN, DEPLOY and ADVANCE, are discussed next.

6.2.1 The RODIN Project

At the time of developing Event-B, Abrial was involved in an initiative to create an EU proposal on formal methods for dependable systems. The proposal became the RODIN (Rigorous Open Development Environment for Complex Systems)[3] project (2004-2007) and a primary objective was "the development of an open source extensible tool set to support refinement-based formal development" (Jastram 2012, p. 9). In the words of the project summary:

> "Our overall objective is the creation of a methodology and supporting open tool platform for the cost effective rigorous development of dependable complex software systems and services" (RODIN 2004).

Abrial's ideas on Event-B shaped the requirements for the tool. Development of the core tool platform was led by Abrial and Laurent Voisin. Eclipse was identified as a suitable platform for an open tool set, primarily

[2] http://www.eclipse.org/
[3] http://rodin.cs.ncl.ac.uk/

because of the ease with which the core functionality can be extended with plug-ins. The tools developed in the RODIN project retained the name Rodin after the project ended (Abrial 2010; Jastram 2012, p. 9).

6.2.2 The DEPLOY Project

The RODIN Project was followed by the DEPLOY[4,5] Project (2008-2012) focussing on *Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity*. The project plan (reflected by the name, DEPLOY) was to introduce Event-B into a number of preselected industrial organisations and ultimately to learn from the experience. The project ran for four years and involved fifteen partners from academia and industry. The project was coordinated by Alexander Romanovsky.

Industrial partners included, *inter alia*, Bosch[6], Siemens Transportation Systems[7] and Space Systems Finland[8]. Their diverse systems (i.e. automotive, space and railway systems) demanded a rich variety in Rodin/Event-B to cope with heterogeneous requirements. Hence, industrial application of Rodin/Event-B exposed difficulties, failures and shortcomings in coping with varying complexities of real-world, industrial systems (Romanovsky and M. Thomas 2013). Tool developers had to "implement significant improvements in performance, usability and stability of Rodin and key plug-ins" (Jastram 2012, p. 9).

6.2.3 The ADVANCE Project

A few months before the termination of the DEPLOY project, the ADVANCE[9] research project commenced. The ADVANCE project (2011-2014) had the overall objective of developing "a *unified* tool-based framework for automated formal verification and simulation-based validation of cyber-physical systems" (Colley 2011b) and was coordinated by the University of Southampton (Colley 2011a).

In order to meet the project goals, the decision was made to build on the existing formal modelling language Event-B and the associated tools environment Rodin:

> "Rodin will be further strengthened and augmented with novel approaches to multi-simulation and testing. Building on Event-B and Rodin will allow us to make considerable progress within the period of the ADVANCE project" (Colley 2011b, p. 2).

Leading European industrial partners collaborated with academic partners and technology providers to apply the ADVANCE tools and methods in upcoming projects and business areas. Again, the industrial and commercial experience provided valuable stimulus for technological advances. Systerel[10] and the Universities

[4] http://www.deploy-project.eu/
[5] DEPLOY is an European Commission Information and Communication Technologies FP7 project (*DEPLOY* 2008).
[6] http://www.bosch.com/
[7] https://www.siemens.com
[8] http://www.ssf.fi/
[9] http://www.advance-ict.eu/
[10] http://www.systerel.fr/

of Düsseldorf and Southampton led the development of new methods and tools for Rodin/Event-B. Industrial partners included ALSTOM Transport[11], Critical Software Technologies Ltd.[12] and Selex ES Ltd.[13] (Colley 2011c).

One of the ADVANCE project work packages (WP) had a direct impact on our experimental work, namely WP3. WP3 was aimed at extending the Rodin platform's usability in industrial applications, e.g., enhanced formal proof support. The decision to exploit off-the-shelf automated provers resulted in the creation of an SMT plug-in for Rodin (Colley 2011b, p. 17). Rodin Platform 3.1 was released in December 2014 with plug-in support for the SMT provers veriT and CVC3. Another update in January 2016 included the powerful SMT provers CVC4 and Z3.

The SMT plug-in is provided by Systerel and was developed jointly by Systerel and *Universidade Federal do Rio Grande do Norte*. Additional information is available on the "About Rodin Platform Features" description for the SMT Solvers Plug-in under the Rodin application's "Help/About Rodin Platform" menu option.

For a discussion of the integration of SMT provers into the Rodin theorem proving environment, refer to Déharbe et al. (2012).

6.3 Event-B Modelling Components

When developing formal models with the Rodin platform, a project is the most basic concept for capturing a discrete transition system (Abrial 2010). Projects can contain two different types of components, namely machines and contexts. The remainder of our discussion of Rodin/Event-B is mostly limited to Event-B contexts, because Event-B machines were not used in our experimental work.

Contexts describe the static part of a model (as opposed to machines that model the dynamic part of a model via variables changed by events). A context contains carrier sets, constants, axioms and extended contexts. When context *A* extends context *B*, all the constants and axioms of context *B* are included in context *A*.

Example 6.1 (Event-B Contexts)**.** Contexts `Base` and `Derived` are two Rodin contexts. Context `Derived` extends context `Base`. Note that comments appear as text in green.

CONTEXT Base
Context `Base` is extended by context `Derived`
SETS
set1 *set1* is a carrier set
CONSTANTS
cst1 *cst1* is a constant

[11] `http://www.alstom.com/`
[12] `http://www.criticalsoftware.com/`
[13] `http://www.us.selex-es.com/`

AXIOMS

axm1: $cst1 \in set1$

axm1 is an axiom

END

CONTEXT Derived

Context `Derived` extends context `Base`

EXTENDS Base

AXIOMS

thm1: $\langle$theorem$\rangle$ $set1 \neq \varnothing$

thm1 is a theorem

END

■

Constants are declared by adding a unique identifier to the *Constants* section. The declaration of a constant is only complete once the type of the constant can be inferred from one of the axioms.

Axioms are added to the *Axioms* section and consist of label-predicate pairs where the label (name) must be unique. Axioms are assumed to be true in the rest of the model. Any free identifiers appearing in an axiom's predicate must be a constant. A screen-shot of Rodin's context editor is included in Appendix B, Figure B.1.

When an axiom is marked as a theorem, a proof obligation (PO) is automatically generated and the validity of the theorem can be proven from previously declared axioms. These proof obligations appear as THM nodes in the *Event-B Explorer* (see Figure B.2 in Appendix B.1).

Formally, let A_{thm} be a theorem and A_b be the conjunction of all axioms (including axioms in extended contexts) declared before A_{thm}. Then A_{thm} is defined as the goal $A_b \Rightarrow A_{thm}$ in Table 6.1.

	An axiom as theorem
Name	label/THM
Goal	$A_b \Rightarrow A_{thm}$

Table 6.1: Event-B Definition of A_{thm} in a Rodin Context (Abrial 2010)

Finally, when an axiom contains a well-definedness condition[14], a well-definedness PO is generated automatically - refer to Jastram (2012, p. 126-7) for a detailed discussion. Formally, let A_w be a theorem and A_b be the conjunction of all axioms (including axioms in extended contexts) declared before A_w. Then the well-definedness condition of A_w is the goal $A_b \Rightarrow \mathcal{L}(A_w)$ in Table 6.2.

[14]The mathematical notation of the well-definedness conditions of each operator are defined by the $\mathcal{L}$-operator. See Section 6.4.2.

	Well-definedness of an axiom
Name	label/WD
Goal	$A_b \Rightarrow \mathcal{L}(A_w)$

Table 6.2: Event-B Definition of A_w in a Rodin Context (Abrial 2010)

6.4 Mathematical Notation

In this section we introduce the Event-B data types and expand on the concept of well-definedness. For a detailed discussion of the Event-B mathematical notation, refer to (Jastram 2012, Section 3.3), and a detailed discussion of the Event-B set-theoretic language can be found in (Abrial 2010, Section 9.5).

6.4.1 Data Types

There are only 3 basic data types in the Event-B language:

1. $\mathbb{Z}$ is the set of all integers;
2. BOOL $= \{$TRUE, FALSE$\}$ is the set of boolean values; and
3. Carrier sets are user defined types declared in the *Sets* section of a context and are never empty. No other assumptions are made about carrier sets unless explicitly stated as an axiom.

From these basic data types, additional types can be constructed. Let α and β be any of the three basic data types. Then the additional types are:

1. $\mathbb{P}(\alpha)$ is the *power set* of α, i.e. the set of subsets of α; and
2. $\alpha \times \beta$ is the *Cartesian product* of α and β, i.e. $x \mapsto y \in \alpha \times \beta$ implies x is of type α and y is of type β.

The discussion of the Event-B data types is concluded with a number of examples to illustrate how the types of various Event-B constructs are defined in terms of power sets and Cartesian products. The symbol $\in$ is the Event-B *type of* operator. Finally, the examples below are given in terms of the set of integers, $\mathbb{Z}$, but each example is valid if $\mathbb{Z}$ is replaced by subsets of $\mathbb{Z}$ or carrier sets.

Example 6.2 (Hierarchy of Types)**.** The power set operator yields (in theory) an infinite hierarchy of types:

$S \in \mathbb{Z} \Rightarrow S \in \mathbb{Z}$

$S \in \mathbb{P}(\mathbb{Z}) \Rightarrow S \in \mathbb{P}(\mathbb{Z})$

$S \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \Rightarrow S \in \mathbb{P}(\mathbb{P}(\mathbb{Z}))$

$\ldots$

$S \in \mathbb{P}^n(\mathbb{Z}) \Rightarrow S \in \mathbb{P}^n(\mathbb{Z})$ ■

Example 6.3 (Cartesian Product)**.** Cartesian products are necessary to create sets of tuples:

$S = \{x, y \cdot x \in \mathbb{Z} \land y \in \mathbb{Z} \mid x \mapsto y\} \Rightarrow S \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z})$

$S = \{x, y, z \cdot x \in \mathbb{Z} \land y \in \mathbb{Z} \land z \in \mathbb{Z} \mid x \mapsto y \mapsto z\} \Rightarrow S \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z})$ ■

Example 6.4 (Relations and Functions)**.** The combination of the power set operator and Cartesian products yield the relational and functional types. Let $\phi \in \{\leftrightarrow, \nleftrightarrow, \leftrightarrow\!\!\!\!\!\rightarrow, \nleftrightarrow\!\!\!\!\!\rightarrow, \pfun, \fun, \rightarrowtail, \twoheadrightarrow, \twoheadrightarrow\!\!\!\!\!\!\rightarrowtail, \fun\!\!\!\!\!\!\rightarrowtail\}$, then:

$S \in \mathbb{Z}\ \phi\ \mathbb{Z} \Rightarrow S \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z})$

$S \subseteq \mathbb{Z}\ \phi\ \mathbb{Z} \Rightarrow S \in \mathbb{P}(\mathbb{P}(\mathbb{Z} \times \mathbb{Z}))$ ■

6.4.2 Well-definedness

Well-definedness conditions for the safe evaluation of expressions and predicates are captured in Rodin by a predicate given by the $\mathcal{L}$-operator. The $\mathcal{L}$-operator is used to generate POs when expressions and predicates have well-definedness conditions.

Example 6.5. Consider the expression x/y which can only be evaluated when $y \neq 0$. In terms of the $\mathcal{L}$-operator, we have $\mathcal{L}(x/y) \mathrel{\widehat{=}} y \neq 0$, i.e. the well-definedness predicate of x/y is logically equivalent to $y \neq 0$. ■

If the well-definedness predicate is included in an axiom, no well-definedness PO is generated. For example, for the well-defined axiom $y \neq 0 \land x/y \geq 2$, no well-definedness PO is generated. But if the axiom contains only $x/y \geq 2$, the well-definedness PO $y \neq 0$ is generated.

It is also interesting to note that in terms of well-definedness, the operators $\land$ and $\lor$ are no longer commutative. In the example above, the axiom $y \neq 0 \land x/y \geq 2$ is well-defined, but $x/y \geq 2 \land y \neq 0$ is not and still has the well-definedness condition $y \neq 0$. This is a consequence of the $\mathcal{L}$-operator definitions for $\land$ and $\lor$:

$$\mathcal{L}(P \land Q) \mathrel{\widehat{=}} \mathcal{L}(P) \land (P \Rightarrow \mathcal{L}(Q))$$

$$\mathcal{L}(P \lor Q) \mathrel{\widehat{=}} \mathcal{L}(P) \land (P \lor \mathcal{L}(Q))$$

6.5 The Event-B Core

The Event-B core tool chain consists of three major components: the static checker, the proof obligation generator and the proof obligation manager (Abrial, Butler, Hallerstede, Hoang, et al. 2010, p. 455). Figure 6.1 shows the relationship between these components, and the Rodin builder handles the scheduling of these components' activities (Abrial, Butler, Hallerstede, and Voisin 2006, p. 596).

Rodin's graphical user interface consists of two parts: the modelling user interface (MUI) and the proving user interface (PUI). Figure 6.2 illustrates the integration of the core tool chain and the user interfaces. The two

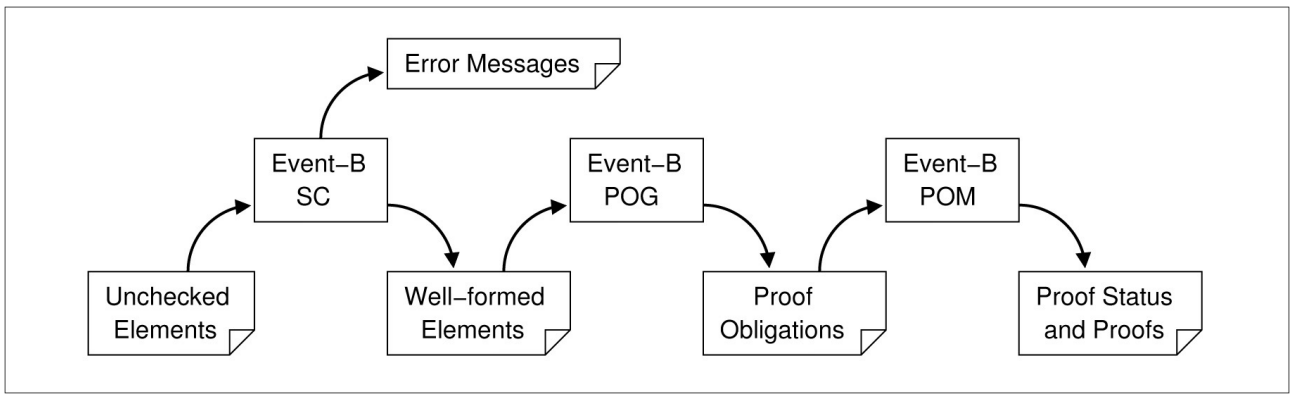


Figure 6.1: The Event-B Core Tool Chain from Abrial, Butler, Hallerstede, Hoang, et al. (2010, p. 456)

interfaces are connected via the core tool chain and are presented to the user as Eclipse perspectives (Abrial, Butler, Hallerstede, Hoang, et al. 2010, p. 455–456).

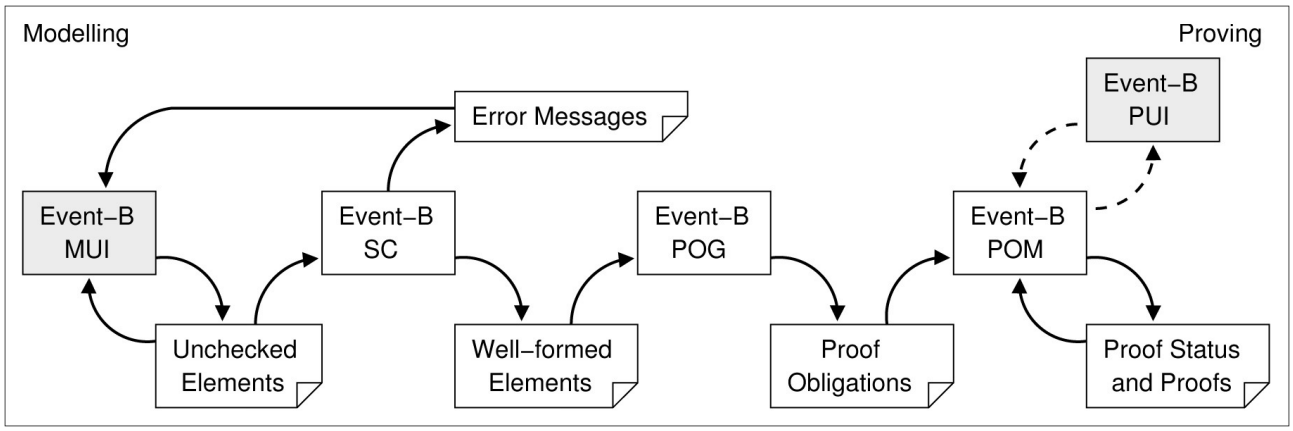


Figure 6.2: The Rodin User Interface from Abrial, Butler, Hallerstede, Hoang, et al. (2010, p. 456)

Screen shots of the user interfaces are included in Appendix B: Figures B.1 and B.2 show the modelling and proving user interfaces respectively.

6.5.1 The Static Checker

The static checker (SC) analyses Event-B contexts and machines, and provides the user with syntactic and typing error information. Event-B's mathematical notation is specified by a context-free grammar and the SC rejects elements in the user input that do not satisfy the grammar.

6.5.2 The Proof Obligation Generator

The proof obligation generator (POG) generates POs such as feasibility, event consistency, refinement and convergence POs for machines. The POG is also responsible for generating well-definedness (WD) POs.

6.5.3 The Proof Obligation Manager

The proof obligation manager (POM) keeps track of POs, their associated proofs and the proof status (e.g., discharged, reviewed or pending). Since proofs for a PO can be built automatically or interactively, the POM needs to work both as a part of the tool chain and with the proving user interface (UI).

The POM is divided into two parts, namely an *extensible* and a *static* part. The extensible part applies *reasoners* to generate individual proof rules (represented as *sequents*) which are used for proving POs (see Section 6.6 below). The static part collates proof rules to construct and maintain proofs.

Finally, the POM provides *tactics* (see Sections 6.9 and 6.11 below) to encapsulate frequently used proof construction and manipulation steps. The POM can be extended by adding new tactics and reasoners (Abrial, Butler, Hallerstede, Hoang, et al. 2010, p. 457).

6.6 Proof and Rewrite Rules

A sequent is a formal statement describing a theorem we want to prove. Sequents are of the form:

$$\mathsf{H} \vdash G,$$

where H is a set of predicates (the hypotheses) and G is a single predicate (the goal or theorem) that we want to prove from the hypotheses. $\mathsf{H} \vdash G$ is read as: Goal G holds under the hypotheses H, or: H entails G (Abrial 2010, p. 309).

A proof rule is, in mathematical terms, a tool to perform a formal proof. Proof rules are denoted by:

$$\boxed{\frac{A}{C}}$$

where A is a (possibly empty) list of sequents: the antecedents of the proof rule; and C is a sequent: the consequent of the rule. The proof rule can be interpreted as follows: the proof of each of the sequents in A together give a proof of the sequent C (Abrial, Butler, Hallerstede, Hoang, et al. 2010, p. 457).

The remainder of our discussion in this section is based on Chapter 9 in (Abrial 2010, p. 306–334). An up-to-date list of rewrite and inference rules are available online at `http://wiki.event-b.org/index.php/All_Rewrite_Rules` and `http://wiki.event-b.org/index.php/Inference_Rules` respectively. These pages also describe how each rule is used by default, i.e. automatically or manually (Abrial, Butler, Hallerstede, Hoang, et al. 2010, p. 457).

6.6.1 The Event-B Sequent Calculus

The first three basic proof rules are **HYP**, **MON** and **CUT**. Their definitions are (Abrial 2010, p. 309):

- **HYP**: If the goal P of a sequent belongs to the set of hypotheses of this sequent, then P is proved:

$$\frac{}{\mathsf{H}, P \vdash P}\ \textbf{HYP}$$

- **MON**: In order to prove a goal, it is sufficient to prove another sequent with the same goal, but with fewer hypotheses:

$$\frac{\mathsf{H} \vdash Q}{\mathsf{H}, P \vdash Q}\ \textbf{MON}$$

- **CUT**: If P holds under the set of hypotheses H, then P can be added to the set of hypotheses H for proving the goal Q:

$$\frac{\mathsf{H} \vdash P \qquad \mathsf{H}, P \vdash Q}{\mathsf{H} \vdash Q}\ \textbf{CUT}$$

We note that H, P and Q are *meta-variables*. In other words, H is a meta-variable representing any finite set of predicates, and P and Q are meta-variables for arbitrary predicates. The proof rules should therefore not be regarded as individual rules, but rather *rule schemas*. This also applies to the remainder of the discussion in Sections 6.6.1 to 6.6.4.

6.6.1.1 The Propositional and Predicate Language

The Event-B propositional language is built around five constructs: *falsity* ($\bot$), *negation*, *conjunction*, *disjunction* and *implication*. Two additional operators are defined in terms of rewrite rules (Abrial 2010, p. 315):

Predicate	**Rewritten**
$\top$	$\neg \bot$
$P \Leftrightarrow Q$	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$

For each type of predicate, two rules are given: a left rule (indicated by _**L**) and a right rule (indicated by _**R**). The left rule corresponds to the predicate appearing in the hypotheses and the right rule corresponds to the predicate appearing in the goal of the sequent. The proof rules are (Abrial 2010, p. 311):

$$\frac{}{\mathsf{H}, \bot \vdash P}\ \textbf{FALSE\_L} \qquad \frac{\mathsf{H} \vdash P \quad \mathsf{H} \vdash \neg P}{\mathsf{H} \vdash \bot}\ \textbf{FALSE\_R}$$

$$\frac{\mathsf{H}, \neg Q \vdash P}{\mathsf{H}, \neg P \vdash Q}\ \textbf{NOT\_L} \qquad \frac{\mathsf{H}, P \vdash \bot}{\mathsf{H} \vdash \neg P}\ \textbf{NOT\_R}$$

$$\frac{\mathsf{H}, P, Q \vdash R}{\mathsf{H}, P \wedge Q \vdash R}\ \textbf{AND\_L} \qquad \frac{\mathsf{H} \vdash P \quad \mathsf{H} \vdash Q}{\mathsf{H} \vdash P \wedge Q}\ \textbf{AND\_R}$$

$$\frac{\mathsf{H}, P \vdash R \quad \mathsf{H}, Q \vdash R}{\mathsf{H}, P \vee Q \vdash R}\ \textbf{OR\_L} \qquad \frac{\mathsf{H}, \neg P \vdash Q}{\mathsf{H} \vdash P \vee Q}\ \textbf{OR\_R}$$

$$\frac{\mathsf{H}, P, Q \vdash R}{\mathsf{H}, P, P \Rightarrow Q \vdash R}\ \textbf{IMP\_L} \qquad \frac{\mathsf{H}, P \vdash Q}{\mathsf{H} \vdash P \Rightarrow Q}\ \textbf{IMP\_R}$$

$$\frac{\mathsf{H} \vdash P}{\mathsf{H}, \top \vdash P}\ \textbf{TRUE\_L} \qquad \frac{}{\mathsf{H} \vdash \top}\ \textbf{TRUE\_R}$$

Quantified predicates require additional rules of inference. The rules for universally quantified predicates are[15] (Abrial 2010, p. 317):

$$\frac{\mathsf{H}, \forall x \cdot P, [x := E]P \vdash Q}{\mathsf{H}, \forall x \cdot P \vdash Q}\ \textbf{ALL\_L} \qquad \frac{\mathsf{H} \vdash P}{\mathsf{H} \vdash \forall x \cdot P}\ \textbf{ALL\_R} \text{ (x nfin } \mathsf{H})$$

[15] *nfin* is short for *does not appear free in* or simply *not free in*.

The first rule (**ALL_L**) allows the addition of another predicate to the hypotheses when the hypotheses already contain a universally quantified assumption. The additional predicate is obtained by instantiation of the quantified variable x in the predicate P by the expression E (this is denoted by $[x := E]P$).

The second rule (**ALL_R**) allows the removal of a universal quantifier in the goal when the quantified variable x does not appear free in the set of assumptions H.

The rules for existentially quantified predicates are (Abrial 2010, p. 317):

$$\frac{\mathsf{H}, P \vdash Q}{\mathsf{H}, \exists x \cdot P \vdash Q}\ \textbf{XST\_L}\ (\text{x nfin } \mathsf{H} \text{ and } Q) \qquad \frac{\mathsf{H} \vdash [x := E]P}{\mathsf{H} \vdash \exists x \cdot P}\ \textbf{XST\_R}$$

6.6.2 Equality

The equality predicate is simply given by $E = F$ for the expressions E and F. The inference rules for equality are (Abrial 2010, p. 320):

$$\frac{[x := F]\mathsf{H}, E = F \vdash [x := F]P}{[x := E]\mathsf{H}, E = F \vdash [x := E]P}\ \textbf{EQ\_LR}$$

$$\frac{[x := E]\mathsf{H}, E = F \vdash [x := E]P}{[x := F]\mathsf{H}, E = F \vdash [x := F]P}\ \textbf{EQ\_RL}$$

EQ_L and **EQ_R** allow the application of an equality assumption in the remaining assumptions or in the goal. The application of these proof rules are discussed in Section 8.3.1.

The reflexivity of equality and the equality of pairs are defined as rewrite rules (Abrial 2010, p. 320):

Operator	**Predicate**	**Rewritten**
Equality	$E = E$	$\top$
Equality of pairs	$E \mapsto F = G \mapsto H$	$E = G \wedge F = H$

Finally, the *one point rules* are provided as two additional rewrite rules (Abrial 2010, p. 320):

Predicate	Rewritten
$\forall x \cdot x = E \Rightarrow P$	$[x := E]P$
$\exists x \cdot x = E \wedge P$	$[x := E]P$

6.6.3 The Set-theoretic Language

Some expressions in Event-B are sets. When the expressions E and S are sets, then $E \in S$ is the standard *membership* predicate.

The axioms of set theory are defined in terms of rewrite rules under the form of equivalences of various set memberships. Syntactic support is provided for a large number of set-theoretic operators. The rewrite rules for operators used in our experimental work are included here for ease of reference (Abrial 2010, p. 321-9).

The rewrite rules for the basic set operators are (Abrial 2010, p. 322):

Operator	Predicate	Rewritten	Side cond.
Cartesian product	$E \mapsto F \in S \times T$	$E \in S \wedge F \in T$	
Power set	$E \in \mathbb{P}(S)$	$\forall x \cdot x \in E \Rightarrow x \in S$	x nfin E, x nfin S
Set comprehension	$E \in \{x \cdot P \mid F\}$	$\exists x \cdot P \wedge E = F$	x nfin E
Set equality	$S = T$	$S \in \mathbb{P}(T) \wedge T \in \mathbb{P}(S)$	

We note that the last rule (Set equality) is the Event-B implementation of the Axiom of Extensionality (see Section 4.2).

The classical set operators are (Abrial 2010, p. 323):

Operator	Predicate	Rewritten
Inclusion	$S \subseteq T$	$S \in \mathbb{P}(T)$
Union	$E \in S \cup T$	$E \in S \ \vee \ E \in T$
Intersection	$E \in S \cap T$	$E \in S \ \wedge \ E \in T$
Difference	$E \in S \setminus T$	$E \in S \ \wedge \ \neg(E \in T)$
Set extension	$E \in \{a, \ldots, b\}$	$E = a \ \vee \ \ldots \ \vee \ E = b$

Empty set	$E \in \varnothing$	$\bot$

The rewrite rules for generalised union and intersection are (Abrial 2010, p. 324):

Operator	**Predicate**	**Rewritten**	**Side cond.**
Generalised union	$E \in \mathrm{union}(S)$	$\exists s \cdot s \in S \land E \in s$	s nfin E, s nfin S
Generalised intersection	$E \in \mathrm{inter}(S)$	$\forall s \cdot s \in S \Rightarrow E \in s$	s nfin E, s nfin S

The generalised intersection rewrite rule requires $\mathrm{inter}(S)$ to be well defined, the well-definedness condition being $S \neq \varnothing$.

The rewrite rules for binary relation operators are (Abrial 2010, p. 325):

Operator	**Predicate**	**Rewritten**	**Side cond.**
Domain	$E \in \mathrm{dom}(r)$	$\exists y \cdot E \mapsto y \in r$	y nfin E, y nfin r
Range	$F \in \mathrm{ran}(r)$	$\exists x \cdot x \mapsto F \in r$	x nfin F, x nfin r
Set of all binary relations	$r \in S \leftrightarrow T$	$r \subseteq S \times T$	
Set of all total relations	$r \in S \nleftrightarrow T$	$r \in S \leftrightarrow T \land \mathrm{dom}(r) = S$	
Set of all surjective relations	$r \in S \leftrightarrow\!\!\!\!\rightarrow T$	$r \in S \leftrightarrow T \land \mathrm{ran}(r) = T$	
Set of all total and surjective relations	$r \in S \nleftrightarrow\!\!\!\!\rightarrow T$	$r \in S \nleftrightarrow T \land r \in S \leftrightarrow\!\!\!\!\rightarrow T$	
Converse	$E \mapsto F \in r^{-1}$	$F \mapsto E \in r$	
Relational image	$F \in r[U]$	$\exists x \cdot x \in U \land x \mapsto F \in r$	x nfin F, x nfin r, x nfin U

Forward composition	$E \mapsto F \in f \,;\, g$	$\exists x \cdot E \mapsto x \in f \land x \mapsto F \in g$	x nfin E, x nfin F, x nfin f, x nfin g
Backward composition	$E \mapsto F \in g \circ f$	$F \mapsto E \in f \,;\, g$	

The rewrite rules for functional operators are (Abrial 2010, p. 329):

Operator	**Predicate**	**Rewritten**
Identity	$E \mapsto F \in \mathrm{id}$	$E = F$
Set of all partial functions	$f \in S \nrightarrow T$	$f \in S \leftrightarrow T \land (f^{-1} \,;\, f) \subseteq \mathrm{id}$
Set of all total functions	$f \in S \rightarrow T$	$f \in S \nrightarrow T \land \mathrm{dom}(f) = S$
Set of all partial injections	$f \in S \rightarrowtail T$	$f \in S \nrightarrow T \land f^{-1} \in T \nrightarrow S$
Set of all total injections	$f \in S \rightarrowtail T$	$f \in S \rightarrow T \land f^{-1} \in T \nrightarrow S$
Set of all partial surjections	$f \in S \twoheadrightarrow T$	$f \in S \nrightarrow T \land \mathrm{ran}(f) = T$
Set of all total surjections	$f \in S \twoheadrightarrow T$	$f \in S \rightarrow T \land \mathrm{ran}(f) = T$
Set of all bijections	$f \in S \rightarrowtail\!\!\!\!\twoheadrightarrow T$	$f \in S \rightarrowtail T \land f \in S \twoheadrightarrow T$

Finally, the rewrite rules for binary relations on numbers are (Abrial 2010, p. 333):

Operator	**Predicate**	**Rewritten**	**Well-definedness condition**
Less than or equal to	$a \leq b$	$\exists c \cdot c \in \mathbb{N} \land b = a + c$	
Less than	$a < b$	$a \leq b \land a \neq b$	
Greater than or equal to	$a \geq b$	$\neg (a < b)$	
Greater than	$a > b$	$\neg (a \leq b)$	
Interval	$c \in a \,..\, b$	$a \leq c \land c \leq b$	
Subtraction	$c = a - b$	$a = b + c$	

Division	$c = a/b$	$\exists r \cdot (r \in \mathbb{N} \land r < b \land a = c * b + r)$	$b \neq 0$
Modulo	$r = a \bmod b$	$a = (a/b) * b + r$	$0 \leq a \land b > 0$

6.6.4 The Boolean and Arithmetic Language

The Peano axioms and boolean expressions are defined by the following predicates (Abrial 2010, p. 332):

$$\text{BOOL} = \{\text{TRUE}, \text{FALSE}\}$$

$$\text{TRUE} \neq \text{FALSE}$$

$$0 \in \mathbb{N}$$

$$\text{succ} \in \mathbb{Z} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{Z}$$

$$\text{pred} = \text{succ}^{-1}$$

$$\forall S \cdot 0 \in S \land (\forall n \cdot n \in S \Rightarrow \text{succ}(n) \in S) \Rightarrow \mathbb{N} \subseteq S$$

$$\forall a \cdot a + 0 = a$$

$$\forall a \cdot a * 0 = 0$$

$$\forall a \cdot a \hat{} 0 = \text{succ}(0)$$

$$\forall a, b \cdot a + \text{succ}(b) = \text{succ}(a + b)$$

$$\forall a, b \cdot a * \text{succ}(b) = a * b + a$$

$$\forall a, b \cdot a \hat{} \text{succ}(b) = a \hat{} b * a$$

6.7 Reasoners

Reasoners generate proof rules. The input to a reasoner typically consists of a sequent and the reasoner is successfully applied when it can generate a proof rule that is applicable to the input sequent. The proof rule itself is the output of the reasoner and is trusted by the proof obligation manager (POM). How the proof rule is generated is not visible to the POM and the only assumptions the POM makes about the reasoner are (Abrial, Butler, Hallerstede, Hoang, et al. 2010, p. 457):

1. The generated proof rule must be *logically valid* (i.e. can be derived) in the mathematical logic; and
2. The reasoner must *operate deterministically*, i.e. the reasoner must always generate the same proof rule given the same input.

Example 6.6. Simplifier The simplifier rule, **SIM**, derives new hypotheses and simplifies the goal of a sequent according to rewriting rules, such as $E = E \;==\; \top$ or $E + 0 \;==\; E$ (see Section 6.6 for more examples of rewriting rules).

The simplifier generates proof rules of the form:

$$\frac{H, H' \;\vdash\; G'}{H \;\vdash\; G} \quad \textbf{SIM}$$

Here, H' and G' are the rewritten forms of H and G. H' and G' are formed by repeatedly applying any rewriting rules until no more rules apply. When the **SIM** proof rule is applied, the goal changes from G to G' and the rewritten hypothesis H' is added. The original hypothesis H is preserved.

So, for the sequent

$$a = a \Rightarrow b = c \;\vdash\; b + 0 = c,$$

the **SIM** reasoner generates the following inference rule:

$$\frac{a = a \Rightarrow b = c,\; b = c \;\vdash\; b = c}{a = a \Rightarrow b = c \;\vdash\; b + 0 = c} \quad \textbf{SIM}$$

■

6.8 Proof Trees

Proof trees in Rodin are recursive data structures based on *proof tree nodes*. Each node consists of three components: a sequent, a proof rule and child nodes. Each proof obligation has a corresponding proof tree with a root node sequent representing the proof obligation (Abrial, Butler, Hallerstede, Hoang, et al. 2010, p. 458).

A proof tree node is *pending* when no rule is applied and *non-pending* otherwise. A proof tree is considered valid when its child nodes are valid. The validity of a tree node is defined recursively:

- A pending node's children must be null.
- When a node is not pending, its children must not be null. Also, the node's proof rule must be applicable to the node's associated sequent. The child nodes must correspond to the result of the application of the rule to the sequent.
- Finally, a node's children must all be valid.

The proof obligation manager (POM) provides a number of operations on proof trees:

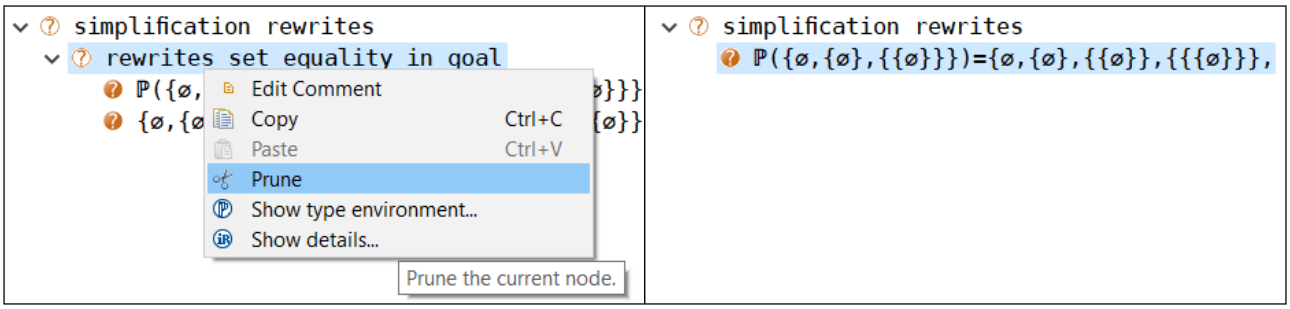
- **Rule application** A proof tree is expanded when a rule is applied to a pending node. The rule is checked for applicability to the node's sequent and if successful, the rule is attached to the node. Child nodes are generated and attached based on the outcome of the application of the rule.
- **Pruning** A proof tree can be pruned at any of its tree nodes via the proving UI. Both the rule and associated child nodes are removed from the pruned node.
- **Checking completeness** A proof tree is complete when it does not contain any pending proof tree nodes.

6.9 Tactics

Tactics provide a more convenient mechanism to construct and manipulate proofs. A tactic can act as a wrapper for an underlying reasoner and can call other tactics to modify proofs (Jastram 2012, p. 161). The list of all proof tactics is maintained on the Rodin Wiki (see `http://wiki.event-b.org/index.php/Rodin_Proof_Tactics`).

Tactics can be applied in a number of ways:

1. **Automatic**: Rodin can automatically apply the auto-tactic profiles (predefined lists of tactics) when an axiom is created or modified. The auto-tactic profiles can also be applied manually after an interactive proof step.
2. **Proof tree**: In the proving UI where the graphical proof tree is displayed, the pruning tactic can be applied to a proof tree node to remove the node and the node's child nodes (see Figure 6.3).
3. **Sequents**: The proving UI displays some sequent elements in red. When the user clicks on these elements, a menu with applicable tactics is displayed. When the user clicks on one of the menu options the tactic is applied automatically (see Figure 6.4).



(a) Before Pruning (b) After Pruning

Figure 6.3: Pruning a Proof Tree Node

A tactic is either *basic* or *tactical*. Basic tactics include:

- Pruning of proof tree nodes;

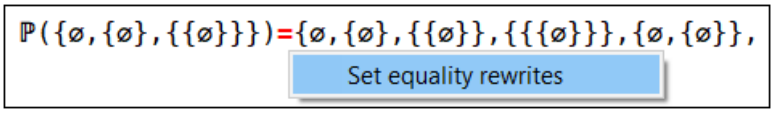


Figure 6.4: Interactive Tactics in the Proving UI

- Proof rule application; and
- Reasoner application.

Tactical tactics are constructed from existing tactics and are determined by strategic or heuristic decisions. Proof strategies are constructed by combining tactical tactics. Examples include:

- Apply on All Pending - The tactic will apply a list of sub-tactics to all pending nodes.
- Repeating - The tactic will repeat a list of sub-tactics to a pending node until application fails.
- Composing Sequential - The tactic applies a list of sub-tactics until one of the sub-tactics succeeds.

6.10 Provers

When a proof obligation is not discharged after applying proof and rewrite rules, a theorem prover is invoked. In Jastram's words, "In the end, provers perform the actual work" (Jastram 2012, p. 162).

The Rodin provers that were used in our experimental work were introduced in Section 5.5 where an overview of the inference mechanism of each prover was given. These are the Atelier B tool suite provers ML and PP, the model checker ProB, and the SMT provers CVC3, CVC4, veriT and Z3.

The Rodin proving UI supports the interactive application of these provers (see Section 8.2). Alternatively, provers can be included in the auto-tactics. Two auto-tactics are discussed in the next section: the default auto-tactic profile includes the proves ML and PP, and the default auto-tactic with SMT additionally includes all of the SMT provers.

6.11 Tactic Profiles

Tactic profiles provide a powerful mechanism for discharging proof obligations in Rodin. A tactic profile is an ordered list of combinators (e.g., control statements for looping) and tactics for rewrite rules, inference rules and provers. Instead of applying individual tactics to specific proof tree nodes, tactic profiles can be applied to all pending proof tree nodes repeatedly until some condition is satisfied.

The user can change the selected auto tactic profile or turn the automatic application of tactic profiles off altogether. A screen shot of the Auto/Post Tactic Selector is shown in Appendix B (Figure B.3).

Rodin includes an advanced option for constructing and editing additional tactic profiles, since the default tactic profiles cannot be modified. However, the construction of additional tactic profiles was not pursued in

our experimental work.

Tables 6.18 and 6.19 show the lists of combinators and tactics respectively that are available in Rodin for inclusion in bespoke tactic profiles. A screen shot of the Tactic Profile Editor is included in Appendix B (Figure B.4).

Attempt [1]
Attempt after Lasso [1]
Compose Until Failure [1 or more]
Compose Until Success [1 or more]
Loop [1]
Loop on All Pending [1 or more]
On All Pending [1]
Sequence [1 or more]

Table 6.18: Rodin Combinators

All enabled SMT configurations (Discharge)	For-all Goal (Simplify)
Atelier B ML (Discharge)	Functional Image (Simplify)
Atelier B P0 (Discharge)	Implicative Goal (Simplify)
Atelier B PP (Discharge)	Implicative Hypotheses with Conjunctive RHS (Simplify)
Belongs to domain (Discharge)	Implicative Hypotheses with Disjunctive LHS (Simplify)
Bounded Goal with finite Hypothesis (Discharge)	Partition Rewriter (Simplify)
Datatype Destructor WD (Discharge)	Remove disjunction in a disjunctive goal (Simplify)
False Hypothesis (Discharge)	Rewrite domain to Cartesian product in goal (Simplify)
Find Contradictory Hypotheses (Discharge)	Shrink Enumerated Set (Simplify)
Finite Goal (Discharge)	Shrink Implicative Hypotheses (Simplify)
Functional Goal (Discharge)	Simplification Rewriter (Simplify)
Functional image membership (Discharge)	Type Rewriter (Simplify)
Goal Disjunct in Hypotheses (Discharge)	Use Equals Hypotheses (Simplify)
Goal in Hypotheses (Discharge)	Conjunctive Goal (Split)
Membership in Goal (Discharge)	Functional Overriding in Goal (Split)
NewPP after lasso (Discharge)	Functional Overriding in Hypothesis (Split)
NewPP restricted (Discharge)	One Point Rule in Goal (Split)
NewPP unrestricted (Discharge)	One Point Rule in Hypotheses (Split)
Rewrites function application in domain (Discharge)	Put in Negation Normal Form (Split)
SMT Solver CVC3 (Discharge)	Remove maplet overriding relation (Split)
SMT Solver CVC4 (Discharge)	Lasso
SMT Solver veriT (Discharge)	ProB (Dis)Prover
SMT Solver Z3 (Discharge)	Remove all equivalences in goal
True Goal (Discharge)	Remove all equivalences in hypotheses
Clarify Goal (Mixed)	Remove all Membership/inclusion in goal
Generalized Modus Ponens (Mixed)	Remove all Membership/inclusion in hypotheses
Exists Hypotheses (Simplify)	Remove top Membership/inclusion in goal

Table 6.19: Rodin Tactics

The default installation of the Rodin tool suite includes two tactic profiles, namely the *auto tactic profile* and the *post tactic profile*. An additional tactic profile is available when the SMT provers (see Section 5.5.5) are installed. The default tactic profiles are shown in Table 6.20.

6.12 Summary

The aim of this work is to determine to what extent a *hybrid theorem proving environment* may benefit from the OTTER heuristics in the same way that OTTER benefited from these heuristics. We recall our earlier definition of a hybrid theorem proving environment, namely that at least two different inference mechanisms are applied while discharging proof obligations. This chapter introduced the hybrid theorem proving environment that we consider in this dissertation. The Rodin tool suite was selected as a suitable hybrid theorem proving environment, because Rodin is an Eclipse-based integrated development environment which can be extended with plug-ins. The tool suite can therefore include any theorem prover for which a suitable plug-in has been created. As of Rodin version 3, the following inference mechanisms were available via theorem prover extension plug-ins: SMT (CVC3, CVC4, veriT and Z3), model checking (ProB), term rewriting (ML) and resolution (PP, NewPP).

The Rodin tool suite was designed to support the construction and verification of models in the Event-B formal method. The current chapter provided an in-depth discussion of Rodin/Event-B starting with a brief overview of the historical development of the Rodin tool suite. The remainder of the chapter introduced the Event-B modelling language and discussed various aspects of Rodin's theorem proving mechanism.

In the next chapter we shall present our experimental results on the utility of the OTTER heuristics for discharging proof obligations in Rodin.

Auto Tactic Profile	**Auto Tactic Profile with SMT**	**Post Tactic Profile**
Loop on All Pending [1 or more] True Goal (Discharge) False Hypothesis (Discharge) Goal in Hypotheses (Discharge) Functional Goal (Discharge) Bounded Goal with finite Hypothesis (Discharge) Partition Rewriter (Simplify) Generalized Modus Ponens (Mixed) Simplification Rewriter (Simplify) Put in Negation Normal Form (Split) Type Rewriter (Simplify) Find Contradictory Hypotheses (Discharge) Finite Goal (Discharge) Shrink Implicative Hypotheses (Simplify) Functional Overriding in Goal (Split) Clarify Goal (Mixed) One Point Rule in Goal (Split) Functional Overriding in Hypothesis (Split) Functional Image (Simplify) One Point Rule in Hypotheses (Split) Use Equals Hypotheses (Simplify) Belongs to domain (Discharge) Functional image membership (Discharge) Atelier B ML (Discharge) Atelier B PO (Discharge) Datatype Destructor WD (Discharge)	**Loop on All Pending [1 or more]** True Goal (Discharge) False Hypothesis (Discharge) Goal in Hypotheses (Discharge) Functional Goal (Discharge) Bounded Goal with finite Hypothesis (Discharge) **Attempt after Lasso [1]** All enabled SMT configurations (Discharge) Partition Rewriter (Simplify) Generalized Modus Ponens (Mixed) Simplification Rewriter (Simplify) Put in Negation Normal Form (Split) Type Rewriter (Simplify) Find Contradictory Hypotheses (Discharge) Finite Goal (Discharge) Shrink Implicative Hypotheses (Simplify) Functional Overriding in Goal (Split) Clarify Goal (Mixed) One Point Rule in Goal (Split) Functional Overriding in Hypothesis (Split) Functional Image (Simplify) One Point Rule in Hypotheses (Split) Use Equals Hypotheses (Simplify) Belongs to domain (Discharge) Functional image membership (Discharge) Atelier B ML (Discharge) Atelier B P0 (Discharge) Datatype Destructor WD (Discharge)	**Loop on All Pending [1 or more]** True Goal (Discharge) False Hypothesis (Discharge) Goal in Hypotheses (Discharge) Goal Disjunct in Hypotheses (Discharge) Functional Goal (Discharge) Belongs to domain (Discharge) Functional image membership (Discharge) Bounded Goal with finite Hypothesis (Discharge) Datatype Destructor WD (Discharge) Generalized Modus Ponens (Mixed) Simplification Rewriter (Simplify) Put in Negation Normal Form (Split) Type Rewriter (Simplify) Exists Hypotheses (Simplify) Find Contradictory Hypotheses (Discharge) Use Equals Hypotheses (Simplify) Shrink Implicative Hypotheses (Simplify) Shrink Enumerated Set (Simplify) Functional Overriding in Goal (Split) Clarify Goal (Mixed)

Table 6.20: Rodin Auto Tactics

Chapter 7

Resolution-Based Reasoning Heuristics

In Chapter 5 we have discussed the inference mechanisms of the theorem provers used in the experimental work we report on in the current and the following chapters. We have also discussed the inference mechanism of the Rodin tool suite in Chapter 6, and we have noted how additional theorem provers can be plugged in to augment Rodin's theorem proving capabilities.

In the current chapter we discuss our evaluation of the reasoning heuristics developed by Labuschagne and Van der Poll (1999) and Van der Poll (2000) to aid the resolution-based automated reasoner OTTER. In Section 7.1 we discuss the need for heuristics such as the OTTER heuristics and the applicability of these heuristics to other theorem provers. The section also outlines our approach to evaluating the applicability of the OTTER heuristics to discharging proof obligations in Rodin. Sections 7.2 to 7.11 introduce the OTTER heuristics. Each section has the following general structure:

- One of the OTTER heuristics is introduced and explained;
- The problem used to illustrate the heuristic in (Van der Poll 2000) is discussed;
- OTTER's initial attempt to discharge the resulting proof obligation is presented;
- The heuristic is applied to restructure the proof;
- OTTER's subsequent attempt to discharge the modified proof obligation is discussed;
- The applicability of the heuristic to Vampire and Gandalf is discussed;
- The applicability of the heuristic to Rodin is discussed.

The chapter is concluded with a summary in Section 7.12.

7.1 The OTTER Heuristics

Fourteen search guiding and input restructuring heuristics were developed for the resolution-based theorem prover OTTER. Van der Poll (2000) demonstrated the value of these heuristics for discharging set-theoretic

proof obligations arising from formal specification development in the specification language Z (Spivey 1998).

However, set-theoretic proofs pose demanding challenges, because set theory is inherently hierarchical: the inclusions $a \in A$, $A \in \mathbb{P}(A)$, $\mathbb{P}(A) \in \mathbb{P}(\mathbb{P}(A))$, ... continue indefinitely. For example, to show

$$A \subseteq B \Leftrightarrow \mathbb{P}(A) \subseteq \mathbb{P}(B),$$

a human (such as a mathematician or an experienced specification writer) will consider elements of $A, B, \mathbb{P}(A)$ and $\mathbb{P}(B)$, but not elements of $\mathbb{P}(\mathbb{P}(A)), \mathbb{P}(\mathbb{P}(\mathbb{P}(A))), \ldots$ Since this type of intuition is absent in an automated reasoner, heuristics are needed to prevent automated theorem provers from exploring the consequences of irrelevant information in search of a proof.

Steyn (2009) was able to apply the OTTER heuristics to the original (or in some cases, enlarged) problems in (Van der Poll 2000) using the next generation (in terms of OTTER) resolution-based provers Vampire (Riazanov and Voronkov 2002) and Gandalf (Tammet 1997). Steyn (2009) was able to evaluate eleven of the fourteen OTTER heuristics.

In terms of Vampire, Steyn (2009) concluded the following:

1. Of the eleven heuristics evaluated, Vampire needed a total of ten heuristics to discharge the original (or enlarged) problems in (Van der Poll 2000);
2. In some cases, the benefit of applying a specific heuristic could only be observed once the problem was enlarged, whence Steyn concluded that Vampire is a more powerful theorem prover than OTTER;
3. However, there still existed a need for heuristics such as the OTTER heuristics.

Despite performing better than OTTER in general, Gandalf was not able to prove some of the more complex problems; of the eleven heuristics evaluated, Gandalf needed nine. Again, despite advances in prover development, more powerful provers are still in need of search guiding heuristics.

Even though the OTTER heuristics were developed for resolution-based automated theorem proving, their set-theoretic nature lends itself to evaluation with other automated reasoners. Our experimental work is centred around the evaluation of the OTTER heuristics in a hybrid theorem proving environment, namely the Rodin tool suite. Some of the OTTER heuristics cannot be applied to Rodin proofs, because Rodin does not have a counterpart for various OTTER settings such as weighting or inference rule selection. Table 7.1 shows the list of OTTER heuristics that were evaluated and how the OTTER heuristics defined by Van der Poll (2000) map to the heuristics listed in this chapter[1].

All the problems in (Van der Poll 2000) that were used to illustrate the heuristics along with additional problems in (Steyn 2009) where attempted in Rodin. Execution times for the Rodin provers are not available, so

[1]For example, the first OTTER heuristic (*Heuristic #1*) was not applicable to our experimental work. Therefore, the second OTTER heuristics (*Heuristic #2*) is mapped to *Heuristic #1* in column 3 of Table 7.1. The heuristic name that appears in the third column of Table 7.1 is also the name used to refer to the heuristics in the the remainder of the current and subsequent chapters.

Heuristic Name	OTTER Heuristic	Maps to
Weight strategy	Heuristic #1	—
Equality vs. extensionality	Heuristic #2	Heuristic #1
Nested functors	Heuristic #3	Heuristic #2
Divide-and-Conquer	Heuristic #4	Heuristic #3
Exemplification	Heuristic #5	—
Multivariate functor	Heuristic #6	Heuristic #4
Intermediate structure	Heuristic #7	Heuristic #5
Element structure	Heuristic #8	Heuristic #6
Redundant information	Heuristic #9	Heuristic #7
Search-guiding	Heuristic #10	Heuristic #8
Inference rule selection	Heuristic #11	—
Set-of-support enlargement	Heuristic #12	—
Resonance	Heuristic #13	Heuristic #9
Tuple condense	Heuristic #14	Heuristic #10

Table 7.1: The OTTER Heuristics

exact execution times are not included for Rodin proofs. We have reported the *wall clock time* (measured in seconds) where possible. Execution times are included for other theorem provers where available to demonstrate the improvement in theorem prover *strength* over the years.

Rodin's *Default Auto-tactic Profile with SMT* (see Section 6.11) was used throughout. The provers ML and ProB are not included in the auto-tactic profile by default and were applied on an ad hoc basis when the auto-tactic profile could not discharge a proof obligation.

All proofs reported on in this dissertation were done on an Intel(R) Core(TM) I7-7500U processor with 8GB RAM, running at a clock speed of 2.7GHz. The operating system used was Windows 10 Home edition (64-bit operating system, x64-based processor). We have conducted our experimental work on version 3.3 of the Rodin platform, version 2.2.1.r16701 of the *Atelier B Provers* and version 1.4.0.8c9a179 of the *SMT Solvers Plug-in*.

7.2 Equality versus Extensionality

> **OTTER *Heuristic #1***: Use the principle of extensionality to replace set equality with the condition under which two sets are equal, i.e. when two sets contain the same elements.

7.2.1 OTTER

The first heuristic is aimed at input containing set-theoretic equality. The Axiom of Extensionality (see Section 4.2) allows us to replace an equality with a formula stating that two sets contain the same elements. For example, for $A = B$, we have

$$(\forall x)(x \in A \Leftrightarrow x \in B) \Rightarrow A = B.$$

We can therefore apply the principle of extensionality to replace $A = B$ by $(\forall x)(x \in A \Leftrightarrow x \in B)$ in our input.

An interesting extension of equality replacement is replacing a subset formula (see Section 4.3.1) with a formula about element containment. For example, by the definition of subset (Enderton 1977, p. 18), we have for sets A and B:

$$(\forall x)(x \in A \Rightarrow x \in B) \Rightarrow A \subseteq B.$$

Again, this shows that we can replace $A \subseteq B$ by $(\forall x)(x \in A \Rightarrow x \in B)$ in our input.

Consider the following example in Van der Poll (2000, p. 80)[2]:

$$\mathbb{P}(\{\{1\}\}) = \{\varnothing, \{\{1\}\}\}. \tag{7.1}$$

OTTER could not find a proof for (7.1) in 20 minutes (Van der Poll 2000, p. 81). However, if *heuristic #1* is applied to unfold (7.1) as

$$(\forall x)(x \in \mathbb{P}(\{\{1\}\}) \Leftrightarrow x \in \{\varnothing, \{\{1\}\}\}), \tag{7.2}$$

then OTTER finds a proof for (7.2) in 0.03 seconds.

7.2.2 Vampire and Gandalf

When Steyn (2009, p. 79) attempted these proofs with Vampire and Gandalf in 2009, both theorem provers were able to prove (7.1) directly. The application of *heuristic #1* did however allow both provers to discover a proof in a shorter amount of time.

Since the proof discovery times were already short (especially for Vampire), Steyn considered a more complex problem:

$$\bigcap\{\{1,2,3\},\{2,3,4\}\} = \{2,3\}. \tag{7.3}$$

Vampire was unable to find a proof of (7.3) within a reasonable amount of time without the application of *heuristic #1*, but found a proof in 0.4 seconds once *heuristic #1* was applied. Steyn concluded that the heuristic appeared to be especially useful for Vampire when solving more complex problems.

[2]Of the theorem provers under consideration, OTTER, Vampire and Gandalf do not accept input in set-theoretic notation. Refer to Section 4.11 for an explanation of how set-theoretic notation is converted to first-order logic input for these provers. In the remainder of the text we show input in set-theoretic notation unless otherwise stated.

7.2.3 Rodin

All the SMT provers rapidly generated proofs of (7.1). However, only ProB was capable of discharging a proof of the tautology (7.3) directly. None of the other Rodin provers could prove (7.3) directly unless the formula is rewritten as

$$A = \{1,2,3\} \wedge B = \{2,3,4\} \Rightarrow \bigcap\{A,B\} = \{2,3\} \tag{7.4}$$

i.e. by introducing the constants *A* and *B*. All the theorem provers were able to rapidly generate proofs of (7.4). This technique is discussed in Section 8.4 and was identified as one of the Rodin heuristics.

The Rodin provers did not need *heuristic #1*. When *heuristic #1* was applied to (7.3) to obtain the formula

$$(\forall x)(x \in \bigcap\{\{1,2,3\},\{2,3,4\}\} \Leftrightarrow x \in \{2,3\}), \tag{7.5}$$

only ProB was able to find a proof. The other Rodin provers could neither proof (7.5) directly, nor with interactive assistance.

Rodin contexts for these examples appear in Appendices C.10.4 and C.10.7. We note that all proof obligtions in Appendix C could be discharged without application of *heuristic #1*. Refer to the following Rodin contexts to see examples of more complicated proof obligations that could be discharged without the application of *heuristic #1*: C.3.1, C.10.1, C.10.2, C.10.3, C.10.11 or C.10.17.

7.3 Nested Functors

> **OTTER *Heuristic #2***: Avoid the use of nested function symbols (functors).

Formal modelling or formal specification writing that employs only constants and predicates would be extremely limited in terms of expressiveness (and probably infeasible on an industrial scale). So, in addition to constants and predicates, function symbols allow us to express complex thoughts succinctly and with great clarity. However, function symbols lend themselves to increased complexity (increased term depth) because they can be nested to any arbitrary depth. The next heuristic is therefore aimed at reducing this type of complexity in theorem prover input (Van der Poll 2000).

7.3.1 OTTER

Let us consider an example: The associativity of set-theoretic symmetric difference (see Section 4.3.5) is given by

$$(A + B) + C = A + (B + C). \tag{7.6}$$

When trying to establish (7.6) with OTTER, Van der Poll (2000, p. 82) first applies *heuristic #1* and reports that OTTER can find a proof of the logically equivalent formula

$$(\forall x)(x \in ((A + B) + C) \Leftrightarrow x \in (A + (B + C))) \tag{7.7}$$

in 4 minutes 3 seconds. However, when Skolem constants are introduced to unfold the nested functors, OTTER takes only 0.17 seconds to find a proof of

$$D = A + B \wedge E = D + C \wedge F = B + C \wedge G = A + F \Rightarrow (\forall x)(x \in E \Leftrightarrow x \in G). \tag{7.8}$$

7.3.2 Vampire and Gandalf

Vampire could generate a proof of (7.7) in 0.1 seconds, but was unable to find a proof of (7.6) within 30 minutes (i.e. without the application of *heuristic #1*). This is another illustration of the utility of *heuristic #1* for Vampire.

The effect of *heuristic #2* on Vampire was illustrated by rewriting (7.6) with Skolem constants and the set equality retained (i.e. *heuristic #1* was not applied), namely

$$D = A + B \wedge E = D + C \wedge F = B + C \wedge G = A + F \Rightarrow E = G. \tag{7.9}$$

This formula was proved in 0.5 seconds. Both *heuristic #1* and *heuristic #2* have a similar effect on Vampire, although *heuristic #2* resulted in a longer proof time.

Gandalf could generate a proof of (7.7) in 41 seconds and (7.8) in 5 minutes 43 seconds. In other words, the application of *heuristic #2* resulted in a longer proof time. In terms of the more complex problem, Gandalf could not generate a proof for (7.6) after 30 minutes, but could prove (7.9) after 5 minutes 44 seconds after the application of *heuristic #2*.

Heuristic #2 did not have the same effect on Gandalf's performance in both cases. Steyn concluded with the remark that "the nested functor heuristic #2 may have to be augmented in certain cases" (Steyn 2009, p. 82).

7.3.3 Rodin

Rodin's default auto-tactic cannot show (7.6) automatically, but when the PP prover is applied interactively the proof obligation is discharged. On the other hand, the use of Skolem constants allows Rodin (specifically, the SMT provers) to find a proof automatically. Since we value automatic proof discovery so highly, we conclude this section with the remark that *heuristic #2* does benefit the Rodin provers.

See Appendices C.9.19 and C.9.18 for these examples. Appendix C.9.18 shows the direct statement of (7.6) and C.9.19 shows a context using Skolem constants.

7.4 Divide-and-Conquer

OTTER *Heuristic #3*: Perform subproofs where possible:

- ***Heuristic #3.1***

  When proving a set-theoretic equality, perform two separate subset proofs;

- ***Heuristic #3.2***

  When proving a conjunction, perform a separate subproof for each conjunct; and

- ***Heuristic #3.3***

  When proving an "if and only if" formula, perform the "if" and "only if" proofs separately.

7.4.1 Simplifying a set equality proof

7.4.1.1 OTTER

Van der Poll (2000, p. 82) uses the following example as a problem that is difficult for OTTER to solve:

$$\mathbb{P}(\{0,1\}) = \{\varnothing, \{0\}, \{1\}, \{0,1\}\}. \tag{7.10}$$

OTTER cannot proof the equality directly within 30 minutes. Once *heuristic #1* is applied, OTTER finds a proof of

$$(\forall x)(x \in \mathbb{P}(\{0,1\}) \Leftrightarrow x \in \{\varnothing, \{0\}, \{1\}, \{0,1\}\}) \tag{7.11}$$

in 3 minutes 23 seconds. If (7.11) is split into two subproofs, namely

$$(\forall x)(x \in \mathbb{P}(\{0,1\}) \Rightarrow x \in \{\varnothing, \{0\}, \{1\}, \{0,1\}\}) \tag{7.12}$$

and

$$(\forall x)(x \in \{\varnothing, \{0\}, \{1\}, \{0,1\}\} \Rightarrow x \in \mathbb{P}(\{0,1\})), \tag{7.13}$$

OTTER finds a proof of (7.12) in 0.43 seconds and of (7.13) in 0.03 seconds.

7.4.1.2 Vampire and Gandalf

Vampire was also unable to find a proof of (7.10) within a 30 minute time limit (Steyn 2009, p. 83). However, (7.11), (7.12) and (7.13) are proved in 0.8, 0.3 and 0.1 seconds respectively. On a more complex problem such as

$$\mathbb{P}(\{0,1,2\}) = \{\varnothing, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}, \tag{7.14}$$

the effect of the heuristics were more pronounced: 8 minutes 40 seconds was needed to find a proof of (7.14) after applying *heuristic #1*, and 28 seconds and 2 seconds respectively to find proofs of the two subproofs of (7.14) after applying *heuristic #3.1*.

Gandalf could also not generate a proof of (7.10) after 30 minutes. However, having applied the extensionality heuristic, *heuristic #1*, a proof is generated in 1 minute 27 seconds. Further application of the Divide-and-Conquer *heuristic #3.1* did not appear to be useful, since Gandalf required 1 minute 36 seconds and 1 minute 14 seconds respectively to generate the subproofs.

On the more complex problem (7.14), Gandalf was again unable to generate a proof after 30 minutes, even after applying *heuristic #1*. The Divide-and-Conquer heuristic leads to some success in this case, since it enabled the prover to generate one of the two subproofs in 1 minute 15 seconds.

7.4.1.3 Rodin

In Rodin, (7.10) and (7.14) are shown easily. The Rodin context containing these examples appears in Appendix C.10.2.

Not withstanding these examples, *heuristic #3* is a useful heuristic in Rodin proofs and is one of the interactive steps available in the theorem proving environment. The interactive step is called *Set equality rewrites* in the Rodin proving UI and is discussed in Section 8.3.2. A number of the theorems in the Rodin contexts in Appendix C can only be proved after an initial application of *heuristic #3*. For example:

- Appendix C.9.18, *thm6*:
  $PLUS(PLUS(A \mapsto B) \mapsto C) = PLUS(A \mapsto PLUS(B \mapsto C))$
- Appendix C.10.16, *thm4*:
  $A \cap B \cap C = \{x \cdot x \in \mathbb{Z} \mid 2 * 2 * 3 * 3 * 5 * x\}$

7.4.2 Simplifying a conjunction

Although execution times were omitted, we mention another example from Van der Poll (2000, p. 83): Let f be a function to which we add an ordered pair (a,b) where $a \notin dom(f)$. Let $g = f \cup \{(a,b)\}$; then we expect g to be a function too. We infer from the discussion of this example that OTTER cannot show that g is a function directly. Van der Poll applies *heuristic #3* to establish the following properties: g is a relation AND g is functional.

We point the reader to Appendix C.9.24 and C.9.25 for two different approaches to this example in Rodin. In both cases, the proof is discharged automatically without applying any of the heuristics. A more general example that could also be discharged automatically in Rodin is presented in Appendix C.9.26: the union of two functions f and g is a function when $dom(f) \cap dom(g) = \varnothing$.

7.4.3 An example from Bledsoe

We look at an example from Bledsoe (1971, p. 61) as it is interesting to note that the extensionality and Divide-and-Conquer heuristics (*heuristic #1* and *heuristic #3* respectively) have been applied for nearly 50 years and are still relevant. We want to show that

$$\mathbb{P}(A) \cap \mathbb{P}(B) = \mathbb{P}(A \cap B). \tag{7.15}$$

Bledsoe's theorem prover PROVER transformed the set equality in (7.15) into two subset conjuncts, i.e. the Divide-and-Conquer *heuristic #3* was applied to obtain:

$$(\mathbb{P}(A) \cap \mathbb{P}(B) \subseteq \mathbb{P}(A \cap B)) \land (\mathbb{P}(A \cap B) \subseteq \mathbb{P}(A) \cap \mathbb{P}(B)) \tag{7.16}$$

Now, (7.16) is a conjunction, so *heuristic #3* is applied again and each subproof is transformed as follows:

Subproof 1

$$\mathbb{P}(A) \cap \mathbb{P}(B) \subseteq \mathbb{P}(A \cap B) \tag{7.17}$$

$$\Leftrightarrow \quad (\forall x)(x \in (\mathbb{P}(A) \cap \mathbb{P}(B)) \Rightarrow x \in \mathbb{P}(A \cap B)), \textit{ heuristic \#1} \text{ for subsets,} \tag{7.18}$$

$$\Leftrightarrow \quad (\forall x)((x \in \mathbb{P}(A) \land x \in \mathbb{P}(B)) \Rightarrow x \in \mathbb{P}(A \cap B)) \tag{7.19}$$

$$\Leftrightarrow \quad (\forall x)((x \subseteq A \land x \subseteq B) \Rightarrow x \subseteq A \cap B) \tag{7.20}$$

$$\Leftrightarrow \quad (\forall x)((x \subseteq A \land x \subseteq B) \Rightarrow (x \subseteq A \land x \subseteq B)) \tag{7.21}$$

$$\Leftrightarrow \quad \top \tag{7.22}$$

Subproof 2

The proof is similar to Subproof 1.

PROVER requires a number of reference theorems (Bledsoe 1971, p. 62) to establish (7.15)), namely:

$$(\forall A)(\forall B)(A = B \Leftrightarrow (\forall x)(x \in A \Leftrightarrow x \in B)) \tag{7.23}$$

$$(\forall A)(\forall B)(\forall x)(x \in A \cap B \Leftrightarrow (x \in A \land x \in B)) \tag{7.24}$$

$$(\forall A)(\forall x)(x \in \mathbb{P}(A) \Leftrightarrow x \subseteq A). \tag{7.25}$$

This is similar to the approach one takes with OTTER, Vampire and Gandalf. However, Rodin no longer requires these reference theorems, since set-theoretic support is built into the sequent calculus via the axioms of set theory (Abrial 2010, p. 321) – see Section 6.6.3.

For an implementation of (7.15) in Rodin, see Appendix C.10.15.

7.5 Multivariate Functors

> **OTTER *Heuristic #4***: Always attempt to state definitions in the simplest possible terms – either not including functors, or else functors with the least number of variable argument positions.

When defining sets, relations and functions, one often faces a choice between generic formulas that contain variables and formulas that instead incorporate constants terms, but which are less widely applicable. Irrespective of the inference mechanism, variables introduce additional complexity since they allow replacements or bindings with like terms. The latter is often integral to discovering a proof, even though a prover can be led astray and may end up spending most of its time performing such replacements or bindings. Therefore, limiting the number of variables in our input, places the prover in a better position to discover a proof (assuming one exists).

7.5.1 OTTER

Van der Poll (2000, p. 85) illustrates *heuristic #4* by taking another look at (7.10), namely $\mathbb{P}(\{0,1\}) = \{\varnothing, \{0\}, \{1\}, \{0,1\}\}$. Let $C = \{0,1\}$ and $D = \mathbb{P}(\{0,1\})$.

Now, consider the following definition of D:

$$(\forall x)(x \in D \Rightarrow x \subseteq C) \tag{7.26}$$

where the subset definition is given by:

$$(\forall A)(\forall B)(A \subseteq B \Leftrightarrow (\forall x)(x \in A \Rightarrow x \in B)). \tag{7.27}$$

OTTER finds no proof of (7.10) using this indirect definition of D.

Removing one of the universally quantified variables in (7.27) reduces the effect of Skolemisation while using the following definition of subset (where C is now a constant)

$$(\forall A)(A \subseteq C \Leftrightarrow (\forall x)(x \in A \Rightarrow x \in C)) \tag{7.28}$$

allows OTTER to find a proof of (7.10) in 4 minutes 5 seconds.

7.5.2 Vampire and Gandalf

Vampire, on the other hand, is able to find a proof of (7.10) with or without *heuristic #4* applied. However, Steyn (2009, p. 87) records the difference in execution times before and after applying the heuristic as 21 seconds versus 0.1 seconds. We conclude, therefore, that Vampire also benefits from the use of this heuristic.

Similarly, Gandalf finds a proof of (7.10) in 8 minutes 11 seconds before *heuristic # 4* is applied and in 1 minute 31 seconds after the heuristic is applied.

7.5.3 Rodin

We have already seen in section 7.4 that Rodin can prove (7.10) directly. Let us consider another example in an attempt to establish the usefulness of *heuristic # 4*. Let

$$G = \{2x \mid x \in \mathbb{Z}\} \tag{7.29}$$

and let us proceed to show that G is a group under the total function $Op : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ given by

$$Op(x, y) = x + y. \tag{7.30}$$

Now, for $g_1, g_2, g_3 \in G$, (G, Op) is a group iff

1. $Op(g_1, g_2) \in G$, i.e. G is closed under Op;
2. $Op(g_1, Op(g_2, g_3)) = Op(Op(g_1, g_2), g_3)$, i.e. Op is associative;
3. $0 \in G$, i.e. G contains an identity element; and
4. $(\forall g)(g \in G \Rightarrow -g \in G)$, i.e. if G contains g, it also contains the inverse $-g$ of g.

Our first approach to formulating (1) and (2) in Event-B matches the group axioms closely, namely

$$\begin{aligned} & \forall g1, g2 \cdot g1 \in G \land g2 \in G \\ \Rightarrow \quad & Op(g1 \mapsto g2) \in G \end{aligned} \tag{7.31}$$

$$\begin{aligned} & \forall g1, g2, g3 \cdot g1 \in G \land g2 \in G \land g3 \in G \\ \Rightarrow \quad & Op(g1 \mapsto Op(g2 \mapsto g3)) = Op(Op(g1 \mapsto g2) \mapsto g3). \end{aligned} \tag{7.32}$$

However, the generality of these definitions prevents the Rodin provers from finding proofs for (7.31) and (7.32)). We apply *heuristic #4* and introduce $g1$, $g2$ and $g3$ as constants and restate the group axioms as follows:

$$Op(g1 \mapsto g2) \in G \tag{7.33}$$

$$Op(g1 \mapsto Op(g2 \mapsto g3)) = Op(Op(g1 \mapsto g2) \mapsto g3). \tag{7.34}$$

After the elimination of the quantified variables, proofs are generated by the SMT provers Z3 and CVC3 respectively.

Appendix C.4.4 shows a Rodin context specifying the four group axioms for G.

7.6 Intermediate Structure

> **OTTER *Heuristic #5***: Use indirect definitions for intermediate structures when indirect definitions appear less likely to produce complex functor expressions than direct definitions.

7.6.1 OTTER

Van der Poll (2000, p. 87) continues the discussion of multivariate functors, but focusses specifically on multivariate functors associated with direct definitions. For example, suppose we want to show:

$$A \times \bigcup(B) \subseteq \bigcup\{A \times X \mid X \in B\} \tag{7.35}$$

Since OTTER cannot accept the formula in set-theoretic notation, the intermediate structure $E = \{A \times X \mid X \in B\}$ must be unfolded and a possible direct definition is

$$(\forall x)(x \in E \Leftrightarrow (\exists X)(X \in B \wedge (\forall y)(\forall z)((y,z) \in x \Leftrightarrow y \in A \wedge z \in X))) \tag{7.36}$$

Given this definition of E, OTTER found no proof and ran out of memory after 9 minutes 51 seconds. An inspection of the clausal form of (7.36) reveals that $ORD(y,z)$[3] was clausified into the nested functor

$$ORD(f1(x1,x2),f2(x1,x2))$$

where $f1$ and $f2$ are Skolem functions of two variables (Steyn 2009, p. 89). We have already seen in the nested functor and multivariate functor heuristics above that this clausal form complicates the resolution process.

However, when E is defined indirectly, e.g.,

$$(\forall x)(x \in E \Leftrightarrow (\exists X)(X \in B \wedge x = A \times X)) \tag{7.37}$$

then OTTER is able to find a proof of (7.35) in just 0.02 seconds.

7.6.2 Vampire and Gandalf

Similarly, neither Vampire nor Gandalf could find proofs of (7.35) with the direct definition of E in (7.36) within 30 minutes (Steyn 2009, p. 89). However, when the definition of E in (7.37) is used, Vampire found a proof in 0.1 seconds and Gandalf in 0.3 seconds.

7.6.3 Rodin

Rodin accepts (7.35) as a theorem, but none of the provers can find a proof. Also, the sequent calculus of Event-B and the inference mechanisms of the Rodin provers are significantly different from resolution, and it was found experimentally that *heuristic #5* does not assist any of the provers. For example, additional theorems based on (7.36) and (7.37) are ignored by the theorem provers. In fact, every attempt at proving (7.35) ends in the following sequent:

$$\begin{array}{rl} & x \in A, s \in B, x0 \in s \\ \vdash & (\exists s)(s \in \{A \times X \mid X \in B\} \wedge (x, x0) \in s) \end{array} \tag{7.38}$$

An interactive proof can be generated by applying a number of proof rules (see Section 8.3):

[3] $ORD(x,y)$ represents the ordered pair (x,y), since ordered pair is not syntactically supported by OTTER's input language.

1. Apply ‘Remove inclusion’ to the goal

$$\vdash\ A \times union(B)\ \boxed{\subseteq}\ union(\{X \cdot X \in B \mid A \times X\}) \tag{7.39}$$

2. Apply ‘Remove membership’ to hypothesis

$$x \in A,\ x0\ \boxed{\in}\ union(B)\ \vdash\ x \mapsto x0 \in union(\{X \cdot X \in B \mid A \times X\}) \tag{7.40}$$

3. Apply ‘Remove membership’ to goal

$$x \in A,\ s \in B,\ x0 \in s\ \vdash\ x \mapsto x0\ \boxed{\in}\ union(\{X \cdot X \in B \mid A \times X\}) \tag{7.41}$$

4. Apply ‘Exists instantiation’ to goal, instantiate with $A \times s$

$$x \in A,\ s \in B,\ x0 \in s\ \vdash\ \boxed{\exists}\ s \cdot (\exists X \cdot X \in B \wedge A \times X = s) \wedge x \mapsto x0 \in s \tag{7.42}$$

5. Apply ‘Default auto tactic with SMT’

$$x \in A,\ s \in B,\ x0 \in s\ \vdash \exists X \cdot X \in B \wedge A \times X = A \times s \tag{7.43}$$

Finally a proof is produced by CVC3 (the first of the SMT provers in the ‘Default auto tactic with SMT’ strategy).

Appendix C.10.24 shows a context containing (7.35).

7.7 Element Structure

> **OTTER *Heuristic #6***: Specify the elements of relations and functions at the level of ordered pairs or ordered n-tuples whenever the tuples are opened up during the proof.

7.7.1 OTTER

Consider the following example (Van der Poll 2000, p. 88):

$$F = \{\varnothing \mapsto a, \{\varnothing\} \mapsto b, a \mapsto b\}\ \ \Rightarrow\ \ F^{-1} = \{a \mapsto \varnothing, b \mapsto \{\varnothing\}, b \mapsto a\} \tag{7.44}$$

In order to prove formula (7.44) with OTTER, Vampire or Gandalf, the formula must be rewritten without any reference to set-theoretic constructs. Van der Poll (2000, p. 89) proposed two approaches:

7.7.1.1 First Definition of Formula (7.44)

Let $ORD(x, y)$ be a functor representing the ordered pair (x, y). Then define F by

$$(\forall x)(x \in F \Leftrightarrow x = ORD(\varnothing, a) \vee ORD(\{\varnothing\}, b) \vee ORD(a, b)) \tag{7.45}$$

together with the following facts:

$$(\forall y)(\forall z)(ORD(y,z) \in F \Leftrightarrow ORD(z,y) \in F^{-1}) \tag{7.46}$$

$$(\forall x)(x \in F \Rightarrow (\exists y)(\exists z)(x = ORD(y,z))) \tag{7.47}$$

$$(\forall u)(\forall v)(\forall w)(\forall x)(ORD(u,v) = ORD(w,x) \Leftrightarrow (u = w \wedge v = x)) \tag{7.48}$$

When given the goal

$$(\forall x)(x \in F^{-1} \Rightarrow (x = ORD(a,\varnothing)) \vee (x = ORD(b,\{\varnothing\})) \vee (x = ORD(b,a))), \tag{7.49}$$

OTTER was unable to find a proof in 20 minutes.

7.7.1.2 Second Definition of Formula (7.44)

If we use the alternative, more direct definition of F given by

$$(\forall y)(\forall z)(ORD(y,z) \in F \Rightarrow (y = \varnothing \wedge z = a) \vee (y = \{\varnothing\} \wedge z = b) \vee (y = a \wedge z = b)) \tag{7.50}$$

together with definition (7.46) for F^{-1}, then OTTER proved the goal

$$(\forall y)(\forall z)(ORD(y,z) \in F^{-1} \Rightarrow (y = a \wedge z = \varnothing) \vee (y = b \wedge z = \{\varnothing\}) \vee (y = b \wedge z = a)) \tag{7.51}$$

in just 0.03 seconds.

7.7.2 Vampire and Gandalf

Steyn (2009, p. 91) reported similar results for both Vampire and Gandalf: neither could find a proof of (7.49) after 30 minutes, but proofs for (7.51) were generated in 0.1 and 0.2 seconds respectively. *Heuristic #6* was, therefore, useful for both Vampire and Gandalf.

7.7.3 Rodin

Rodin is able to generate a proof of formula (7.44) automatically. The problem can be stated directly in terms of the set-theoretic constructs of function, inverse function and ordered pair. A complete Rodin context containing this example is included in Appendix C.10.10, but the statement of the theorem is included here for ease of reference: `axm1` defines F as a total function in terms of the carrier set U (see Section 6.4.1), and `thm2` is the direct statement of formula (7.44).

CONTEXT SetTheory10
AXIOMS
`axm1`: $F \in \mathbb{P}(\mathbb{P}(U)) \rightarrow \mathbb{P}(\mathbb{P}(U))$
`thm2`: ⟨theorem⟩
$F = \{\varnothing \mapsto a, \{\varnothing\} \mapsto b, a \mapsto b\} \Leftrightarrow F^{-1} = \{a \mapsto \varnothing, b \mapsto \{\varnothing\}, b \mapsto a\}$
END

We conclude this section with the remark that *Heuristic #6* was not needed to discharge any of the Rodin proof obligations in Appendix C.

7.8 Redundant Information

> **OTTER *Heuristic #7***: Do not include formulas that are not necessary to produce a proof.

7.8.1 OTTER

For the functions f and g, consider the following formula (Van der Poll 2000, p. 90):

$$[Fun(f) \wedge Fun(g) \wedge (\forall x)(x \in dom(f) \cap dom(g) \Rightarrow f(x) = g(x))] \Rightarrow Fun(f \cup g) \tag{7.52}$$

where $Fun(f)$ means f is a function.

The detailed first-order formulation of (7.52) relies on the definition of ordered pair, but if one also includes the following:

$$(\forall u)(\forall v)(\forall w)(\forall x)(ORD(u,v) = ORD(w,x) \Leftrightarrow (u = w \wedge v = x)) \tag{7.53}$$

OTTER is unable to find a proof within 30 minutes. However, removing (7.53) in the input results in a proof after only 0.08 seconds.

7.8.2 Vampire and Gandalf

Vampire was able to find a proof of (7.52) with or without the application of *heuristic #7* (Steyn 2009, p. 92), i.e. with or without the inclusion of (7.53). However, when additional definitions are included, such as

$$(\forall A)(\forall B)(\forall x)(\forall y)(ORD(x,y) \in PROD(A,B) \Leftrightarrow (x \in A \wedge y \in B)) \tag{7.54}$$

$$(\forall R)(\forall y)(y \in RAN(R) \Leftrightarrow (\exists x)(ORD(x,y) \in R)) \tag{7.55}$$

then Vampire is no longer able to find a proof within 30 minutes.

Gandalf did not need *heuristic # 7* in order to produce proofs when including the various redundant definitions in this section. Both with and without redundant information in the prover input, Gandalf required approximately 1 minute to find a proof of (7.52). A possible explanation of Gandalf's resilience to redundant information in this example, is that Gandalf selected a set of strategies that are likely to succeed based on the given input. Each strategy is allocated a time slice and runs until the strategy times out (Tammet 1997). This example shows that even if redundant information has an impact on some strategies, the proof attempt can still succeed (Steyn 2009, p. 93).

We also note that this example does not show that Gandalf will in all cases be resilient to the influence of redundant information.

7.8.3 Rodin

Example (7.52) does not illustrate the usefulness of *heuristic #7* for Rodin, since the Event-B syntax is sufficiently rich to not require the inclusion of definitions such as (7.53), (7.54) or (7.55). However, the Rodin user manual contains the following warning:

> "By providing only necessary hypotheses and no more, we drastically increase the chance that the prover will find the solution before timing out. On large models it is next to impossible to prove everything without hand-picking the hypotheses" (Jastram 2012, p. 69).

Similar to the situation with Gandalf, we conclude that *heuristic #7* is useful in general and we simply have not been able to construct a complex enough example to illustrate its application.

A Rodin context containing (7.52) is included in Appendix C.9.23.

7.9 Search-guiding

> **OTTER *Heuristic #8***: For biconditional formulas, generate and use only the half-definitions that are essential to the proof attempt.

7.9.1 OTTER

The previous heuristic warns against the inclusion of unnecessary information in the input. However, it is not always easy to decide what information is required to discharge a proof obligation. Van der Poll (2000, p. 91) proposed a technique to determine which part (called a *half-definition*) of a biconditional formula is vital and which part is less likely to contribute to the proof.

We consider the following example:

$$\bigcap(A \cup B) \subseteq \bigcap(A) \cap \bigcap(B) \tag{7.56}$$

Since OTTER is unable to process (7.56), we have to start unfolding the formula:

$$C = A \cup B \wedge D = \bigcap(C) \wedge E = \bigcap(A) \wedge F = \bigcap(B) \wedge G = E \cap F \Rightarrow D \subseteq G \tag{7.57}$$

where a first-order definition of the goal is

$$(\forall x)(x \in D \Rightarrow x \in G) \tag{7.58}$$

OTTER was used to attempt the proof at this point and Van der Poll outlined the technique for inspecting the generated output from the proof attempt: by inspecting the generated clauses, it is clear that of the two half-definitions of the first-order definition of G:

$$(\forall x)(x \in E \wedge x \in F \Leftrightarrow x \in G) \tag{7.59}$$

only the *only-if* direction is needed:

$$(\forall x)(x \in E \wedge x \in F \Rightarrow x \in G) \tag{7.60}$$

A similar procedure reveals that of the first-order definition of D, the arbitrary intersection of C, only a half-definition is needed:

$$(\forall x)(x \in D \Rightarrow (\forall b)(b \in C \Rightarrow x \in b)) \tag{7.61}$$

Without the application of *heuristic #9*, OTTER generated a proof for (7.56) in 8.36 seconds. However, a proof containing only the relevant half-definitions took 0.12 seconds. In essence, the proof attempt is speeded up because we guided the prover along the path of half-definitions.

7.9.2 Vampire and Gandalf

Vampire generated a proof of (7.56) in less than 1 second, both with or without applying *heuristic # 8*. In order to illustrate the use of half-definitions, Steyn (2009, p. 94) re-examined (7.35). The unfolded definition

$$C = \bigcup(B) \wedge D = A \times C \wedge E = \{A \times X \mid X \in B\} \wedge F = \bigcup(E) \Rightarrow D \subseteq F \tag{7.62}$$

combined with the definition of E given by (7.36) took Vampire 40 seconds to prove. Steyn identified and removed unused half-definitions from the definitions of C, D and F. The result was a proof in 0.5 seconds.

Gandalf could also generate a proof of (7.56) in less than 1 second with or without applying *heuristic #8* (Steyn 2009, p. 96). The more complex problem (7.62) required 1 minute 15 seconds before removing unused half-definitions and the time was reduced to 59 seconds when the unused half-definitions from the definitions of C, D and F were removed.

Heuristic # 8 was therefore particularly useful for Vampire and, although the effect was less dramatic, also for Gandalf.

7.9.3 Rodin

Although *heuristic #8* seems useful in general, it was not required to generate any of the proofs reported on in our collection of contexts and theorems in Appendix C. Specifically, all four of the SMT provers are able to generate a proof of (7.56) rapidly. A Rodin context containing (7.56) is included in Appendix C.10.25.

Neither did the more complex problem (7.62) require *heuristics #8*. See Section 7.6.3 for a discussion of (7.62).

7.10 Resonance

> **OTTER *Heuristic #9***: Give corresponding terms in formulas a syntactically similar structure.

7.10.1 OTTER

Van der Poll (2000, p. 97) attributes the idea behind the resonance heuristic to Wos (1995; 1996).

Let *Emp* be a set of employee records (Van der Poll 2000, p. 97). Each record consists of the following fields: personnel number, employee name, rank, department, monthly salary, and home address. Assume personnel number is a unique identifier. Then *Emp* belongs to the set of partial functions from *ID* (the set of all personnel numbers) to *PERSON*, i.e.

$$Emp \in ID \nrightarrow PERSON \tag{7.63}$$

where

$$PERSON = Name \times Rank \times Dept \times Salary \times Address \tag{7.64}$$

Now, suppose an employee with personnel number p receives a raise of *amount* per month. The set *Emp* must be updated to reflect this change and the raise can typically be specified as follows:

$$\begin{aligned} &(\forall x)(\forall n)(\forall r)(\forall d)(\forall s')(\forall a) \\ &(ORD(x, 5TUP(x,n,r,d,s',a)) \in Emp' \Leftrightarrow \\ &((x \neq p \wedge ORD(x, 5TUP(x,n,r,d,s',a)) \in Emp) \vee \\ &(x = p \wedge (\exists s)(x' = s + amount \wedge ORD(x, 5TUP(x,n,r,d,s,a)) \in Emp)))) \end{aligned} \tag{7.65}$$

A proof obligation arises from (7.65), namely to show that Emp' is functional:

$$Emp' \in ID \nrightarrow PERSON \tag{7.66}$$

A first-order definition of functional, called *Fun*, could be

$$(\forall R)(Fun(R) \Leftrightarrow (\forall u)(\forall v)(\forall w)(ORD(u,v) \in R \wedge ORD(u,w) \in R \Rightarrow (v = w))) \tag{7.67}$$

Two additional facts are needed to define ordered pair and 5-tuple equality:

$$(\forall u)(\forall v)(\forall w)(\forall x)(ORD(u,v) = ORD(w,x) \Leftrightarrow (u = w \wedge v = x))$$

and

$$\begin{aligned} &(\forall u)(\forall v)(\forall w)(\forall x)(\forall y)(\forall u')(\forall v')(\forall w')(\forall x')(\forall y') \\ &(5TUP(u,v,w,x,y) = 5TUP(u',v',w',x',y') \Leftrightarrow \\ &(u = u' \wedge v = v' \wedge w = w' \wedge x = x' \wedge y = y')) \end{aligned}$$

With these definitions OTTER failed to find a proof of (7.66) in 20 minutes. However, if (7.67) is rewritten

to be syntactically similar in corresponding terms to (7.65), e.g.,

$$(\forall R)(Fun(R) \Leftrightarrow$$
$$(\forall u)(\forall v)(\forall w)(\forall x)(\forall y)(\forall z)(\forall v')(\forall w')(\forall x')(\forall y')(\forall z')$$
$$(ORD(u, 5TUP(v,w,x,y,z)) \in R \wedge ORD(u, 5TUP(v',w',x',y',z')) \in R \Rightarrow$$
$$5TUP(v,w,x,y,z) = 5TUP(v',w',x',y',z')))$$

then OTTER finds a proof of (7.66) in 11.62 seconds.

7.10.2 Vampire and Gandalf

Vampire was not able to find a proof of (7.66) before the application of *heuristic #9* (Steyn 2009, p. 98). Once the resonance heuristic was applied, Vampire found a proof in less than 1 second. Gandalf was unable to prove (7.66) with or without the resonance heuristic applied. *Heuristic # 9* was therefore only useful for Vampire.

7.10.3 Rodin

Rodin does not require these first-order definitions and the SMT provers can prove (7.66) directly. See Appendix C.9.20 for a Rodin context containing this example. Also, no other examples in our work required *heuristic #9*.

7.11 Tuple Condense

> **OTTER *Heuristic #10***: Reduce the number of variable arguments to a functor by reordering the arguments and folding arguments that are irrelevant to the proof attempt into one.

7.11.1 OTTER

The final heuristic is supplementary to the multivariate heuristic (Van der Poll 2000, p. 99). Where *heuristic #4* advocates parsimony in the inclusion of multivariate functors, the current heuristic proposes a method whereby multivariate functors can be simplified for the purpose of the proof attempt.

We continue with the previous example and take another look at (7.65). The only coordinate that changes during the salary update is the fourth coordinate, s. One can therefore reorder and collapse the remaining variables as follows:

Reorder the tuple $ORD(x, 5TUP(n,r,d,s',a))$ as $ORD(x, 5TUP(n,r,d,a,s'))$, then group the first four arguments into a 4-tuple and represent this 4-tuple by a single variable, say y. We apply this simplification to

(7.65)):

$$(\forall x)(\forall y)(\forall s')(ORD(x, ORD(y, s')) \in Emp' \Leftrightarrow$$
$$((x \neq p \wedge ORD(x, ORD(y, s')) \in Emp) \vee$$
$$(x = p \wedge (\exists s)(x' = s + amount \wedge ORD(x, ORD(y, s)) \in Emp)))) \quad (7.68)$$

When combining (7.68) with a resonance version of functionality, e.g.,

$$(\forall R)(Fun(R) \Leftrightarrow$$
$$(\forall u)(\forall v)(\forall w)(\forall v1)(\forall w1)$$
$$(ORD(u, ORD(v, w)) \in R \wedge ORD(u, ORD(v1, w1)) \in R \Rightarrow ORD(v, w) = ORD(v1, w1))) \quad (7.69)$$

OTTER generated a proof of (7.66) in 0.07 seconds (as opposed to 11.62 seconds when just the resonance heuristic was applied).

7.11.2 Vampire and Gandalf

We recall that Vampire was already able to generate a proof of (7.66) after the application of the resonance heuristic. Tuple condensing did not improve the execution time further.

Gandalf was unable to prove (7.66) even when tuple condensing was applied.

7.11.3 Rodin

We have already noted that Rodin was able to proof (7.66) without any application of either the resonance or the tuple condense heuristics. Also, no other examples in our work required *heuristic #10*.

7.12 Summary

In this chapter we investigated to what extent the OTTER heuristics were applicable to the hybrid theorem proving environment Rodin. Steyn (2009) has also evaluated the OTTER heuristics for the resolution-based theorem provers Vampire and Gandalf. Steyn's findings are included here for comparative purposes. We evaluated 10 of the original 14 OTTER heuristics. Our results are summarised in Table 7.2.

We see that Vampire and Gandalf both required a significant number of the OTTER heuristics: Vampire needed 9 of the 10 heuristics, and Gandalf needed 8 of the 10 heuristics that were evaluated. Both provers performed better than OTTER and in a number of cases, Steyn had to enlarge the problems in (Van der Poll 2000) in order to illustrate the usefulness of a given heuristic (Steyn 2009, p. 100).

However, our experimental work revealed that the OTTER heuristics were largely ineffectual in Rodin. Rodin only needed 2 of the 10 heuristics that were evaluated. In all other cases, either problems that were hard

Heuristic name	Description	OTTER	Vampire	Gandalf	Rodin
Heuristic #1	Equality versus Extensionality	Yes	Yes	Yes	No
Heuristic #2	Nested Functors	Yes	Yes	Yes	No
Heuristic #3	Divide-and-Conquer	Yes	Yes	Yes	Yes
Heuristic #4	Multivariate Functors	Yes	Yes	Yes	Yes
Heuristic #5	Element Structure	Yes	Yes	Yes	No
Heuristic #6	Element Structure	Yes	Yes	Yes	No
Heuristic #7	Redundant Information	Yes	Yes	Yes	No
Heuristic #8	Search-guiding	Yes	Yes	Yes	No
Heuristic #9	Resonance	Yes	Yes	No	No
Heuristic #10	Tuple Condense	Yes	No	No	No

Table 7.2: Summary of the Applicability of the OTTER Heuristics

for the resolution-based provers could be proved automatically, or a different strategy was required to discharge proof obligations that could not be discharged automatically in Rodin.

In the next chapter we develop a number of heuristics specifically for Rodin. These heuristics were derived from our experience with proof obligations that could not be discharged automatically — for example, see Section 7.6.3.

Chapter 8

Rodin Heuristics

In Chapter 7 we discussed the applicability of the OTTER heuristics to building contexts in Rodin. The experimental results in the previous chapter led us to the conclusion that the OTTER heuristics are unlikely to increase the number of automatically discharged proof obligations (POs) in Rodin.

However, Abrial's vision for Rodin/Event-B was to provide a modelling tool for complex systems with an automatic discharge rate of at least 90% (Abrial 2010, p. xviii). Therefore, Rodin provides a number of mechanisms to achieve high proof obligation discharge rates. In this chapter we discuss four mechanisms that were applied to produce proofs for the contexts in Appendix C. These mechanisms are either available in the default Rodin installation or — in the case of the provers — can be installed as Eclipse plug-ins, namely:

- Auto tactic profiles (Section 8.1);
- Different provers (Section 8.2);
- Interactive proof rules (Section 8.3); and
- Constants (Section 8.4).

The chapter is concluded with a summary in Section 8.5.

The outcome of this chapter is a list of heuristics which can be applied while discharging proof obligations for theorems in Rodin contexts. The purpose of these heuristics is to either increase the number of automatically discharged POs or, when a PO cannot be discharged automatically, to suggest some simple steps to simplify the proof interactively.

We did not evaluate the applicability of these heuristics to Rodin machines, but seeing that the underlying proof mechanism is the same for contexts and machines one can surmise that these heuristics will yield similar results when applied to proof obligations generated for Rodin machines.

8.1 The Default Auto and Post Tactic Profiles

In Section 6.11 we discussed Rodin's automated theorem proving strategy, namely tactic profiles. We recall that a tactic profile consists of (1) combinators and (2) tactics that wrap rewrite rules, inference rules and provers. Two tactic profiles, namely the default auto and post tactic profiles, are included in the Rodin installer and are enabled by default. When the default tactic profiles are disabled, proof obligations must be discharged interactively.

When applied to a Rodin workspace that contains the contexts listed in Appendix C, the default auto and post tactic profiles yield a 98.1% automatic discharge rate for well-definedness (WD) proofs. This is above the 90% automatic proof discharge rate envisioned by Abrial (2010, p. xviii).

In terms of discharging theorem (THM) proofs, however, the default auto and post tactic profiles fall short of the ideal 90% discharge rate. When applied to the Rodin workspace that contains the contexts listed in Appendix C, only 33.0% of the THM proof obligations could be discharged automatically.

When combined, the default auto and post tactic profiles were only capable of discharging 60.5% of the WD and THM proof obligations, 29.5% below the ideal 90% automatic discharge rate. These findings are summarised in Table 8.1.

Number of contexts:	75
Number of axioms:	215
Number of theorems:	215

(a) Contexts, Axioms and Theorems

	Total	Automatic		Not discharged	
WD:	157	154	98.1%	3	1.9%
THM:	215	71	33.0%	144	67.0%
Total:	372	225	60.5%	147	39.5%

(b) WD and THM Proofs

Table 8.1: Context Statistics

8.2 Different Provers

Next we turn our attention to the large number of proof obligations that could not be discharged automatically when the default auto and post tactic profiles were applied. Recall that 147 proof obligations — 3 WD and 144 THM proof obligations — could not be discharged automatically (see Table 8.1).

In this section we take a closer look at a number of these unsuccessful proof attempts. We show that many

additional proof obligations could be discharged when external theorem provers were applied. Note that these provers are not included in the default auto and post tactic profiles. The following external theorem provers were used in our experimental work: ProB (see Section 5.5.4), and the SMT provers CVC3, CVC4, veriT and Z3 (see Section 5.5.5).

8.2.1 SMT Proof Obligations

In this section we present two boolean-valued functions, namely *HasControlValues* and *HasElementProperty*. Each function maps a family of sets of integers to the boolean values TRUE or FALSE depending on the property expressed by the function. In Subsections 8.2.1.1 and 8.2.1.2 we discuss applications of these functions to specific sets where the proof obligations could only be discharged by one of the SMT provers.

8.2.1.1 *HasControlValues*

The first example we consider is `thm2` in the context `RelationsAndFunctions03` (included in Appendix C, Section C.9.3). This context extends `RelationsAndFunctions01` (see Section C.9.1) which introduces the constants *ControlValues* and *HasControlValues*. The Event-B constant *HasControlValues* is a boolean-valued function that evaluates to *TRUE* iff a member set of a domain element contains *ControlValues*.

The set-theoretic definitions of *ControlValues* and *HasControlValues* in Event-B notation are the following:

$$ControlValues \subseteq \mathbb{Z} \tag{8.1}$$

$$HasControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow \mathrm{BOOL} \tag{8.2}$$

$$\forall X \cdot X \subseteq \mathbb{P}(\mathbb{Z}) \Rightarrow$$

$$(HasControlValues(X) = \mathrm{TRUE} \Leftrightarrow (\exists x \cdot x \in X \wedge ControlValues \subseteq x)) \tag{8.3}$$

Properties of the function *HasControlValues* are highlighted in the following examples:

Example 8.1. $HasControlValues(\varnothing) = FALSE$ is vacuously true for any set of *ControlValues*, because $\varnothing$ contains no member sets. ■

Example 8.2. $HasControlValues(\mathbb{P}(\mathbb{Z})) = TRUE$, because $ControlValues \in \mathbb{P}(\mathbb{Z})$. ■

Example 8.3. Let $S = \{\{1\}, \{2\}, \{1,3,4\}\}$ and $ControlValues = \{1,4\}$.

Then $HasControlValues(S) = TRUE$, because $\{1,4\} \subseteq \{1,3,4\} \in S$. ■

Example 8.4. Let $S = \{\{1\}, \{2\}, \{1,3,4\}\}$ and $ControlValues = \{4,5,8\}$.

Then $HasControlValues(S) = FALSE$, because $\{4,5,8\} \not\subseteq T$ for any $T \in S$. ■

The Event-B theorem defined in `thm2` is the following:

$$\mathit{HasControlValues}(\{\{x \cdot x \in \mathbb{N} \mid x+1\},$$
$$\{x \cdot x \in \mathbb{Z} \mid 3 * x + 1\}, \{x \cdot 3 * x + 1 = 0 \mid x\}\}) = \text{FALSE} \tag{8.4}$$

If we let $\mathit{ControlValues} = \{0,1,2,3,4,5\}$, then the default tactic profiles were unable to prove theorem (8.4). However, when the default auto tactic with SMT was applied, the CVC4 prover discovered a proof almost instantaneously. When applied interactively[1], Z3 was also capable of discharging the proof obligation associated with `thm2`.

8.2.1.2 *HasElementProperty*

8.2.1.2.1 Discussion of *HasElementProperty*

Our second example in this section is another boolean-valued function. The function *HasElementProperty* is defined in the context `RelationsAndFunctions09`: *HasElementProperty* evaluates to *TRUE* iff every pair of member sets of a domain element has a non-empty intersection (see Appendix C.9.9).

In Event-B notation, the set-theoretic definition of *HasElementProperty* is the following:

$$\mathit{HasElementProperty} \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow \text{BOOL} \tag{8.5}$$

$$\forall S \cdot S \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \Rightarrow$$
$$(\mathit{HasElementProperty}(S) = \text{TRUE} \Leftrightarrow (\forall X, Y \cdot X \in S \wedge Y \in S \Rightarrow X \cap Y \neq \varnothing)) \tag{8.6}$$

Some interesting properties of *HasElementProperty* are explored in the following examples:

Example 8.5. $\mathit{HasElementProperty}(\varnothing) = \mathit{TRUE}$ is vacuously true, because $\varnothing$ does not contain two member sets with an empty intersection.

However, $\mathit{HasElementProperty}(\{\varnothing\}) = \mathit{FALSE}$, because $\{\varnothing\}$ contains a pair of (non-distinct) element sets that have an empty intersection, i.e. $\varnothing \cap \varnothing = \varnothing$. ■

Example 8.6. From Example 8.5 we can derive the following properties:

$\varnothing \in S \Rightarrow \mathit{HasElementProperty}(S) = \mathit{FALSE}$.

$\mathit{HasElementProperty}(\mathbb{P}(\mathbb{Z})) = \mathit{FALSE}$. ■

Example 8.7. For any set S, once it is established that $\mathit{HasElementProperty}(S) = \mathit{FALSE}$, it follows that supersets of S can also not have the *HasElementProperty* property, i.e.

$(\mathit{HasElementProperty}(S) = \mathit{FALSE} \wedge S \subseteq T) \Rightarrow \mathit{HasElementProperty}(T) = \mathit{FALSE}$.

Proof:

Let $\mathit{HasElementProperty}(S) = \mathit{FALSE}$ and let $S \subseteq T$. There are two cases to consider:

[1] The SMT theorem provers are applied sequentially and the sequence of application is: CVC3, CVC4, veriT and Z3.

- Case 1: $\varnothing \in S \Rightarrow \varnothing \in T \wedge \mathit{HasElementProperty}(T) = \mathit{FALSE}$ by Example 8.6; or
- Case 2: $\exists X, Y \cdot (X \in S \wedge Y \in S \wedge X \cap Y = \varnothing) \Rightarrow (X \in T \wedge Y \in T \wedge \mathit{HasElementProperty}(T) = \mathit{FALSE})$ by the definition of *HasElementProperty*.

We note that Case 2 implies Case 1 when $\varnothing \in S$, because instantiating both the existentially quantified variables X and Y with $\varnothing$ yields the first case. ■

Example 8.8. Let $S = \{\{0, 1, 2, 3\}, \{3\}, \{3, 4, 5\}\}$.

In order to calculate the value of $\mathit{HasElementProperty}(S)$, we have to evaluate the following member set intersections:

$$\begin{aligned} \{0,1,2,3\} \cap \{3\} &= \{3\} \\ \{0,1,2,3\} \cap \{3,4,5\} &= \{3\} \\ \{3\} \cap \{3,4,5\} &= \{3\} \\ \{0,1,2,3\} \cap \{0,1,2,3\} &= \{0,1,2,3\} \\ \{3\} \cap \{3\} &= \{3\} \\ \{3,4,5\} \cap \{3,4,5\} &= \{3,4,5\} \end{aligned}$$

Since none of these intersections are empty, we have $\mathit{HasElementProperty}(S) = \mathit{TRUE}$. ■

`RelationsAndFunctions11` extends context `RelationsAndFunctions09` and introduces two theorems:

$$\texttt{thm1} \quad \mathit{HasElementProperty}(\{\{x \cdot x > 5 \mid 2 * x - 1\}\}) = \mathit{TRUE} \tag{8.7}$$

$$\texttt{thm2} \quad \mathit{HasElementProperty}(\{\{x \cdot x > 5 \mid 2 * x - 1\}, \{3, 5, 7, 11\}, \{5, 15\}\}) = \mathit{TRUE} \tag{8.8}$$

The default tactic profiles were once again unable to prove either theorem (8.7) or (8.8). When the default auto tactic with SMT was applied, the CVC4 prover discovered a proof of theorem (8.7) almost instantaneously. However, in order to prove theorem (8.8), CVC4 required more than 2 minutes of processing.

8.2.1.2.2 Counting Member Set Intersections

The discussion of *HasElementProperty* is concluded by considering the potential number of member set intersections that has to be evaluated in order to show $\mathit{HasElementProperty}(S) = \mathit{TRUE}$ for some set S. Let n be the number of member sets in S. Since the definition of *HasElementProperty* does not preclude checking trivial member set intersections (i.e. $X \cap X = X$), there are two types of intersections to count:

- **Trivial intersections:** There are n trivial intersections, one for each member set; and
- **Other intersections:** There are $\binom{n}{2}$ other intersections, one for each distinct pair of member sets.

In other words, there are up to $n + \binom{n}{2}$ intersection tests for each $n \in \mathbb{N}$.

n	$n + \binom{n}{2}$	n	$n + \binom{n}{2}$
0	0	30	465
1	1	40	820
2	3	50	1,275
3	6	100	5,050
4	10	200	20,100
5	15	300	45,150
10	55	1,000	500,500
20	210	10,000	50,005,000

Table 8.2: Sample Values of $n + \binom{n}{2}$

We can therefore surmise that the probability of showing *HasElementProperty*(S) = *TRUE* will diminish rapidly for large n, because the maximum number of intersections will be large too. See Table 8.2 where we present some sample values of the maximum number of intersections required for different values of n.

8.2.2 Tactic Profiles with SMT

> ***Rodin Heuristic #1***: Enable the *Default Auto Tactic with SMT* and set the *Timeout for the SMT auto-tactic* to $0.15s$ seconds.

Throughout the discussion of our experimental results in the previous chapter (Chapter 7) and the current chapter (Chapter 8), the SMT provers were repeatedly able to discharge proof obligations that none of the other Rodin provers were able to discharge. It is therefore of interest to consider the impact of changing the auto tactic profile from the *default auto tactic profile* to the *default auto tactic with SMT*[2].

In the remainder of this section we discuss the effect of applying the *default auto tactic profile* as opposed to the *default auto tactic with SMT* to a Rodin workspace that contains the contexts listed in Appendix C. We have also evaluated the SMT profile with various time-out values for the SMT provers. The time-out value can be set (in milliseconds) in Rodin under *Windows* $\rightarrow$ *Preferences* $\rightarrow$ *SMT* and defaults to 1000 milliseconds. Time-out values of 10, 50, 100, 150, 200, 250, 500 and 1000 milliseconds were evaluated.

Table 8.3 presents these findings in terms of the amount of time (given in seconds) required to build the entire Rodin workspace, the percentage of automatically discharged WD proofs and the percentage of automatically discharged THM proofs (relative to the total number of WD and THM proofs respectively, see Table

[2]Note that the post tactic profile remained unchanged throughout the remainder of this section, irrespective of which default auto tactic profile was selected.

8.1)[3].

The row containing the SMT $0.15s$ tactic profile data is highlighted in Table 8.3, because it is significant when considering the trade-off between performance and the amount of time needed to build the workspace. The data shows an increase in the number of discharged proof obligations with an increase in the SMT time-out value. Increases in the number of discharged proof obligations are most pronounced for time-out values of up to $0.15s$. For larger time-out values, the number of discharged proof obligations either remains unchanged (WD proofs), or the increase is less pronounced (THM proofs). These trends can also be observed in Figure 8.1.

On the other hand, the amount of time required to build the workspace increased rapidly for larger SMT time-out values. All the tactic profiles up to the $0.15s$ time-out could build the workspace in approximately 3 minutes or less. However, further increases up to the $1.0s$ time-out eventually required $344s = 5$ minutes and 44 seconds (i.e. a 77.3% increase in build time when compared to the SMT $0.15s$ tactic profile), but no additional WD proof obligations and only 8.5% additional THM proof obligations were discharged. Figure 8.2 shows the nearly exponential growth in the amount of time required to build the workspace.

Profile	Duration	Increase	% WD	Improvement	% THM	Improvement
Default	$149s$	–	98.1%	–	33.0%	–
SMT $0.01s$	$157s$	5.4%	98.1%	0.0%	33.5%	1.4%
SMT $0.05s$	$173s$	10.2%	98.7%	0.6%	55.8%	66.7%
SMT $0.1s$	$183s$	5.8%	99.4%	0.6%	67.0%	20.0%
SMT $0.15s$	$194s$	6.0%	99.4%	0.0%	71.2%	6.3%
SMT $0.2s$	$204s$	5.2%	99.4%	0.0%	72.1%	1.3%
SMT $0.25s$	$214s$	4.9%	99.4%	0.0%	73.0%	1.3%
SMT $0.5s$	$262s$	22.4%	99.4%	0.0%	74.9%	2.5%
SMT $1.0s$	$344s$	31.3%	99.4%	0.0%	77.2%	3.1%

Table 8.3: Comparing Auto Tactic Profiles

Given the significance of the SMT $0.15s$ tactic profile to our experimental work, we conclude this section with a comparison between the SMT $0.15s$ tactic profile and three other tactic profiles. The objective is to justify the observation that the *default SMT time-out value* of 1 second was suboptimal for our experimental work. The three comparisons are summarised in Table 8.4.

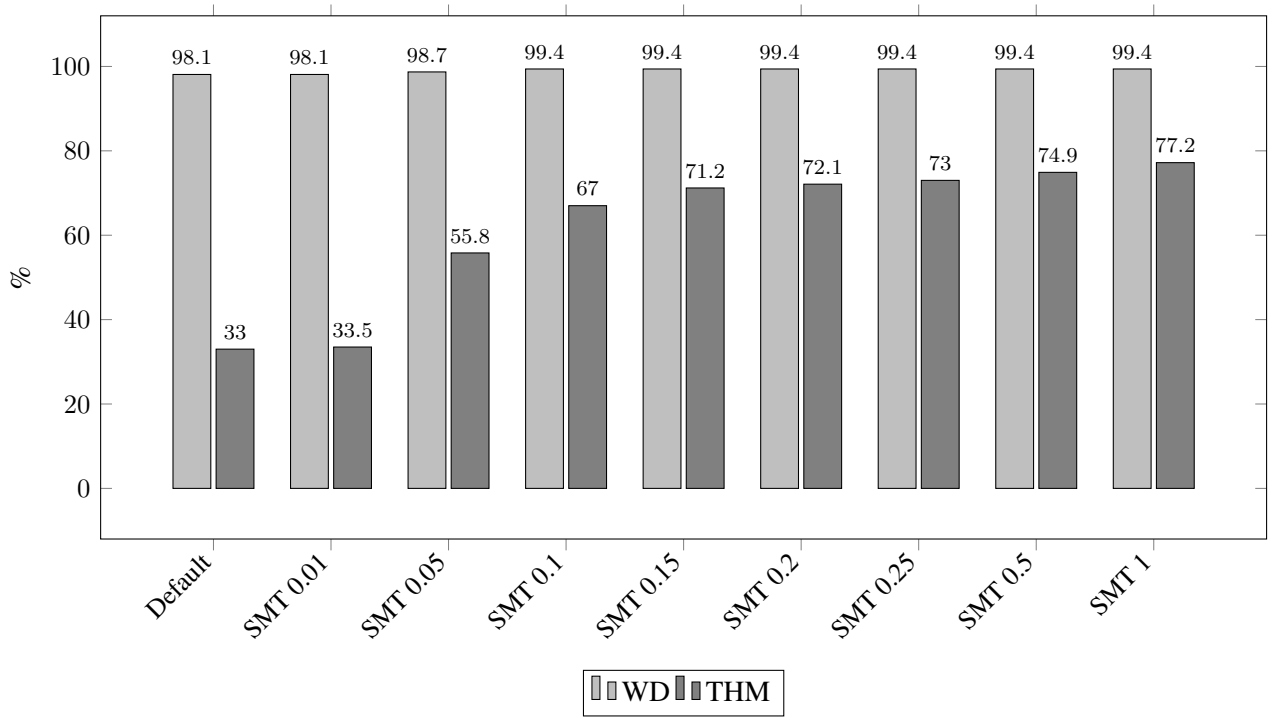


Figure 8.1: Percentage of Proof Obligations Discharged

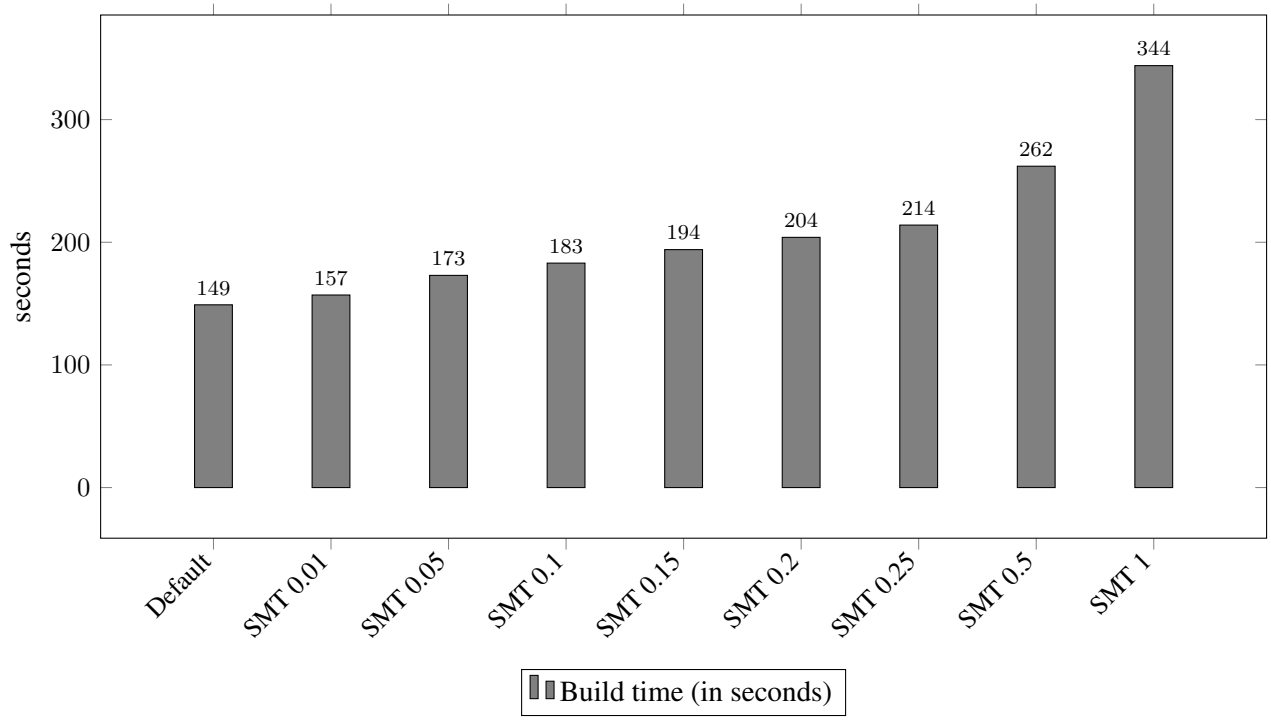


Figure 8.2: Time Required to Build the Rodin Workspace

Default versus SMT $0.15s$
SMT $0.01s$ versus SMT $0.15s$
SMT $0.15s$ versus SMT $1.0s$

Table 8.4: Auto Tactic Profile Comparisons

Profile	Duration	Increase	% WD	Improvement	% THM	Improvement
Default	$149s$	–	98.1%	–	33.0%	–
SMT $0.15s$	$194s$	30.2%	99.4%	1.3%	71.2%	115.5%

Table 8.5: Comparing Auto Tactic Profiles: Default versus SMT $0.15s$

8.2.2.1 Comparison 1: Default versus SMT $0.15s$

We have seen in Section 8.1 that the default auto tactic profile was not capable of discharging a large percentage of proof obligations: only 33.0% of the THM proofs. However, the default tactic profile built the workspace in 149 seconds.

By contrast, Table 8.5 shows an increase of 30.2% in the amount of time required to build the workspace using the SMT $0.15s$ tactic profile. But the table also shows a large increase of 115.5% in the number of discharged THM proof obligations as well as an increase of 1.3% in the number of discharged WD proof obligations.

In conclusion, we recall that the contexts in Appendix C consist of 372 WD and THM proofs. The default auto tactic profile was able to discharge a total of 225 (i.e. 60.5%) proof obligations in 2 minutes and 29 seconds. However, the SMT $0.15s$ tactic profile required an additional $45s$ to build the workspace, but a combined total of 309 (i.e. 83.1%) proof obligations where discharged — 6.9% below Abrial's 90% ideal automatic discharge rate (2010, p. xviii).

8.2.2.2 Comparison 2: SMT $0.01s$ versus SMT $0.15s$

Profile	Duration	Increase	% WD	Improvement	% THM	Improvement
SMT $0.01s$	$157s$	–	98.1%	–	33.5%	–
SMT $0.15s$	$194s$	23.6%	99.4%	1.3%	71.2%	112.5%

Table 8.6: Comparing Auto Tactic Profiles: SMT $0.01s$ versus SMT $0.15s$

[3]The improvements are calculated as follows:
If row *n* has a value of 10% and row $n+1$ has a value of 20%, then the improvement is: (20% - 10%) / 10% = 100%.
If row *n* has a value of 64.24% and row $n+1$ has a value of 74.24%, then the improvement is: (74.24% - 64.24%) / 64.24% = 15.6%.
Note that $64.24\% + 15.6\% = 79.84\% \neq 74.24\%$.

Inspection of Table 8.3 reveals that setting the time-out values for the SMT provers too low is detrimental to their performance. We see this by looking at the improvement in the number of discharged proof obligations between the default and the SMT $0.01s$ tactic profiles: 0.0% improvement in the number of WD proofs and 1.4% improvement in the number of THM proofs discharged.

Table 8.6 shows that increasing the time-out value from $0.01s$ to $0.15s$ has a far more dramatic effect: the number of discharged WD proofs increased by 1.3% and the number of discharged THM proofs increased by 112.5%.

In other words, when the time-out value is too low, many proofs are not discovered in time before the search is terminated.

8.2.2.3 Comparison 3: SMT $0.15s$ versus SMT $1.0s$

Profile	Duration	Increase	% WD	Improvement	% THM	Improvement
SMT $0.15s$	$194s$	–	99.4%	–	71.2%	–
SMT $1.0s$	$344s$	77.3%	99.4%	0.0%	77.2%	8.5%

Table 8.7: Comparing Auto Tactic Profiles: SMT $0.15s$versus SMT $1.0s$

Where time-out values that are too low will prevent the SMT provers from discovering many proofs, we see in Table 8.7 that high time-out values can unnecessarily increase the amount of time spent building the workspace, with no or relatively minor increases in the number of discharged proof obligations.

The direct comparison of the SMT $0.15s$ and the SMT $1.0s$ tactic profiles in Table 8.7 reveals an 8.5% increase in the number of discharged THM proof obligations, but requiring $344s = 5$ minutes and 44 seconds to build the workspace. That is an increase of 77.3% in the amount of time needed to build the workspace.

8.2.3 ProB Proof Obligations

> ***Rodin Heuristic #2***: Attempt *ProB* when none of the provers are able to discharge a proof obligation.

Consider `thm1` from context `SetTheory01` in Appendix C, Section C.10.1:

$$\begin{array}{l} \mathbb{P}(\{\{0\},\{1\},\{0,2\},\{1,2\}\}) \\ = \\ \{\varnothing,\{\{0\}\},\{\{1\}\},\{\{0,2\}\},\{\{1,2\}\},\{\{0\},\{1\}\},\{\{0\},\{0,2\}\}, \\ \{\{0\},\{1,2\}\},\{\{1\},\{0,2\}\},\{\{1\},\{1,2\}\},\{\{0,2\},\{1,2\}\}, \\ \{\{0\},\{1\},\{0,2\}\},\{\{0\},\{1\},\{1,2\}\},\{\{0\},\{0,2\},\{1,2\}\}, \\ \{\{1\},\{0,2\},\{1,2\}\},\{\{0\},\{1\},\{0,2\},\{1,2\}\}\} \end{array} \tag{8.9}$$

The default auto and post tactic profiles were unable to prove equation (8.9). However, ProB discharged the proof obligation in less than 2 seconds without any additional interactive assistance.

ProB was also capable of validating the power set of a 12-element set in less than 2 seconds (the power set has $2^{12} = 4,096$ elements). An example of a 12-element power set is included in Appendix C, Section C.10.5. None of the other theorem provers were able to discharge these theorems.

We note that these sets are much larger than any of the sets considered by Van der Poll (2000) and Steyn (2009).

8.3 Interactive Proof Rules

In this section we discuss a number of interactive proof rules that were applied to discharge the proof obligations for the remaining unproven theorems after the workspace was built with the "Default Auto Tactic with SMT". These proof obligations could only be discharged when interactive proof rules were applied (Jastram 2012, p. 106-7).

8.3.1 Apply Equality

> ***Rodin Heuristic #3***: Apply *Apply Equality* to formulas with the set equality operator.
> Check for useful substitutions or simplifications when trying both *Apply equality from left to right* and *Equality from right to left*.

Figure 8.3 shows the two options for applying an equality rewrite: either from left to right or from right to left. Application of the *apply equality* proof rule substitutes one side of the equation with the other throughout the proof sequent. When the proof rule is applied to the equality $\alpha = \beta$, then the option

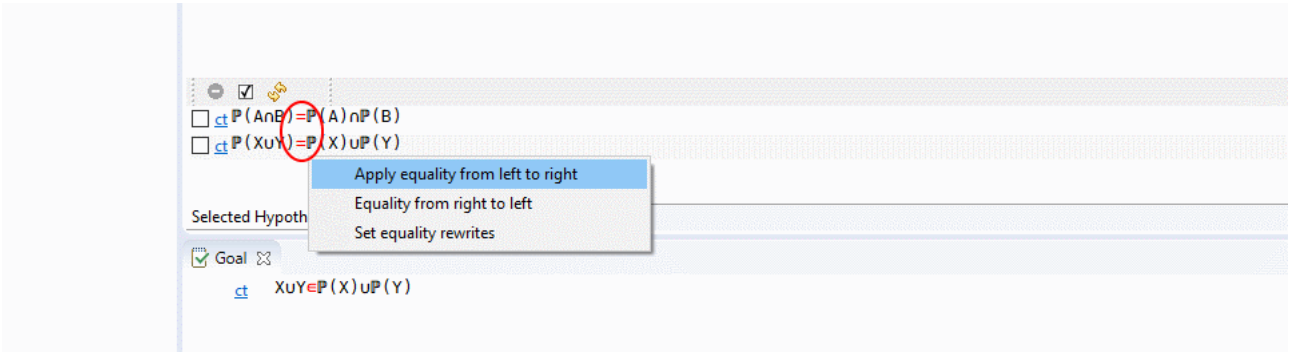


Figure 8.3: Interactive Proof Rule: Apply Equality

- **Apply equality from left to right** substitutes all occurrences of the expression α with the expression β (eliminating α from the sequent); and
- **Equality from right to left** substitutes all occurrences of the expression β with the expression α (eliminating β from the sequent).

Example 8.9. Let $S = \{0, 1\}$ be a formula in a Rodin sequent. Then *Apply equality from left to right* will replace all occurrences of S with $\{0, 1\}$ and *Equality from right to left* will replace all occurrences of $\{0, 1\}$ with S throughout the proof sequent. ■

In the remainder of this section we discuss an example that was adapted from (Bledsoe 1971, p. 61). When expressed in Event-B syntax, we want to show that

$$\begin{array}{ll} \texttt{axm1} & A \subseteq \mathbb{Z} \land B \subseteq \mathbb{Z} \\ \texttt{thm2} & \mathbb{P}(A \cup B) = \mathbb{P}(A) \cup \mathbb{P}(B) \;\Leftrightarrow\; A \subseteq B \lor B \subseteq A \end{array} \tag{8.10}$$

The Rodin theorem provers could not proof (8.10) either directly or with interactive assistance, so a lemma was added:

$$\begin{array}{ll} \texttt{axm1} & A \subseteq \mathbb{Z} \land B \subseteq \mathbb{Z} \\ \texttt{thm2} & \forall X, Y \cdot X \subseteq \mathbb{Z} \land \mathbb{P}(X \cup Y) = \mathbb{P}(X) \cup \mathbb{P}(Y) \\ & \Rightarrow\; X \cup Y \in \mathbb{P}(X) \cup \mathbb{P}(Y) \\ \texttt{thm3} & \mathbb{P}(A \cup B) = \mathbb{P}(A) \cup \mathbb{P}(B) \Leftrightarrow A \subseteq B \lor B \subseteq A \end{array} \tag{8.11}$$

The inclusion of the lemma enabled the provers to generate a proof of (8.10). However, the provers failed to prove lemma (8.11) directly and the proof attempt stopped after generating the following sequent:

$$\mathbb{P}(X \cup Y) = \mathbb{P}(X) \cup \mathbb{P}(Y) \;\vdash\; X \cup Y \in \mathbb{P}(X) \cup \mathbb{P}(Y).$$

A human observer can immediately see a way forward:

$$\begin{aligned} \mathbb{P}(X \cup Y) = \mathbb{P}(X) \cup \mathbb{P}(Y) \quad &\Rightarrow \quad X \cup Y \in \mathbb{P}(X) \cup \mathbb{P}(Y) \\ &\Rightarrow \quad X \cup Y \in \mathbb{P}(X \cup Y) \end{aligned} \tag{8.12}$$

The crucial step was replacing $\mathbb{P}(X) \cup \mathbb{P}(Y)$ with $\mathbb{P}(X \cup Y)$. This is achieved in Rodin via the interactive option *Equality from right to left*. Once the rewrite rule was applied, Rodin produced sequent (8.12) and the ML prover could discharge the proof obligation. Appendix C.10.15 contains a complete Rodin context for this example.

8.3.2 Set Equality Rewrites

> ***Rodin Heuristic #4***: Apply *Set equality rewrites* to formulas with the set equality operator.

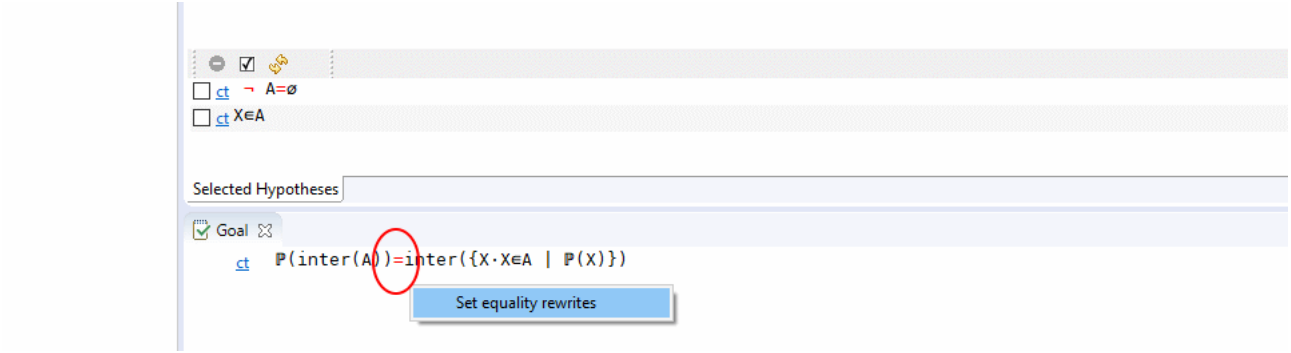


Figure 8.4: Interactive Proof Rule: Set Equality Rewrites

We have seen a similar heuristic, namely the Divide-and-Conquer heuristic in Section 7.4, where certain formulas are *manually rewritten* to simplify proof obligations. The OTTER heuristic recommends restructuring of the proof by splitting the goal into two subproofs when the goal contains a set-theoretic equality. Once the restructuring is applied, the outcome is two new proof obligations that require independent maintenance and processing.

One could argue that restructuring theorems in Rodin contexts to avoid set-theoretic equalities is desirable, since it could increase the number of automatically discharged proof obligations. But then, when dealing with hundreds or thousands of axioms and theorems (not unheard of in industrial settings), the introduction of an additional theorem for every equality can reduce readability and increase the cost of maintaining and modifying Rodin contexts, because one must always remember to edit two theorems instead of just one.

In this section, we see how Rodin improves on this situation: the subgoals are maintained as part of the automatically generated proof sequent and no changes to the context are necessary. *Set equality rewrites* is the name of the interactive rewrite rule in Rodin that automates the Divide-and-Conquer heuristic (see Section 7.4).

Application of the rewrite rule — see Figure 8.4 — splits a goal containing a set equality into two subgoals each containing a set inclusion.

Example 8.10. Let $\{0\} \cup \{1\} = \{0, 1\}$ be the goal of a proof sequent. Then *Set equality rewrites* rule will generate two child sequents with two new goals, namely $\{0\} \cup \{1\} \subseteq \{0, 1\}$ and $\{0, 1\} \subseteq \{0\} \cup \{1\}$. ■

Consider context `SetTheory26` (see Appendix C.10.26) where we show that the arbitrary intersection of the power set of a non-empty set A is equal to the power set of the arbitrary intersection of A. Stated in Event-B syntax, we have:

$$\begin{array}{ll} \texttt{axm1} & A \subseteq \mathbb{P}(\mathbb{Z}) \land A \neq \varnothing \\ \texttt{thm2} & \mathbb{P}(\text{inter}(A)) = \text{inter}(\{X \cdot X \in A \mid \mathbb{P}(X)\}) \end{array} \tag{8.13}$$

None of the provers were able to discharge the proof obligation for (8.13). The only available option was to apply the interactive set equality rewrite proof rule. The first subproofs produced by the rewrite rule, namely

$$\mathbb{P}(\text{inter}(A)) \subseteq \text{inter}(\{X \cdot X \in A \mid \mathbb{P}(X)\}),$$

could subsequently be discharged by the theorem prover PP.

The second subproof,

$$\text{inter}(\{X \cdot X \in A \mid \mathbb{P}(X)\}) \subseteq \mathbb{P}(\text{inter}(A)),$$

was more resilient. A series of interactive steps, such as *Remove Inclusion*, *Remove Membership*[4] and *Quantifier Instantiation*[5], had to be applied before arriving at the following sequent:

$$\frac{\begin{array}{c} X \in A \\ x0 \in x \\ s \in A \\ (\exists X \cdot X \in A \land \mathbb{P}(X) = \mathbb{P}(s)) \;\Rightarrow\; x \in \mathbb{P}(s) \end{array}}{x0 \in s}$$

The proof obligation could then be discharged by the theorem prover PP.

8.3.3 Quantifier Instantiation

> ***Rodin Heuristic #5***: Attempt *quantifier instantiation* to simplify formulas. Quantifiers can often be instantiated with symbols and definitions already present in the sequent.

[4] See Section 8.3.4 for a discussion of the *Remove Inclusion* and *Remove Membership* rewrite rules.
[5] See Section 8.3.3 for a discussion of *Quantifier Instantiation*.

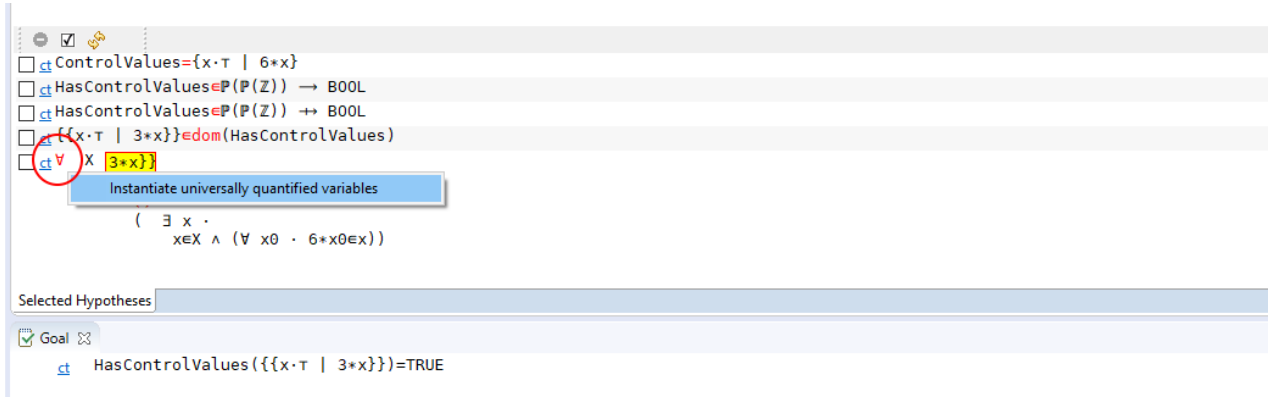


Figure 8.5: Interactive Proof Rule: Universal Quantifier Instantiation

The user can replace some or all of the quantified variables in a formula with correctly typed[6] variables, expressions or constants. Both universally and existentially quantified variables can be instantiated.

Example 8.11 (Universally quantified variable)**.** Consider the formula

$$(\forall x)(\exists y)(x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \;\Rightarrow\; x + 1 = y) \tag{8.14}$$

Since x is universally quantified and $x \in \mathbb{Z}$, x can be instantiated with any integer to produce additional true formulas such as

$$[x := -5] \quad (\exists y)(-5 \mapsto y \in \mathbb{Z} \times \mathbb{Z} \;\Rightarrow\; -5 + 1 = y)$$

$$[x := 0] \quad (\exists y)(0 \mapsto y \in \mathbb{Z} \times \mathbb{Z} \;\Rightarrow\; 0 + 1 = y)$$

$$[x := 7] \quad (\exists y)(7 \mapsto y \in \mathbb{Z} \times \mathbb{Z} \;\Rightarrow\; 7 + 1 = y)$$

Formula (8.14) can be replaced by any of the instantiated formulas in the list of hypotheses in a Rodin sequent, because formula (8.14) logically implies each of the instantiated formulas. ■

Example 8.12 (Existentially quantified variable)**.** Let $S = \{1, 2\}$ and $T = \{2, 3\}$, and consider the formula

$$(\exists x)(x \in S \wedge x \in T) \tag{8.15}$$

Formula (8.15) yields the true formula $2 \in S \wedge 2 \in T$ for the instantiation $[x := 2]$. ■

Example 8.13 (False statements)**.** When an instantiation such as $[x := 5]$ is applied to (8.15), we get to observe Rodin's handling of a false statement.

The instantiation results in the following proof sequent:

$$\frac{S = \{1, 2\} \qquad T = \{2, 3\}}{\text{Subproof 1: } 5 \in \{1, 2\} \quad \text{Subproof 2: } 5 \in \{2, 3\}}$$

As expected, none of the provers can discharge either of the subproofs. The subproofs neither follow

[6] See Section 6.4.1 for an explanation of Rodin data types.

syntactically from the list of hypotheses nor are they implied by set theory or the theory of integers. However, when $5 \in \{1, 2, 3\}$ was included in the list of hypotheses, the proof was trivial.

This example also illustrates the importance of insuring the correctness of statements included in the list of hypotheses, because a false statement does not preclude valid deductions. ■

Consider context `RelationsAndFunctions02` (see Appendix C.9.2) where we prove that the set

$$\{\{x \cdot x \in \mathbb{Z} \mid 3 * x\}\}$$

in `thm2` has the `HasControlValues` property. The property states that a family of sets must contain at least one member set, say *S*, such that *S* contains a set of control values. The definition of `HasControlValues` is expressed in the context `RelationsAndFunctions01` (see Appendix C.9.1):

CONTEXT RelationsAndFunctions01
CONSTANTS
ControlValues
HasControlValues
AXIOMS
axm1: $ControlValues \subseteq \mathbb{Z}$
axm2: $HasControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$
axm3:
$\forall X \cdot X \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \Rightarrow$
$(HasControlValues(X) = TRUE$
$\Leftrightarrow$
$(\exists x \cdot x \in X \land ControlValues \subseteq x))$
END

Now we want to show that

$$ControlValues = \{x \cdot x \in \mathbb{Z} \mid 6 * x\} \;\Rightarrow\; HasControlValues(\{\{x \cdot x \in \mathbb{Z} \mid 3 * x\}\}) = TRUE$$

This is expressed in Event-B as follows:

CONTEXT RelationsAndFunctions02
EXTENDS RelationsAndFunctions01
AXIOMS
axm1: $ControlValues = \{x \cdot x \in \mathbb{Z} \mid 6 * x\}$
thm2: ⟨theorem⟩ $HasControlValues(\{\{x \cdot x \in \mathbb{Z} \mid 3 * x\}\}) = TRUE$
END

Inspection of the generated proof obligation revealed that the list of hypotheses did not include the definition of `HasControlValues`:

$$\frac{ControlValues = \{x \cdot \top \mid 6 * x\}}{HasControlValues(\{\{x \cdot \top \mid 3 * x\}\}) = TRUE}$$

A single application of the *lasso* operator (see Section 8.3.5) produced the desired list of hypotheses:

$$ControlValues = \{x \cdot \top \mid 6 * x\}$$

$$HasControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$$

$$HasControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \nrightarrow BOOL$$

$$\{\{x \cdot \top \mid 3 * x\}\} \in dom(HasControlValues)$$

$$\frac{\forall X \cdot HasControlValues(X) = TRUE \Leftrightarrow (\exists x \cdot x \in X \wedge (\forall x0 \cdot 6 * x0 \in x))}{HasControlValues(\{\{x \cdot \top \mid 3 * x\}\}) = TRUE}$$

None of the provers could discharged the proof obligation at this point. However, the Rodin theorem proving UI has added a UI control to cater for the instantiation of the universally quantified variable X in the final hypothesis as shown in Figure 8.5. The instantiation $[X := \{\{x \cdot \top \mid 3 * x\}\}]$ resulted in the following proof sequent:

$$ControlValues = \{x \cdot \top \mid 6 * x\}$$

$$HasControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$$

$$HasControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \nrightarrow BOOL$$

$$\{\{x \cdot \top \mid 3 * x\}\} \in dom(HasControlValues)$$

$$\frac{\forall x \cdot \neg\, x = \{x \cdot \top \mid 3 * x\} \vee (\exists x0 \cdot \neg\, 6 * x0 \in x)}{HasControlValues(\{\{x \cdot \top \mid 3 * x\}\}) = TRUE}$$

Again, the theorem provers are unable to discharge the proof directly, but when the auto tactic profile with SMT was applied, both the CVC4and Z3could discharge the proof obligation.

8.3.4 Remove Inclusion and Remove Membership

> ***Rodin Heuristic #6***: Apply *Remove Inclusion* and *Remove Membership* to simplify formulas.

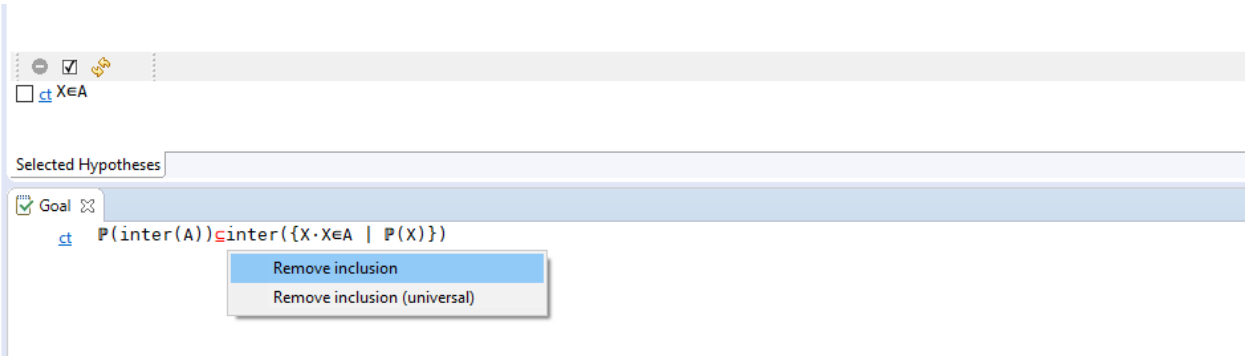


Figure 8.6: Interactive Proof Rule: Remove Inclusion

(a) Remove Membership in the List of Hypotheses

(b) Remove Membership in the Goal

Figure 8.7: Interactive Proof Rule: Remove Membership

Rodin's interactive theorem proving environment offers a number of ways to simplify formulas containing subset or membership operators. Figures 8.6 and 8.7 show how these rewrite rules can be applied to formulas in both the list of hypotheses and the goal. The behaviour of the rewrite rules discussed in this section are as follows:

- **Remove inclusion:** The subset operator is replaced by the definition given in Section 4.3.1: $X \subseteq Y$ is replaced by $(\forall x)(x \in X \Rightarrow x \in Y)$; and
- **Remove membership:** The membership operator is rewritten based on the definition of set containment.

  Example 8.14 (Member of Arbitrary Intersection)**.** When $x \in \mathit{inter}(X)$, then the membership operator can be removed to produce the formula $(\forall s)(s \in X \Rightarrow x \in s)$. See Section 4.3.3 for the discussion of arbitrary intersection. ■

  Example 8.15 (Member of Power Set)**.** When $x \in \mathbb{P}(X)$, then the membership operator can be removed to produce the formula $x \subseteq X$. See Section 4.6 for the discussion of power set. ■

We have seen in Section 8.3.2 that theorem `thm2` in context `SetTheory27` could not be discharged without application of the *Remove Inclusion* and *Remove Membership* rewrite rules. We resume the discussion of the theorem in context `SetTheory27` by repeating formula (8.3.2) before considering the series of interactive proof rules that had to be applied before the proof obligation could be discharged successfully:

$$\text{inter}(\{X \cdot X \in A \mid \mathbb{P}(X)\}) \subseteq \mathbb{P}\,(\text{inter}(A)) \tag{8.16}$$

None of the Rodin provers were able to discharge the proof obligation with goal (8.16). The only proof rule that is available in the proving UI is *Remove inclusion* and application of the proof rule produced the sequent:

$$\frac{\begin{array}{c} X \in A \\ x \in \mathrm{inter}(X \cdot X \in A \mid \mathbb{P}(X)) \end{array}}{x \in \mathbb{P}(\mathrm{inter}(A))}$$

The Rodin theorem provers were still not able to discharge the proof obligation and additional interactive steps had to be applied. First the goal was simplified through a series of interactive steps:

Remove membership:

$$\frac{\begin{array}{c} X \in A \\ x \in \mathrm{inter}(X \cdot X \in A \mid \mathbb{P}(X)) \end{array}}{x \subseteq \mathrm{inter}(A)}$$

Remove inclusion:

$$\frac{\begin{array}{c} X \in A \\ x \in \mathrm{inter}(X \cdot X \in A \mid \mathbb{P}(X)) \\ x0 \in x \end{array}}{x0 \in \mathrm{inter}(A)}$$

Remove membership:

$$\frac{\begin{array}{c} X \in A \\ x \in \mathrm{inter}(X \cdot X \in A \mid \mathbb{P}(X)) \\ x0 \in x \\ s \in A \end{array}}{x0 \in s}$$

Finally the formula $x \in \mathrm{inter}(X \cdot X \in A \mid \mathbb{P}(X))$ in the list of hypotheses could be simplified:

Remove membership:

$$\frac{\begin{array}{c} X \in A \\ x0 \in x \\ s \in A \\ (\forall s)(((\exists X)(X \in A \wedge \mathbb{P}(X) = s)) \Rightarrow x \in s) \end{array}}{x0 \in s}$$

Instantiate universally quantified variable ($[s := \mathbb{P}(s)]$):

$$X \in A$$

$$x0 \in x$$

$$s \in A$$

$$\frac{((\exists X)(X \in A \wedge \mathbb{P}(X) = \mathbb{P}(s))) \Rightarrow x \in \mathbb{P}(s)}{x0 \in s}$$

The final proof obligation could be discharged by all the Rodin theorem provers except ML.

8.3.5 Lasso

> ***Rodin Heuristic #7***: When all else fails, apply the *lasso* operator.

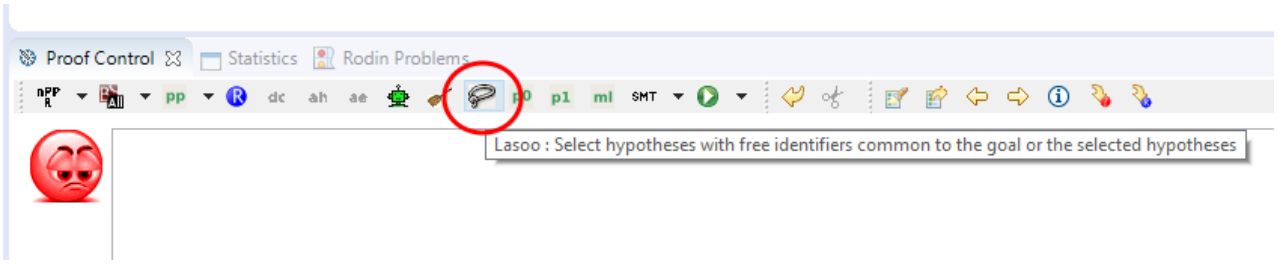


Figure 8.8: Interactive Operation: Lasso

Not all axioms and theorems in a context are automatically included as hypotheses in a proof. Here we are not referring to the dynamic hypothesis selection mechanism which only selects the preceding list of axioms and theorems from a context for the current goal (see Section 6.3). Instead, we are interested in looking at the case where the actual list of hypotheses is shorter than the potential list of all hypotheses for a generated proof obligation.

The *lasso* operation adds additional, unselected hypotheses (with free identifiers in common with the goal or the selected hypotheses) to the list of selected hypotheses. We have constructed two Rodin contexts to illustrate the selection of hypotheses:

- A base context which contains the definition of a function f:

CONTEXT LassoBaseContext
CONSTANTS
f
AXIOMS
axm1: $f \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \rightarrow \{0,1\}$
axm2:
$\forall X, Y \cdot \{X, Y\} \subseteq \mathbb{P}(\mathbb{Z}) \wedge X \cap Y \neq \varnothing \Leftrightarrow$
$f(X \mapsto Y) = 1$
END

- A derived context which contains the lemma *thm5* and a theorem about f:

CONTEXT LassoDerivedContext
EXTENDS BaseContext
AXIOMS
thm5: ⟨theorem⟩ $\{2,4,6,8,10\} \cap \{x \cdot x \in \mathbb{Z} \mid x * x\} \neq \varnothing$
Lemma
thm6: ⟨theorem⟩ $f(\{2,4,6,8,10\} \mapsto \{x \cdot x \in \mathbb{Z} \mid x * x\}) = 1$
END

Rodin generated the following proof obligation for thm6:

$$\frac{\neg\ \{2,4,6,8,10\} \cap \{x \cdot \top \mid x * x\} = \varnothing}{f(\{2,4,6,8,10\} \mapsto \{x \cdot \top \mid x * x\}) = 1}$$

We notice immediately that the definition of f was not included in the list of hypotheses and none of the theorem provers were able to discharge this proof obligation. A single application of the lasso operator produced the list of hypotheses needed to discharge the proof:

$$\frac{\begin{array}{c}\neg\ \{2,4,6,8,10\} \cap \{x \cdot \top \mid x * x\} = \varnothing \\ f \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \to \{0,1\} \\ f \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \nrightarrow \mathbb{Z} \\ \{2,4,6,8,10\} \mapsto \{x \cdot \top \mid x * x\} \in dom(f) \\ \forall X, Y \cdot X \cap Y = \varnothing \ \Rightarrow\ f(X \mapsto Y) = 0 \\ \forall X, Y \cdot \neg\ X \cap Y = \varnothing \ \Rightarrow\ f(X \mapsto Y) = 1\end{array}}{f(\{2,4,6,8,10\} \mapsto \{x \cdot \top \mid x * x\}) = 1}$$

Once the universal quantifier in the last hypothesis, namely

$$\forall X, Y \cdot \neg\ X \cap Y = \varnothing \ \Rightarrow\ f(X \mapsto Y) = 1,$$

was instantiated with $[X := \{2,4,6,8,10\},\ Y := \{x \cdot \top \mid x * x\}]$, the proof became trivial and the proof obligation could be discharged. These contexts are included in Sections C.6.1 and C.6.2 respectively.

8.4 Constants

> ***Rodin Heuristic #8***: Introduce *constants* for set-theoretic objects to assist the proof search effort and reduce duplication of definitions.

In Section 7.3 we have discussed the use of Skolem constants to simplify and unfold nested functors. Additional uses of Rodin constants are presented in this section.

In our experimental work we have seen a number of examples where the Rodin provers benefit from constants when constants are used to introduce explicit names for set-theoretic constructs. An example is discussed in Section 8.4.1 where a number of Rodin provers failed to discharge a proof obligation unless constants are declared in order to introduce explicit names.

Section 8.4.2 presents another advantage of the use of Rodin constants in the light of the DRY (*Don't Repeat Yourself*) principle: the use of constants reduces repetition of set-theoretic definitions. This, in turn, improves the maintainability and readability of contexts.

8.4.1 Automatically Discharged POs with Constants

In this section we present two logically equivalent Rodin contexts, namely context `SetTheory07_Simple` and `SetTheory07`. Context `SetTheory07_Simple` contains no constants:

CONTEXT SetTheory07_Simple
AXIOMS
`thm1`: ⟨theorem⟩ $inter(\{\{1,2,3\},\{2,3,4\}\}) = \{2,3\}$
END

The generated proof obligation for `thm1` could not be proved by either the default auto tactic profile or the default auto tactic profile with SMT. Only ProB — when applied interactively — could discharge the proof obligation.

Context `SetTheory07` introduces the constants $A = \{1,2,3\}$ and $B = \{2,3,4\}$:

CONTEXT SetTheory07
CONSTANTS
A
B
AXIOMS
`axm1`: $A = \{1,2,3\} \land B = \{2,3,4\}$
`thm2`: ⟨theorem⟩ $inter(\{A,B\}) = \{2,3\}$
END

Now the proof obligation for `thm2` could be discharged automatically by the default auto tactic profiles. All the theorem provers (i.e. PP, CVC3, CVC4, veriT and Z3) were able to discharge the proof obligation. Only ML could not discharge the proof.

We conclude this section with an example where the introduction of constants did not alter the behaviour of the theorem provers, but we see in the following section that constants can be beneficial even where their use does not yield additional automatically discharged proof obligations.

Example 8.16. Consider `thm1` in `SetTheory16_Simple` and `thm4` in `SetTheory16`: both theorems were proved automatically when the default auto tactic profile with SMT was applied. Here we

have an example where the introduction of constant symbols did not alter the behaviour of the theorem provers directly as observed in the discussion of context `SetTheory07` above.

Consider context `SetTheory16_Simple` which contains a single theorem:

CONTEXT SetTheory16_Simple
AXIOMS
`thm1`: ⟨theorem⟩
$\{x \cdot x \in \mathbb{Z} \mid 4 * x\}$
$\cap \{x \cdot x \in \mathbb{Z} \mid 9 * x\}$
$\cap \{x \cdot x \in \mathbb{Z} \mid 10 * x\}$
$= \{x \cdot x \in \mathbb{Z} \mid 2 * 2 * 3 * 3 * 5 * x\}$
END

`SetTheory16` contains the same theorem, but stated in terms of the constants A, B and C. These constants introduce explicit names for the sets containing the multiples of 4, 9 and 10 respectively.

CONTEXT SetTheory16
CONSTANTS
A
B
C
AXIOMS
`axm1`: $A = \{x \cdot x \in \mathbb{Z} \mid 4 * x\}$
`axm2`: $B = \{x \cdot x \in \mathbb{Z} \mid 9 * x\}$
`axm3`: $C = \{x \cdot x \in \mathbb{Z} \mid 10 * x\}$
`thm4`: ⟨theorem⟩ $A \cap B \cap C = \{x \cdot x \in \mathbb{Z} \mid 2 * 2 * 3 * 3 * 5 * x\}$
END

Both `thm1` in `SetTheory16_Simple` and `thm4` in `SetTheory16` could be proved automatically. ■

8.4.2 The DRY Principle

The DRY (Don't Repeat Yourself) principle was first formulated by Hunt and D. Thomas (2000, p. 26–8) in a section entitled "The Evils of Duplication". The context of their work "The Pragmatic Programmer" is software design, but their discussion of duplication has wider applicability. The principle is as follows:

> "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

We posit that the introduction of Rodin constants (to name set-theoretic constructs in contexts) is an example of design conforming to the DRY principle.

When contexts change and evolve over time, the benefit of having named set-theoretic objects is immediately obvious: every object is defined only once. Hence changes to definitions need not be repeated in various parts of a context or possibly across multiple contexts when contexts are extended.

Rodin supports the application of the DRY principle when designing a chain of extended contexts: typed constants can be declared in a base context and defined in a derived context.

For example, context `GroupTheory01`[7] contains the declaration of a set G and the group operator Op. Various properties of Op are stated in terms of G, but neither set-theoretic object is defined in the context:

CONTEXT GroupTheory01
CONSTANTS
G
Op
e
HasClosure
AXIOMS
axm1: $G \subseteq \mathbb{Z} \land Op \in G \times G \to \mathbb{Z}$
axm2: $HasClosure \in (G \times G \to \mathbb{Z}) \to BOOL$
$\vdots$
axm6:
$(\forall g1, g2 \cdot g1 \mapsto g2 \in dom(Op) \Rightarrow Op(g1 \mapsto g2) \in G)$
$\Leftrightarrow$
$HasClosure(Op) = TRUE$
END

The complete listing of context `GroupTheory01` in Appendix C.4.1 shows that definitions for G and Op are not included and appear in an extending context, namely context `GroupTheory02`[8]:

CONTEXT GroupTheory02
EXTENDS GroupTheory01
AXIOMS
axm1: $G = \{1, 2\} \land e = 1$
define G and e
axm2: $Op = \{1 \mapsto 1 \mapsto 1, 1 \mapsto 2 \mapsto 2, 2 \mapsto 1 \mapsto 2, 2 \mapsto 2 \mapsto 1\}$
multiplication mod 3
thm3: $\langle$theorem$\rangle$ $\forall x, y \cdot x \in G \land y \in G \Rightarrow Op(x \mapsto y) \in G$
lemma
thm4: $\langle$theorem$\rangle$ $HasClosure(Op) = TRUE$
END

Rodin imposes no limitations on the *distance* between the declaration of an object and the definition of the object (as long as the type of the object can be determined from the context containing the declaration). In other words, there is no limit on the number of extending contexts separating the declaration and definition of a set-theoretic object.

A word of caution seems in order here: Rodin does not enforce a single definition rule and application of the DRY principle is a matter of choice. Allowing an object to be defined through a series of partial definitions

[7] See Appendix C.4.1 for the complete listing of context `GroupTheory01`.
[8] See Appendix C.4.2 for the complete listing of context `GroupTheory02`.

plays an important part in specification refinement, but one should ensure that these partial definitions do not introduce a contradiction.

Example 8.17 (Contradicting set-theoretic definitions)**.**

CONTEXT Example
CONSTANTS
S
AXIOMS
axm1: $S = \{1\}$
axm2: $S = \{2\}$
thm3: ⟨theorem⟩ $\{1\} = \{2\}$
END

The proof obligation for thm3 was discharged easily by all the Rodin theorem provers. Only ProB reported the contradiction by displaying the message "contradiction in hypotheses". ■

8.5 Summary

In this chapter we presented a number of heuristics specifically for Rodin. We have shown that the *default auto tactic profile* was not capable of discharging a large number proof obligations when selected to build a workspace containing the contexts in Appendix C. Our first heuristic therefore recommended the *default auto tactic with SMT* with a time-out value of $0.15s$. An additional heuristic — Rodin Heuristic #2 — recommended the application of the theorem prover ProB when none of the other provers are capable of discharging a proof obligation.

Five heuristics — namely Rodin Heuristic #3 to Rodin Heuristic #7 — were formulated around frequently used interactive options in the Rodin proving UI. The eighth and final heuristic in this chapter recommended the use of Event-B constants.

Chapter 9

Case Studies

In Chapters 8 we have presented heuristics for discharging proof obligations in Rodin. In the current chapter, we look at examples of proof obligations that could not be discharged automatically without applying one or more of these heuristics. We present a number of examples, each demonstrating a different approach to assisting the automated theorem provers when a proof obligation cannot be discharged automatically.

In Section 9.1 we introduce Morgan's Z specification of a telephone network (Morgan 1993). We look at a proof obligation that arises from one of the system's properties — the minimality property — that cannot be discharged automatically by the default auto tactic profile.

In the following two sections — Sections 9.2 and 9.3 — we show how proof obligations that arise from the group axioms, and from evaluating polynomial intersections, can be discharged in Rodin once the necessary Rodin heuristics are applied.

The chapter is concluded with a summary in Section 9.4.

9.1 Telephone Network

The example in this section is a simple telephone network with connections between pairs of telephones (Morgan 1993, p. 29). The network operations are:

- **Call** - request a connection between telephones;
- **HangUp** - terminate a connection; and
- **Engaged** - indicate whether a specific telephone is engaged, and if engaged, which other phone is participating in the call.

These operations are not discussed here and we limit the discussion to the implementation of the state of the telephone network. Two properties of the telephone network are of interest:

- Ensuring that the telephone network is efficient, i.e. that the maximum number of calls are active at any given point in time; and
- Ensuring that the network operations do not terminate a connection unless termination is necessary to preserve the state invariant, i.e. a minimal set of connections must be terminated by each network operation.

In Section 9.1.2 we discuss a proof obligation that arises from the minimality property above.

9.1.1 The Z Specification

The set of all telephones is simply *PHONE*:

$[PHONE]$

A connection is a set of *PHONE*s:

$CON == \mathbb{P}\, PHONE$

Since no limitation is imposed on which subsets of *PHONE* constitute legitimate connections, the definition allows *conference calls* (i.e. more than two phones participating in a call) as well as *maintenance calls* (i.e. a phone calling itself, e.g., for testing purposes). Even the *empty call* is not excluded by the definition, but it is unlikely to be useful.

Two variables describe the state of the telephone network:

- $reqs : \mathbb{P}\, CON$ the set of requested connections (but not terminated yet); and
- $cons : \mathbb{P}\, CON$ the set of active connections.

Two invariants must be satisfied initially and after every operation:

- $cons \subseteq reqs$ only requested connections can be active; and
- $cons \in disjoint$ no phone may participate in more than one call at any given point in time.

The property *disjoint* applies to a set of sets and the definition is the following:

$[X]$

$$disjoint : \mathbb{P}(\mathbb{P}(\mathbb{P}\, X))$$

$$\forall\, cons : \mathbb{P}(\mathbb{P}\, X) \bullet$$
$$cons \in disjoint \iff (\forall\, c1, c2 : cons \bullet c1 \neq c2 \implies c1 \cap c2 = \varnothing)$$

The schema *TN* describes the state of the telephone network:

TN

$reqs, cons : \mathbb{P}\, CON$

$cons \subseteq reqs$

$cons \in disjoint$

It is desirable to have the maximum number of connections activated at all times. In other words, *cons* must be maximal with respect to *TN*: it must not be possible to enlarge the set *cons* while continuing to satisfy the *TN* invariant. The schema *efficientTN* describes this system property:

efficientTN

TN

$\neg\, (\exists\, cons0 : \mathbb{P}\, CON \bullet$

$\quad cons \subset cons0 \land$

$\quad cons0 \subseteq reqs \land$

$\quad cons0 \in disjoint)$

In full, the definition of *efficientTN* is:

efficientTN

$reqs, cons : \mathbb{P}\, CON$

$cons \subseteq reqs$

$cons \in disjoint$

$\neg\, (\exists\, cons0 : \mathbb{P}\, CON \bullet$

$\quad cons \subset cons0 \land$

$\quad cons0 \subseteq reqs \land$

$\quad cons0 \in disjoint)$

For ease of reference, we include the schema for *efficientTN′*:

efficientTN′

TN'

$\neg\, (\exists\, cons0 : \mathbb{P}\, CON \bullet$

$\quad cons' \subset cons0 \land$

$\quad cons0 \subseteq reqs' \land$

$\quad cons0 \in disjoint)$

The three network operations (*Call*, *HangUp* and *Engaged*) can now be described in terms of the state of the telephone network before (*efficientTN*) and after (*efficientTN'*), and the phone that initiated the operation:

$ph? : PHONE$

This is captured in the schema ΔTN along with an additional constraint: a connection cannot be terminated unless the connection violates the state invariants. The schema for ΔTN is:

ΔTN

$efficientTN$

$efficientTN'$

$ph? : PHONE$

$\neg\ (\exists\, cons1 : \mathbb{P}\, CON \bullet$

$(cons \setminus cons1) \subset (cons \setminus cons') \land$

$efficientTN'[cons1/cons'])$

A consequence of the invariant in ΔTN is that the set of connections terminated after an operation must be *minimal* with respect to *efficientTN'*. In other words, the set of connections cannot be further reduced while maintaining the *efficientTN'* invariant.

9.1.2 The Minimality Property

In essence, a connection must not be terminated unless termination is necessary to preserve the state invariant. In terms of the specification of the telephone network, this means that a connection c which (a) was active before an operation ($c \in cons$) and (b) is still requested after the operation ($c \in reqs'$), must remain active ($c \in cons'$). This yields the proof obligation

$$\forall x(x \in cons \cap reqs' \Rightarrow x \in cons') \tag{9.1}$$

A minimal Rodin context containing the state invariant defined in ΔTN shows that the proof obligation follows from the invariant:

CONTEXT TelephoneNetwork
SETS
PHONE PHONE = {0, 1, 2, 3, ...} = NAT
CONSTANTS
CON
reqs
cons
reqsPrime
consPrime
disjoint

AXIOMS

axm1: $CON = \mathbb{P}(PHONE)$

e.g., CON = {{}, {0}, {1}, ..., {0,1}, {3,4},...}

axm2:

$reqs \in \mathbb{P}(CON)$

$\wedge$

$reqsPrime \in \mathbb{P}(CON)$

e.g., reqs = {{{0,1}, {3,4},...}, {{0}, {1}}, ...}

axm3:

$\neg\ (\exists\, cons1 \cdot cons1 \in \mathbb{P}(CON)$

$\wedge\ cons \setminus cons1 \subset cons \setminus consPrime$

$\wedge\ ($

$\neg\ (\exists\, cons0 \cdot cons0 \in \mathbb{P}(CON)$

$\wedge\ cons1 \subset cons0$

$\wedge\ cons0 \subseteq reqsPrime$

$\wedge\ cons0 \in disjoint)$

$))$

delta TN

thm4: $\langle$theorem$\rangle\ \forall x \cdot x \in (cons \cap reqsPrime) \Rightarrow x \in consPrime$

END

Rodin's default auto tactic profile could not discharge the proof obligation for `thm4`, i.e. formula (9.1) above. Neither of the Atelier B provers PP or ML could discharge the proof obligation.

Next we applied Rodin Heuristic #1, restated here for ease of reference (see Section 8.2.2):

> ***Rodin Heuristic #1***: Enable the *Default Auto Tactic with SMT* and set the *Timeout for the SMT auto-tactic* to $0.15s$ seconds.

When the Rodin Heuristic #1 was applied, the proof obligation was discharged automatically by the SMT prover Z3. Additionally, the SMT provers CVC3, CVC4 and veriT could also discharge the proof obligation.

Appendix C.2.1 contains a Rodin context that includes not only this example, but also the system state invariants of all the Z schemas in this section. The context can form the basis of further exploration of the state properties of the telephone network *TN*.

9.2 Group Theoretic Properties

In this section, we look at a proof obligation that arises in the context of *group theory*. First we recall the group axioms:

Definition 9.1 (The Group Axioms)**.** Let U be any set. Let $G \subseteq U$ and $Op \in G \times G \rightarrow U$. The four axioms that $\langle G, Op \rangle$ must satisfy in order to be a group, are the following:

1. **Closure**: $g1, g2 \in G \ \Rightarrow\ Op(g1, g2) \in G$

2. **Associativity**: $g1, g2, g3 \in G \Rightarrow Op(Op(g1, g2), g3) = Op(g1, Op(g2, g3))$
3. **Identity element**: There exists $e \in G$ such that $Op(g, e) = g = Op(e, g)$ for all $g \in G$
4. **Inverse element**: For every $g \in G$ there exists $g^{-1} \in G$ such that $Op(g, g^{-1}) = e = Op(g^{-1}, g)$

■

Suppose we want to show that $G = \{x \cdot x \in \mathbb{Z} \mid 2 * x\}$ is a group under integer addition. The following Rodin context expresses the group axioms as theorems of the set G (see Appendix C.4.4):

CONTEXT GroupTheory04
CONSTANTS
G
Op
g1
g2
g3
AXIOMS
axm1: $G = \{x \cdot x \in \mathbb{Z} \mid 2 * x\}$
axm2: $Op \in \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$
axm3: $\forall x, y \cdot x \in \mathbb{Z} \wedge y \in \mathbb{Z} \Rightarrow Op(x \mapsto y) = x + y$
axm4: $g1 \in G \wedge g2 \in G \wedge g3 \in G$
thm5: ⟨theorem⟩ $Op(g1 \mapsto g2) \in G$
Group Axiom 1 - Closure
thm6: ⟨theorem⟩ $Op(g1 \mapsto Op(g2 \mapsto g3)) = Op(Op(g1 \mapsto g2) \mapsto g3)$
Group Axiom 2 - Associativity
thm7: ⟨theorem⟩ $0 \in G$
lemma 1
thm8: ⟨theorem⟩
$\exists e \cdot e \in G \wedge (\forall g \cdot g \in G$
$\Rightarrow Op(e \mapsto g) = g \wedge Op(g \mapsto e) = g)$
Group Axiom 3 - Identity Element
thm9: ⟨theorem⟩ $\forall x \cdot x + (-x) = 0 \wedge (-x) + x = 0$
lemma 2
thm10: ⟨theorem⟩ $\forall x \cdot 2 * x + (-2 * x) = 0 \wedge (-2 * x) + 2 * x = 0$
lemma 3
thm11: ⟨theorem⟩
$\forall g \cdot g \in G \Rightarrow$
$(\exists minusg \cdot Op(g \mapsto minusg) = 0$
$\wedge$
$Op(minusg \mapsto g) = 0)$
Group Axiom 4 - Inverse element
END

Several proof obligations were generated and Table 9.1 shows the proof status of each proof obligation when the default auto tactic profile was applied. Four proof obligations could not be discharged: thm5/THM, thm6/THM, thm8/THM and thm11/THM.

Next we applied Rodin Heuristic #1 (see Section 8.2.2):

Name	Status
axm3/WD	Automatically discharged
thm5/WD	Automatically discharged
thm5/THM	Not discharged
thm6/WD	Automatically discharged
thm6/THM	Not discharged
thm7/THM	Automatically discharged
thm8/WD	Automatically discharged
thm8/THM	Not discharged
thm9/THM	Automatically discharged
thm10/THM	Automatically discharged
thm11/WD	Automatically discharged
thm11/THM	Not discharged

Table 9.1: Proof Statuses for the Default Auto Tactic Profile

> ***Rodin Heuristic #1***: Enable the *Default Auto Tactic with SMT* and set the *Timeout for the SMT auto-tactic* to $0.15s$ seconds.

Once the Rodin Heuristic #1 was applied, three additional proof obligation were discharged. The updated proof statuses are shown in Table 9.2.

Name	Status	SMT Prover
axm3/WD	Automatically discharged	–
thm5/WD	Automatically discharged	–
thm5/THM	Automatically discharged	Z3
thm6/WD	Automatically discharged	–
thm6/THM	Automatically discharged	CVC3
thm7/THM	Automatically discharged	–
thm8/WD	Automatically discharged	–
thm8/THM	Not discharged	–
thm9/THM	Automatically discharged	–
thm10/THM	Automatically discharged	–
thm11/WD	Automatically discharged	–
thm11/THM	Automatically discharged	veriT

Table 9.2: Proof Statuses for the Default Auto Tactic with SMT

We note that the proof obligation for thm8/THM could not be discharged by any of the Rodin provers. Inspection of the transformed goal for thm8/THM revealed that the existential quantifier was preserved and

could be instantiated via the proving UI. We could therefore apply Rodin Heuristic #5 (see Section 8.3.3):

> ***Rodin Heuristic #5***: Attempt *quantifier instantiation* to simplify formulas. Quantifiers can often be instantiated with symbols and definitions already present in the sequent.

Application of the instantiation $[e := 0]$ to the goal

$$\boxed{\exists e} \cdot (\exists x \cdot 2 * x = e) \wedge$$
$$(\forall g \cdot (\exists x \cdot 2 * x = g) \Rightarrow Op(e \mapsto g) = g \wedge Op(g \mapsto e) = g)$$

yielded the transformed goal

$$(\exists x \cdot 2 * x = 0) \wedge$$
$$(\forall g \cdot (\exists x \cdot 2 * x = g) \Rightarrow Op(0 \mapsto g) = g \wedge Op(g \mapsto 0) = g)$$

and the proof obligation could be discharged by the SMT prover Z3.

9.3 Polynomial Intersections

In this section we look at polynomials with integer coefficients and use the Rodin provers to decide if specific polynomials intersect, specifically where intersections occurs at integer coordinates.

The set of polynomials we consider in the remainder of this section are defined in the following context (see Appendix C.10.18):

CONTEXT SetTheory18
CONSTANTS
- p1 $p1(x) = 2x - 1$
- p2 $p2(x) = 4x + 3$
- p3 $p3(x) = x^2$
- p4 $p4(x) = x^2 - 1$
- p5 $p5(x) = 2x^2 - 1$
- p6 $p6(x) = -x^4 - x^2 - 1$

AXIOMS
- axm1: $p1 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \wedge 2 * x - 1 = y \mid x \mapsto y\}$
- axm2: $p2 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \wedge 4 * x + 3 = y \mid x \mapsto y\}$
- axm3: $p3 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \wedge x * x = y \mid x \mapsto y\}$
- axm4: $p4 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \wedge x * x - 1 = y \mid x \mapsto y\}$
- axm5: $p5 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \wedge 2 * x * x - 1 = y \mid x \mapsto y\}$
- axm6: $p6 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \wedge -x * x * x * x - x * x - 1 = y \mid x \mapsto y\}$

END

For simple cases (e.g., line intersections) we will see that the proofs were discharged automatically. However, for polynomials of degree $\geqslant 2$, the provers were no longer able to discharge the proof obligations automatically.

Note that Rodin Heuristic #1 was applied throughout in the remainder of this section.

9.3.1 Intersecting versus Non-Intersecting Polynomials

Our next context defines polynomial intersections in terms of the intersection of sets of ordered pairs. Intersecting polynomials have a non-empty set-theoretic intersection and non-intersecting polynomials have an empty set-theoretic intersection (see Appendix C.10.19):

CONTEXT SetTheory19
EXTENDS SetTheory18
AXIOMS
thm1: ⟨theorem⟩ $p1 \cap p2 \neq \varnothing$
Intersecting polynomials, deg 1 and 1
thm2: ⟨theorem⟩ $p3 \cap p4 = \varnothing$
Non-intersecting polynomials, deg 2 and 2
thm3: ⟨theorem⟩ $p1 \cap p4 \neq \varnothing$
Intersecting polynomials, deg 1 and 2
thm4: ⟨theorem⟩ $p1 \cap p3 \neq \varnothing$
Intersecting polynomials, deg 1 and 2
thm5: ⟨theorem⟩ $p3 \cap p5 \neq \varnothing$
Intersecting polynomials, deg 2 and 2
thm6: ⟨theorem⟩ $p3 \cap p6 = \varnothing$
Non-intersecting polynomials, deg 2 and 4
END

Three proof obligations were discharged automatically: thm1/THM, thm2/THM and thm6/THM. Proof obligations for the remaining proofs — thm3/THM, thm4/THM and thm5/THM — could only be discharged after several interactive steps. The same initial sequence of interactive steps were applied to discharge the three remaining proof obligations:

- After the failed attempt to discharge the proof obligation, each of the goals were rewritten as the negation of a set equality, i.e. $\neg\, X = \varnothing$. We could therefore apply Rodin Heuristic #4 (see Section 8.3.2):

> ***Rodin Heuristic #4***: Apply *Set equality rewrites* to formulas with the set equality operator.

- Each goal was rewritten to have the form $X \subseteq \varnothing$. So, the next step was to apply Rodin Heuristic #6 to eliminate the $\subseteq$-operator (see Section 8.3.4):

> ***Rodin Heuristic #6***: Apply *Remove Inclusion* and *Remove Membership* to simplify formulas.

- Now the goals had the form $\exists x, x0 \cdot x \mapsto x0 \in X$ and Rodin Heuristic #6 was applied again to remove the $\in$-operator.
- The goals were rewritten to have the form $\exists x, x0 \cdot x \mapsto x0 \in X \land x \mapsto x0 \in Y$, where X and Y are one of the polynomials $p1$, $p3$, $p4$ or $p5$ respectively. The final interactive step in this sequence was to replace the polynomial names with their definitions. Once this was done, thm3/THM was discharged automatically.

Finally a single application of Rodin Heuristic #5 (see Section 8.3.3) was required, because the rewritten goals for thm4/THM and thm5/THM were $\exists x \cdot x * x = 2 * x - 1$ and $\exists x \cdot 2 * x * x - 1 = x * x$ respectively:

> ***Rodin Heuristic #5***: Attempt *quantifier instantiation* to simplify formulas. Quantifiers can often be instantiated with symbols and definitions already present in the sequent.

After applying the existential quantifier instantiation $[x := 1]$, the automated theorem prover ML could discharge the proof obligations.

9.3.2 Checking Polynomial Intersection Points

In this section we apply the Rodin provers to evaluate the intersection points for the intersecting polynomials in context `SetTheory18` (introduced in Section 9.3). The intersection points are included in context `SetTheory20` (see Appendix C.10.20):

CONTEXT SetTheory20
EXTENDS SetTheory18
AXIOMS
`thm1`: ⟨theorem⟩ $p1 \cap p2 = \{-2 \mapsto -5\}$
Intersecting polynomials, deg 1 and 1
`thm2`: ⟨theorem⟩ $p1 \cap p4 = \{0 \mapsto -1, 2 \mapsto 3\}$
Intersecting polynomials, deg 1 and 2
`thm3`: ⟨theorem⟩ $p1 \cap p3 = \{1 \mapsto 1\}$
Intersecting polynomials, deg 1 and 2
`thm4`: ⟨theorem⟩ $p3 \cap p5 = \{1 \mapsto 1, -1 \mapsto 1\}$
Intersecting polynomials, deg 2 and 2
END

Only the proof obligation for thm1/THM was discharged automatically. None of the proof obligations for polynomials of degree $\geqslant 2$ could be discharged automatically by any of the Rodin provers. The interactive strategy for discharging the proof obligations of thm2/THM, thm3/THM and thm4/THM is similar to the sequence of steps outlined in Section 9.3.1.

9.4 Summary

We concluded in Chapter 7 that the OTTER heuristics are ineffectual when applied to undischarged proof obligations in Rodin. However, we have also seen in Section 8.1 that the Rodin provers were not able to discharge all the proof obligations that arise from the contexts in Appendix C. In other words, there is still a need for heuristics to assist with theorem proving in Rodin.

We have therefore presented a list of Rodin specific heuristics in Chapter 8. In the current chapter, we have discussed a number of proof obligations that could not be discharged automatically without the application of one or more of these Rodin heuristics.

In our first example, we considered a proof obligation that arose from a Z specification for a telephone network. The system's minimality property could not be proved by Rodin's *default auto tactic profile* and required the application of Rodin Heuristic #1.

Next we presented an implementation of the group axioms and again Rodin Heuristic #1 had to be applied to increase the number of automatically discharged proof obligations. One proof obligation also required the application of Rodin Heuristic #5 before a proof was generated.

The final example in this chapter presented a number of different approaches to evaluating intersection points for polynomials with integer coefficients. A few proof obligations remained undischarged after Rodin Heuristic #1 was applied and required a more complex interactive strategy before proofs were generated. These proof obligations required the application of Rodin Heuristic #4, Rodin Heuristic #6 and Rodin Heuristic #5 before yielding to Rodin's proof search efforts.

In our next and final chapter we assess to what extent the research aims identified in Chapter 1 were met.

Chapter 10

Conclusion and Future Work

In the last chapter of this dissertation we take a final look at our original research questions and hypotheses from Chapter 1, and we discuss the conclusions derived from applying the proposed approach in Section 1.4. The chapter concludes with our thoughts on the directions that future work in this area could follow.

10.1 Contributions of this Dissertation

Labuschagne and Van der Poll (1999) and Van der Poll (2000) have formulated a set 14 of reasoning heuristics to aid the resolution-based automated reasoner OTTER (McCune 2003) in finding proofs for set-theoretic problems. OTTER has since been decommissioned by its author, William McCune, and was replaced by Prover9 (McCune 2005). Despite the fact that OTTER was decommissioned, the question of whether the OTTER heuristics are applicable to other theorem provers remained of interest. Steyn (2009), for example, has evaluated the applicability of the OTTER heuristics to the then state-of-the-art resolution-based theorem provers Vampire and Gandalf. Steyn was able to show that the OTTER heuristics were applicable, and that Vampire needed 10 and Gandalf 9 of the 11 heuristics that were evaluated.

The success of Steyn's work led to the additional question of whether the OTTER heuristics are applicable to a hybrid theorem proving environment. We recall that a theorem proving environment is considered a *hybrid* theorem proving environment when the theorem proving tool applies at least two different inference mechanisms while discharging proof obligations.

Our primary hypothesis was:

> The set-theoretic heuristics developed by Labuschagne and Van der Poll for the resolution-based theorem prover OTTER are *not* applicable to a hybrid theorem proving platform.

The Rodin tool suite for the Event-B formal method (see Chapter 6) was selected to verify our hypothesis. Rodin is an Eclipse-based integrated development environment, and as such, its functionality can be extended

through plug-ins. A large number of theorem provers are available via plug-ins, for example, the term rewriter ML, the model checker ProB, and the SMT provers CVC3, CVC4, veriT and Z3. For this reason, Rodin was selected as the hybrid theorem proving environment we used in our research.

In Chapter 7 we state each of the OTTER heuristics obtained by Van der Poll and discuss the proof attempts by OTTER, Vampire and Gandalf. In a number of cases, Steyn had to increase the complexity of problems in order to demonstrate the utility of a give heuristic. Steyn concluded that Vampire needed 10 and Gandalf 9 of the 11 heuristics that were evaluated. Each of the problems in (Van der Poll 2000) and (Steyn 2009) were attempted in Rodin. We concluded that the OTTER heuristics were largely ineffectual or unnecessary, because either proofs that could not be discharged by the resolution-based theorem provers could be discharged by Rodin, or a different strategy was required when proof obligations could not be discharged automatically.

This led to our secondary hypothesis:

> Additional Rodin specific heuristics are necessary to discharge all the proof obligations that arise from the Rodin contexts in Appendix C.

In Chapter 8 we have presented a number of heuristics specifically for Rodin. These heuristics were applied in Chapter 9 to demonstrate their utility when discharging proof obligations in Rodin.

We have therefore provided evidence that the OTTER heuristics are not applicable to a hybrid theorem proving environment in the domain of set-theoretic problems. However, we have also shown that heuristics are still needed to discharge set-theoretic proof obligations in a hybrid theorem proving environment.

10.2 Future Work

The TPTP World is the infrastructure that includes the TPTP problem library (see Section 5.1.2 for a discussion of the TPTP library). One of the services in the TPTP World is the WWW interface called SystemOnTPTP (Sutcliffe 2010). SystemOnTPTP can be used to submit automated theorem proving problems to a range of automated theorem proving systems simultaneously and can be accessed at `http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP`.

SystemOnTPTP can automatically analyse input (using subsidiary tools) to select automated theorem proving systems that are capable of solving the submitted problem according to the problem's characteristics (defined in TPTP World terms by the problem's SPCs, i.e. Specialist Problem Classes).

Since it is possible to target multiple automated theorem provers via SystemOnTPTP, input restructuring heuristics (such as the OTTER heuristics) can be applied to a range of provers simultaneously. Observable outcomes can include:

- Does applying Heuristic X change the problem's SPC?

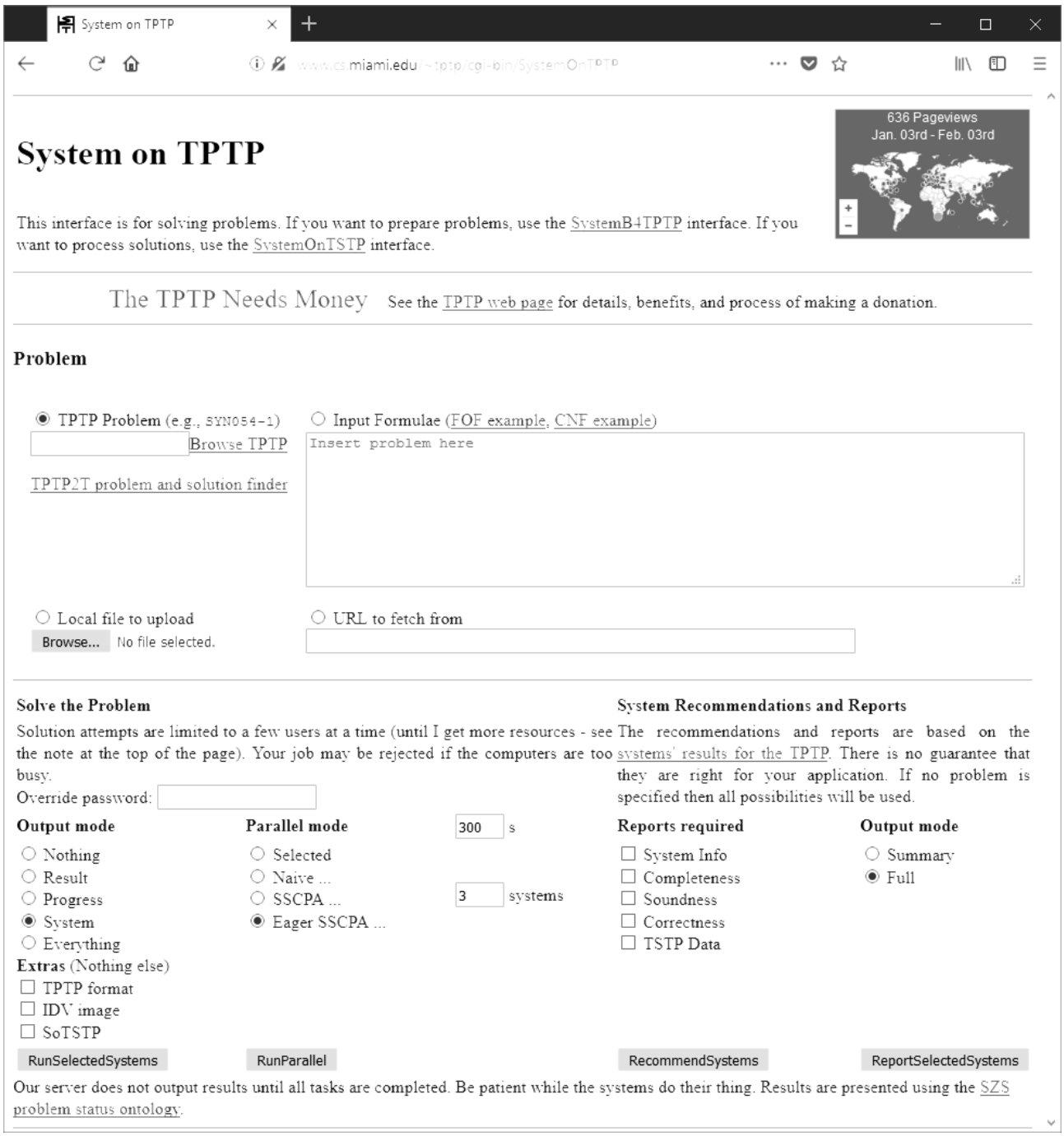


Figure 10.1: Screen Shot of SystemOnTPTP

- Does applying Heuristic X change the list of applicable theorem provers?
- Does applying Heuristic X enable theorem prover Y to find a proof where a proof could not be generated without the heuristic?
- Does applying Heuristic X alter theorem prover Y's proof output characteristics?

  For example, Did it take more or less time to generate a proof? Is the generated proof longer or shorter?

Benefits of evaluating automated theorem proving heuristics via SystemOnTPTP include:

- It is not necessary to install and run various automated theorem proving systems, since SystemOnTPTP

has access to hosted instances of a wide range of automated theorem proving; and

- Since all problems must be formulated in the TPTP language, it is not necessary to learn each automated theorem prover's input language.

Appendix A

Preparation of Experimental Results

Our experimental work was predominantly centred around whether or not proof obligations could be discharged automatically. In this Appendix we discuss our approach to counting the number of automatically versus manually discharged proof obligations over a Rodin workspace.

In order to understand the limitations of the statistical information presented in Rodin, we take a closer look at the structure of a Rodin workspace in Section A.1.

In Section A.2 we discuss the statistics tab in Rodin. The tab presents a number of counts at the Rodin project and context level.

We have created a C# console application to address Rodin's omission of statistics at the workspace level. Also, we were interested in differentiating between proof obligations for WD versus THM proofs. Such refined information is not available in Rodin. The C# application is discussed in Section A.3 and finally the source code is included in Section A.3.2.

A.1 The Structure of a Rodin Workspace

A Rodin workspace is a collection of projects. A project is a logical grouping of contexts as well as a scope delimiter for extending contexts: a context can only be extended within the containing project. Finally, we have seen in Chapter 6 that a context is a collection of carrier sets, constants, axioms and theorems.

Each Rodin context consists of five files that are contained within a project. These files have the following extensions: `.buc`, `.bcc`, `.bpo`, `.bpr` and `.bps`. A description of the content of various Rodin files can be found at `http://wiki.event-b.org/index.php/Rodin_File_Types`. We include the information pertaining to contexts in Table A.1 for ease of reference.

File Extension	Content	Type
.buc	Event-B Context	Xml
.bcc	Event-B Statically Checked Context	Xml
.bpo	Event-B Proof Obligations	Xml
.bpr	Event-B Proofs	Xml
.bps	Event-B Proof Statuses	Xml

Table A.1: Rodin File Extensions for Contexts

A.1.1 Rodin's BUC File Format

As shown in Table A.1, a .buc file contains the context definition. For example, the .buc file for the context LassoBaseContext has the following content (the axioms are listed in Appendix C.6.1):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<org.eventb.core.contextFile
   org.eventb.core.configuration="org.eventb.core.fwd"
   version="3">
  <org.eventb.core.constant name="'"
     org.eventb.core.identifier="f"/>
  <org.eventb.core.axiom name="("
     org.eventb.core.label="axm1"
     org.eventb.core.predicate="..."/>
  <org.eventb.core.axiom name=")"
     org.eventb.core.label="axm2"
     org.eventb.core.predicate="..."/>
</org.eventb.core.contextFile>
```

A.1.2 Rodin's BCC File Format

Additional information is included in the .bcc file, such as type information for constants and a differentiating label for theorems. The .bcc file for the context LassoBaseContext is the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<org.eventb.core.scContextFile org.eventb.core.accurate="true"
   org.eventb.core.configuration="org.eventb.core.fwd">
  <org.eventb.core.scAxiom name="'"
     org.eventb.core.label="axm1"
     org.eventb.core.predicate="..."
     org.eventb.core.source="/Lasso/LassoBaseContext.buc|org.eventb.core.contextFile#
        LassoBaseContext|org.eventb.core.axiom#("
     org.eventb.core.theorem="false"/>
  <org.eventb.core.scAxiom name="("
     org.eventb.core.label="axm2"
     org.eventb.core.predicate="..."
     org.eventb.core.source="/Lasso/LassoBaseContext.buc|org.eventb.core.contextFile#
        LassoBaseContext|org.eventb.core.axiom#)"
     org.eventb.core.theorem="false"/>
  <org.eventb.core.scConstant name="f"
     org.eventb.core.source="/Lasso/LassoBaseContext.buc|org.eventb.core.contextFile#
        LassoBaseContext|org.eventb.core.constant#'"
     org.eventb.core.type="..."/>
</org.eventb.core.scContextFile>
```

A.1.3 Rodin's BPO File Format

Associated with axioms and theorems are WD and THM proof obligations (see Section 6.3 for a discussion of these proof types). Proof information is stored in three files: `.bpo`, `.bpr` and `.bps`.

Proof obligations are stored in a `.bpo` file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<org.eventb.core.poFile org.eventb.core.poStamp="0">
  <org.eventb.core.poPredicateSet name="ABSHYP"
     org.eventb.core.poStamp="0">
    <org.eventb.core.poIdentifier name="f"
       org.eventb.core.type="..."/>
  </org.eventb.core.poPredicateSet>
  <org.eventb.core.poSequent name="axm2/WD"
     org.eventb.core.accurate="true"
     org.eventb.core.poDesc="Well-definedness of Axiom"
     org.eventb.core.poStamp="0">
    <org.eventb.core.poPredicateSet name="SEQHYP"
       org.eventb.core.parentSet="/Lasso/LassoBaseContext.bpo|org.eventb.core.poFile#
           LassoBaseContext|org.eventb.core.poPredicateSet#HYP'"/>
    <org.eventb.core.poPredicate name="SEQHYQ"
       org.eventb.core.predicate="..."
       org.eventb.core.source="/Lasso/LassoBaseContext.buc|org.eventb.core.contextFile#
           LassoBaseContext|org.eventb.core.axiom#)"/>
    <org.eventb.core.poSource name="SEQHYR"
       org.eventb.core.poRole="DEFAULT"
       org.eventb.core.source="/Lasso/LassoBaseContext.buc|org.eventb.core.contextFile#
           LassoBaseContext|org.eventb.core.axiom#)"/>
    <org.eventb.core.poSelHint name="SEQHYS"
       org.eventb.core.poSelHintFst="/Lasso/LassoBaseContext.bpo|org.eventb.core.poFile#
           LassoBaseContext|org.eventb.core.poPredicateSet#ABSHYP"
       org.eventb.core.poSelHintSnd="/Lasso/LassoBaseContext.bpo|org.eventb.core.poFile#
           LassoBaseContext|org.eventb.core.poPredicateSet#HYP'"/>
  </org.eventb.core.poSequent>
  <org.eventb.core.poPredicateSet name="HYP'"
     org.eventb.core.parentSet="/Lasso/LassoBaseContext.bpo|org.eventb.core.poFile#
        LassoBaseContext|org.eventb.core.poPredicateSet#ABSHYP"
     org.eventb.core.poStamp="0">
    <org.eventb.core.poPredicate name="PRD0"
       org.eventb.core.predicate="..."
       org.eventb.core.source="/Lasso/LassoBaseContext.buc|org.eventb.core.contextFile#
           LassoBaseContext|org.eventb.core.axiom#("/>
  </org.eventb.core.poPredicateSet>
  <org.eventb.core.poPredicateSet name="ALLHYP"
     org.eventb.core.parentSet="/Lasso/LassoBaseContext.bpo|org.eventb.core.poFile#
        LassoBaseContext|org.eventb.core.poPredicateSet#HYP'"
     org.eventb.core.poStamp="0">
    <org.eventb.core.poPredicate name="PRD1"
       org.eventb.core.predicate="..."
       org.eventb.core.source="/Lasso/LassoBaseContext.buc|org.eventb.core.contextFile#
           LassoBaseContext|org.eventb.core.axiom#)"/>
  </org.eventb.core.poPredicateSet>
</org.eventb.core.poFile>
```

A.1.4 Rodin's BPR File Format

Proof sequent information is stored in a `.bpr` file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<org.eventb.core.prFile version="1">
```

```
  <org.eventb.core.prProof name="axm2/WD"
     org.eventb.core.confidence="1000"
     org.eventb.core.prFresh=""
     org.eventb.core.prGoal="p0"
     org.eventb.core.prHyps="p1">
    <org.eventb.core.lang name="L"/>
    <org.eventb.core.prRule name="r0"
       org.eventb.core.confidence="1000"
       org.eventb.core.prDisplay="CVC3"
       org.eventb.core.prGoal="p0"
       org.eventb.core.prHyps="p1">
      <org.eventb.core.prString name=".arg"
         org.eventb.core.prSValue="R2000"/>
      <org.eventb.core.prString name=".config_id"
         org.eventb.core.prSValue="CVC3"/>
    </org.eventb.core.prRule>
    <org.eventb.core.prIdent name="f"
       org.eventb.core.type="..."/>
    <org.eventb.core.prPred name="p0"
       org.eventb.core.predicate="..."/>
    <org.eventb.core.prPred name="p1"
       org.eventb.core.predicate="..."/>
    <org.eventb.core.prReas name="r0"
       org.eventb.core.prRID="org.eventb.smt.core.externalSMT"/>
  </org.eventb.core.prProof>
</org.eventb.core.prFile>
```

A.1.5 Rodin's BPS File Format

Finally, the status of each proof is stored in a `.bps` file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<org.eventb.core.psFile>
      <org.eventb.core.psStatus name="axm2/WD"
                  org.eventb.core.confidence="1000"
                  org.eventb.core.poStamp="0"
                  org.eventb.core.psManual="false"/>
</org.eventb.core.psFile>
```

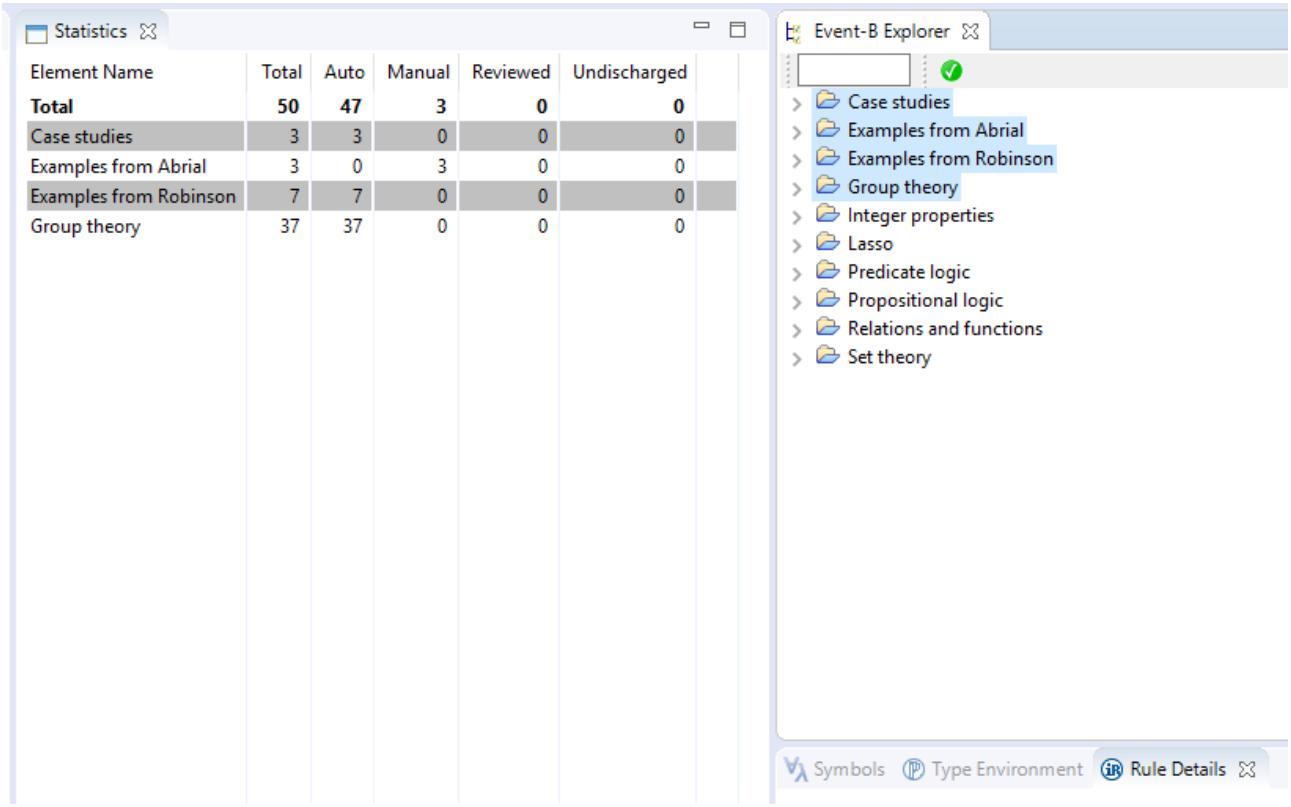
A.2 The Rodin Statistics Tab

Rodin includes a *Statistics* view which displays five different proof obligation discharge counts. The counts are rolled up based on the selected object in the *Event-B Explorer* view. They are (see Figure A.1):

- Total number of proofs;
- Number of automatically discharged proofs;
- Number of manually discharged proofs;
- Number of reviewed proofs; and
- Number of undischarged proofs.

We reiterate that these counts do not differentiate between WD and THM proofs. However, Table 8.3 in Section 8.2.2 shows a significant difference in the behaviour of the theorem provers when attempting WD as opposed

to THM proofs.



(a) Multiple Projects Selected

Figure A.1: The Rodin Statistics View

A.3 The C# Workspace Statistics Calculator

A.3.1 Screen Shots

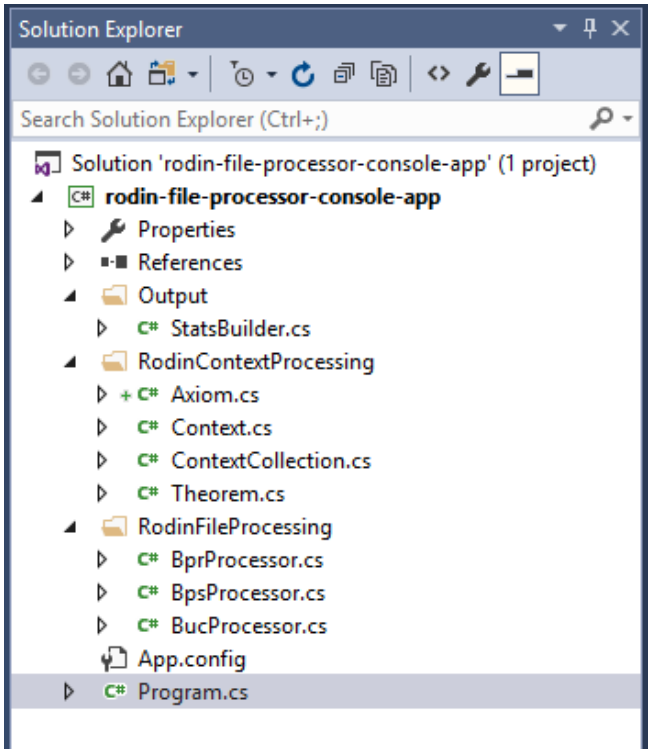


Figure A.2: The Rodin File Processor C# Project

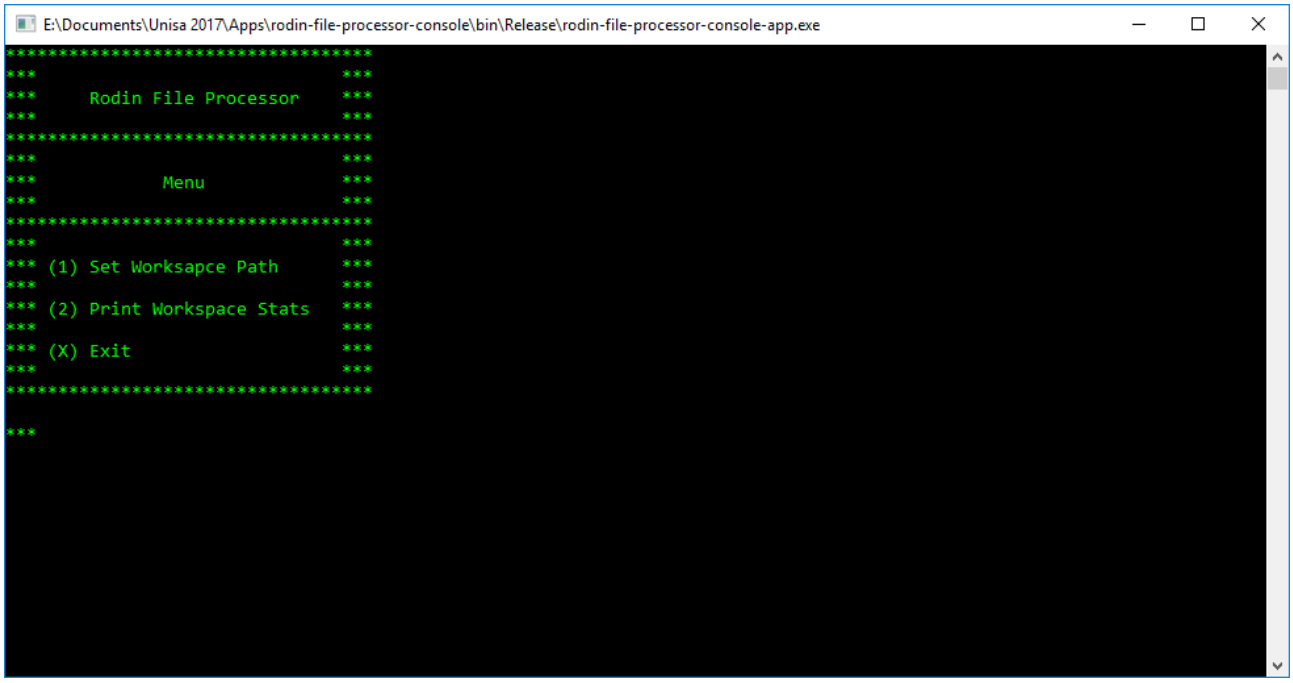


Figure A.3: The Rodin File Processor: Menu

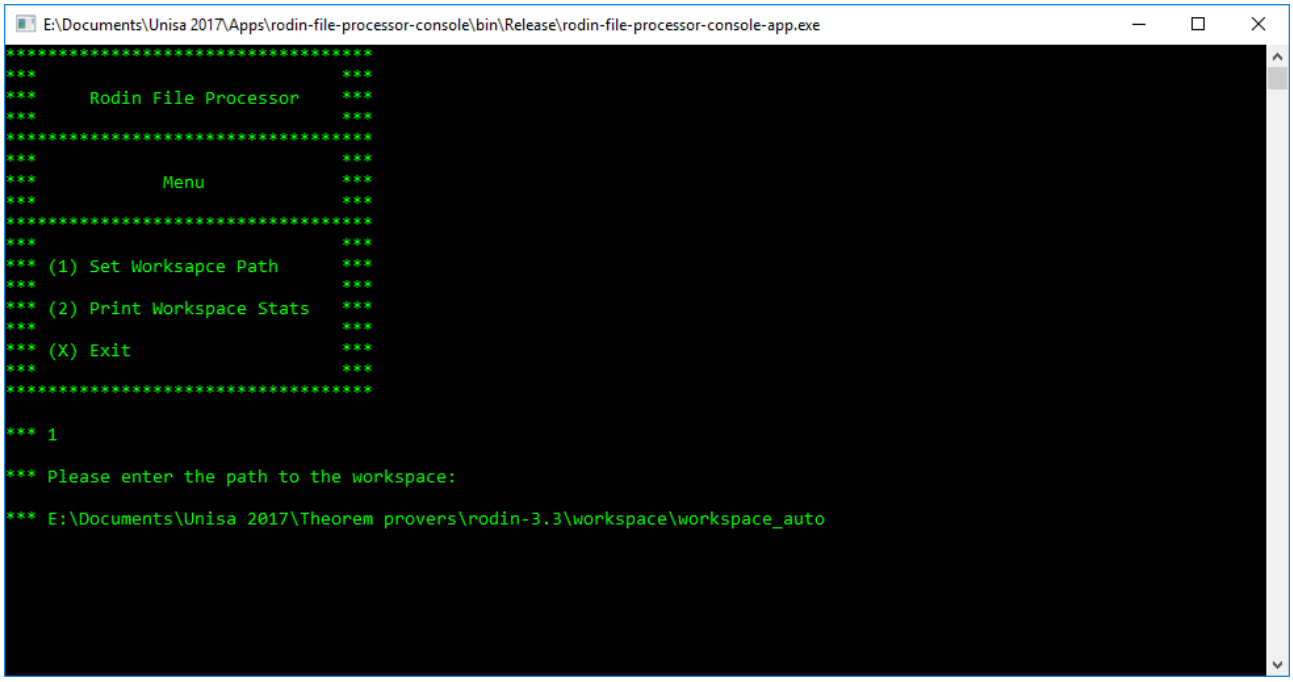


Figure A.4: The Rodin File Processor: Set Workspace Path

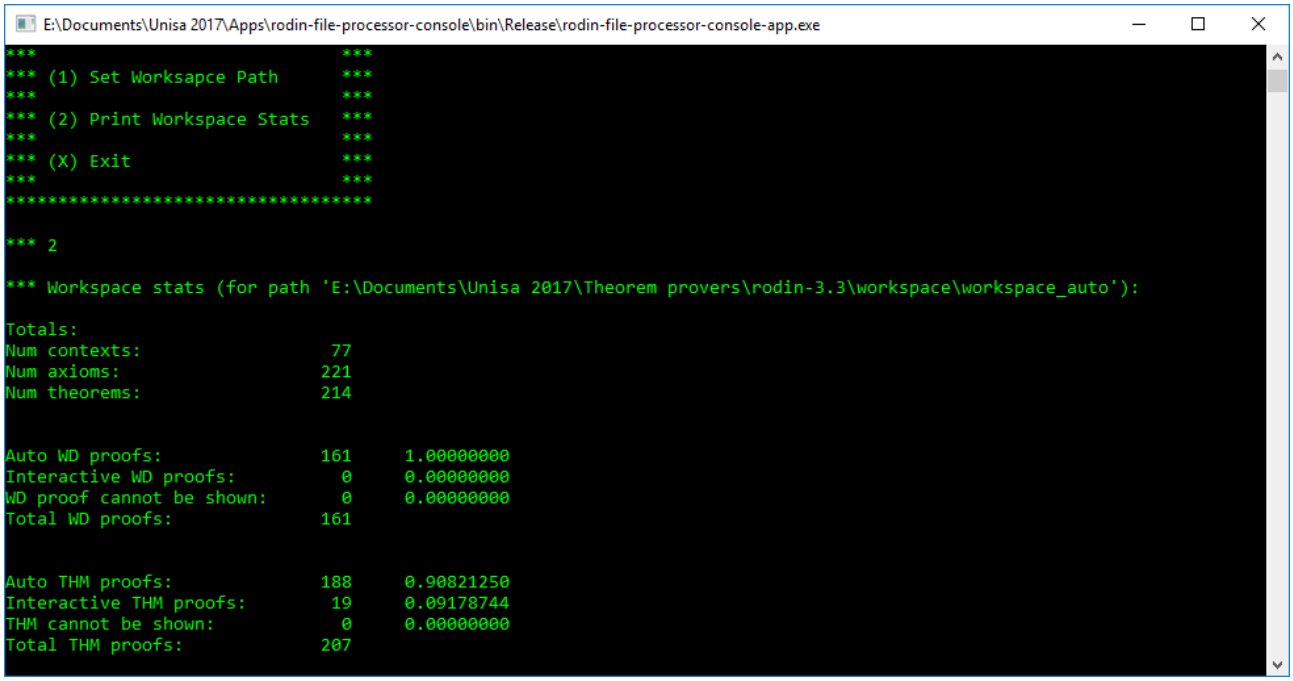


Figure A.5: The Rodin File Processor: Print Workspace Stats

A.3.2 C# Source Code

A.3.2.1 StatsBuilder.cs

```
using System.Text;

using RodinFileProcessor.ContextProcessing;

namespace RodinFileProcessor.Output {
  public static class StatsBuilder {
    private static int prefixWidth = 30;
```

```
private static int postfix1Width = 3;
private static int postfix2Width = 15;

public static string Print(ContextCollection contextCollection) {
  StringBuilder sb = new StringBuilder();

  int autoWdProofCount = 0;
  int autoThmProofCount = 0;
  int interactiveWdProofCount = 0;
  int interactiveThmProofCount = 0;
  int wdProofCannotBeShownCount = 0;
  int thmProofCannotBeShownCount = 0;
  int axmCount = 0;
  int thmCount = 0;

  foreach (Context context in contextCollection.Contexts) {
    autoWdProofCount += context.WdAutoProofCount;
    autoThmProofCount += context.ThmAutoProofCount;
    interactiveWdProofCount += context.WdInteractiveProofCount;
    interactiveThmProofCount += context.ThmInteractiveProofCount;
    wdProofCannotBeShownCount += context.WdProofCannotBeShown;
    thmProofCannotBeShownCount += context.ThmProofCannotBeShown;
    axmCount += context.NumAxioms;
    thmCount += context.NumTheorems;
  }

  sb.AppendLine("Totals:")
    .AppendFormat(getFixedWidthLine("Num contexts:", contextCollection.Contexts.Count))
        .AppendLine()
    .AppendFormat(getFixedWidthLine("Num axioms:", axmCount)).AppendLine()
    .AppendFormat(getFixedWidthLine("Num theorems:", thmCount)).AppendLine();

  sb.AppendLine().AppendLine();
  int totalWdProofs = autoWdProofCount + interactiveWdProofCount +
      wdProofCannotBeShownCount;

  float autoWdProofPercent = totalWdProofs == 0 ? 0 : autoWdProofCount / (float)
      totalWdProofs;
  sb.AppendFormat(getFixedWidthLine("Auto WD proofs:", autoWdProofCount,
      autoWdProofPercent))
    .AppendLine();
  float interactiveWdProofPercent =
    totalWdProofs == 0 ? 0 : interactiveWdProofCount / (float)totalWdProofs;
  sb.AppendFormat(getFixedWidthLine("Interactive WD proofs:", interactiveWdProofCount,
                            interactiveWdProofPercent)).AppendLine();
  float wdCannotBeShownPercent =
    totalWdProofs == 0 ? 0 : wdProofCannotBeShownCount / (float)totalWdProofs;
  sb.AppendFormat(getFixedWidthLine("WD proof cannot be shown:",
      wdProofCannotBeShownCount,
                            wdCannotBeShownPercent)).AppendLine();
  sb.AppendFormat(getFixedWidthLine("Total WD proofs:", totalWdProofs)).AppendLine();

  sb.AppendLine().AppendLine();

  int totalThmProofs = autoThmProofCount + interactiveThmProofCount +
      thmProofCannotBeShownCount;

  float autoThmProofPercent = totalThmProofs == 0 ? 0 : autoThmProofCount / (float)
      totalThmProofs;
  sb.AppendFormat(getFixedWidthLine("Auto THM proofs:", autoThmProofCount,
      autoThmProofPercent))
    .AppendLine();

  float interactiveThmProofPercent =
    totalThmProofs == 0 ? 0 : interactiveThmProofCount / (float)totalThmProofs;
  sb.AppendFormat(getFixedWidthLine("Interactive THM proofs:", interactiveThmProofCount
```

```
        ,
                                interactiveThmProofPercent)).AppendLine();

      float thmCannotBeShownPercent =
        totalThmProofs == 0 ? 0 : thmProofCannotBeShownCount / (float)totalThmProofs;
      sb.AppendFormat(getFixedWidthLine("THM cannot be shown:", thmProofCannotBeShownCount,
                                thmCannotBeShownPercent)).AppendLine();

      sb.AppendFormat(getFixedWidthLine("Total THM proofs:", totalThmProofs)).AppendLine();

      return sb.ToString();
    }

    private static string getFixedWidthLine(string prefix, string postfix) {
      return prefix.PadRight(prefixWidth, ' ') + postfix.PadLeft(postfix1Width, ' ');
    }

    private static string getFixedWidthLine(string prefix, int postfix) {
      return getFixedWidthLine(prefix, postfix.ToString());
    }

    private static string getFixedWidthLine(string prefix, int postfix1, float postfix2) {
      return prefix.PadRight(prefixWidth, ' ') + postfix1.ToString().PadLeft(postfix1Width,
          ' ') +
           postfix2.ToString("0.00000000").PadLeft(postfix2Width, ' ');
    }
  }
}
```

A.3.2.2 Axiom.cs

```
namespace RodinFileProcessor.ContextProcessing {
  public class Axiom {
    // useful when debugging
    public string Name { get; }

    public bool WdHasProof { get; set; }
    public bool WdHasAutoProof { get; set; }
    public bool WdProofCannotBeShown { get; set; }

    public bool ThmHasProof { get; set; }
    public bool ThmHasAutoProof { get; set; }
    public bool ThmProofCannotBeShown { get; set; }

    public virtual bool IsAxiom => true;
    public virtual bool IsTheorem => false;

    public Axiom(string name) {
      Name = name;
    }
  }
}
```

A.3.2.3 Context.cs

```
using System.Collections.Generic;
using System.IO;
using System.Linq;

using RodinFileProcessor.RodinFileProcessing;

namespace RodinFileProcessor.ContextProcessing {
  public class Context {
```

```
    private readonly BucProcessor bucFileProcessor;
    private readonly BpsProcessor bpsFileProcessor;
    private readonly BprProcessor bprFileProcessor;

    private List<Axiom> axioms { get; set; }

    public string BucFilename => bucFileProcessor.Filename;

    public int WdAutoProofCount => axioms.Count(x => x.WdHasAutoProof);
    public int ThmAutoProofCount => axioms.Count(x => x.IsTheorem && x.ThmHasAutoProof);
    public int WdInteractiveProofCount => axioms.Count(x => x.WdHasProof && !x.
        WdHasAutoProof);
    public int ThmInteractiveProofCount => axioms.Count(x => x.ThmHasProof && !x.
        ThmHasAutoProof);
    public int WdProofCannotBeShown => axioms.Count(x => x.WdProofCannotBeShown);
    public int ThmProofCannotBeShown => axioms.Count(x => x.ThmProofCannotBeShown);

    public int NumAxioms => axioms.Count(x => x.IsAxiom);
    public int NumTheorems => axioms.Count(x => x.IsTheorem);

    public bool IsValid => bucFileProcessor != null && bucFileProcessor.IsValid;

    public Context(string filename) {
      bucFileProcessor = new BucProcessor(filename);
      bpsFileProcessor =
        new BpsProcessor(Path.Combine(Path.GetDirectoryName(filename),
                                Path.GetFileNameWithoutExtension(filename)) + ".bps");
      bprFileProcessor =
        new BprProcessor(Path.Combine(Path.GetDirectoryName(filename),
                                Path.GetFileNameWithoutExtension(filename)) + ".bpr");

      processAxioms();
    }

    private void processAxioms() {
      axioms = new List<Axiom>();

      foreach (string axiomName in bucFileProcessor.AxiomNames) {
        Axiom axiom;
        if (bucFileProcessor.IsTheorem[axiomName]) {
          axiom = new Theorem(axiomName);
          bool hasProofThm = bpsFileProcessor.ThmHasProof(axiomName) &&
                        !bprFileProcessor.GetProofCannotBeShown(axiomName, "THM");
          axiom.ThmProofCannotBeShown = bprFileProcessor.GetProofCannotBeShown(axiomName, "
              THM");
          axiom.ThmHasProof = hasProofThm && !axiom.ThmProofCannotBeShown;
          axiom.ThmHasAutoProof = hasProofThm && bpsFileProcessor.ThmHasAutoProof(axiomName)
              ;
        }
        else {
          axiom = new Axiom(axiomName);
        }

        bool hasProofWd = bpsFileProcessor.WdHasProof(axiomName) &&
                      !bprFileProcessor.GetProofCannotBeShown(axiomName, "WD");
        axiom.WdProofCannotBeShown = bprFileProcessor.GetProofCannotBeShown(axiomName, "WD"
            );
        axiom.WdHasProof = hasProofWd && !axiom.WdProofCannotBeShown;
        axiom.WdHasAutoProof = hasProofWd && bpsFileProcessor.WdHasAutoProof(axiomName);

        axioms.Add(axiom);
      }
    }
  }
}
```

A.3.2.4 ContextCollection.cs

```
using System.Collections.Generic;
using System.IO;
using System.Linq;

using RodinFileProcessor.Output;

namespace RodinFileProcessor.ContextProcessing {
  public class ContextCollection {
    public List<Context> Contexts { get; }

    public ContextCollection(string path) {
      Contexts = new List<Context>();

      List<string> filesToProcess = new List<string>();
      if (File.Exists(path)) {
        if (Path.GetExtension(path).ToLower() != ".buc") {
          path = Path.Combine(Path.GetDirectoryName(path), Path.GetFileNameWithoutExtension(
              path)) + ".buc";
        }

        if (File.Exists(path)) {
          filesToProcess.Add(path);
        }
      }

      try {
        if (filesToProcess.Count == 0 && Directory.Exists(path)) {
          string[] files = Directory.GetFiles(path, "*.buc", SearchOption.AllDirectories);
          filesToProcess = files.ToList();
        }
      }
      catch {
        // authorization errors etc.
      }

      foreach (string filename in filesToProcess) {
        var newContext = new Context(filename);
        if (!newContext.IsValid) continue;

        Contexts.RemoveAll(x => x.BucFilename.ToLower() == newContext.BucFilename.ToLower()
            );
        Contexts.Add(newContext);
      }
    }

    public override string ToString() {
      return StatsBuilder.Print(this);
    }
  }
}
```

A.3.2.5 Theorem.cs

```
namespace RodinFileProcessor.ContextProcessing {
  public class Theorem : Axiom {
    public override bool IsAxiom => false;
    public override bool IsTheorem => true;

    public Theorem(string name)
      : base(name) { }
  }
```

```
}
```

A.3.2.6 BprProcessor.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Xml.Linq;

namespace RodinFileProcessor.RodinFileProcessing {
  public class BprProcessor {
    private const string OrgEventbCorePrproof = "org.eventb.core.prProof";
    private const string OrgEventbCoreConfidence = "org.eventb.core.confidence";
    private const string OrgEventbCorePrrule = "org.eventb.core.prRule";
    private const string OrgEventbCorePrante = "org.eventb.core.prAnte";

    // key: axm name, value: true if proof is not complete, i.e. not shown
    private readonly List<string> proofCannotBeShown = new List<string>();

    public BprProcessor(string filename) {
      if (!Path.GetExtension(filename).ToLower().Contains("bpr")) {
        return;
      }

      XDocument xdoc;
      try {
        xdoc = XDocument.Load(filename);
      }
      catch {
        return;
      }

      foreach (XElement el in xdoc.Root.Elements(OrgEventbCorePrproof)) {
        string name = el.Attribute("name").Value;
        int confidence = 0;

        var confAttr = el.Attribute(OrgEventbCoreConfidence);
        if (confAttr != null) {
          confidence = System.Convert.ToInt32(el.Attribute(OrgEventbCoreConfidence).Value);
        }

        recurseProof(el.Element(OrgEventbCorePrrule), name, confidence);
      }
    }

    public bool GetProofCannotBeShown(string axiomName, string proofType) {
      if (proofCannotBeShown.Contains(axiomName + "/" + proofType)) {
        return true;
      }

      return false;
    }

    private void recurseProof(XElement proofBranch, string proofName, int confidence) {
      if (proofBranch != null) {
        confidence = Math.Min(
          confidence,
          Convert.ToInt32(proofBranch.Attribute(OrgEventbCoreConfidence).Value));
      }

      if (proofBranch == null || confidence == 0) {
        // no rule info, "Cannot be shown" case
        // tree ended on an ante, not a rule element
        proofCannotBeShown.Add(proofName);
        return;
```

```
    }

    var children = proofBranch.Elements(OrgEventbCorePrante);
    foreach (XElement rule in children) {
     recurseProof(rule.Element(OrgEventbCorePrrule), proofName, confidence);
    }
   }
  }
}
```

A.3.2.7 BpsProcessor.cs

```
using System.Collections.Generic;
using System.IO;
using System.Xml.Linq;

namespace RodinFileProcessor.RodinFileProcessing {
 public class BpsProcessor {
  private const string orgEventbCorePsstatus = "org.eventb.core.psStatus";
  private const string orgEventbCorePsmanual = "org.eventb.core.psManual";

  private Dictionary<string, bool> IsAutoProof { get; }

  public BpsProcessor(string filename) {
   IsAutoProof = new Dictionary<string, bool>();

   if (!Path.GetExtension(filename).ToLower().Contains("bps")) {
    return;
   }

   // .bps may not exist!
   if (!File.Exists(filename)) {
    return;
   }

   XDocument xdoc = XDocument.Load(filename);

   foreach (XElement el in xdoc.Root.Elements(orgEventbCorePsstatus)) {
    string name = el.Attribute("name").Value;
    bool autoProved = el.Attribute(orgEventbCorePsmanual).Value == "false";

    IsAutoProof.Add(name, autoProved);
   }
  }

  public bool WdHasProof(string axmName) {
   return hasProof(axmName, "WD");
  }

  public bool WdHasAutoProof(string axmName) {
   return hasAutoProof(axmName, "WD");
  }

  public bool ThmHasProof(string axmName) {
   return hasProof(axmName, "THM");
  }

  public bool ThmHasAutoProof(string axmName) {
   return hasAutoProof(axmName, "THM");
  }

  private bool hasProof(string axmName, string proofType) {
   foreach (string proofName in IsAutoProof.Keys) {
    if (proofName == axmName + "/" + proofType) {
     return true;
```

```
      }
     }

     return false;
    }

    private bool hasAutoProof(string axmName, string proofType) {
     foreach (string proofName in IsAutoProof.Keys) {
      if (proofName == axmName + "/" + proofType) {
       return IsAutoProof[proofName];
      }
     }

     return false;
    }
   }
}
```

A.3.2.8 BucProcessor.cs

```
using System.Collections.Generic;
using System.IO;
using System.Xml.Linq;

namespace RodinFileProcessor.RodinFileProcessing {
 public class BucProcessor {
  private const string FileExtension = "buc";
  private const string OrgEventbCoreAxiom = "org.eventb.core.axiom";
  private const string OrgEventbCoreLabel = "org.eventb.core.label";
  private const string OrgEventbCoreTheorem = "org.eventb.core.theorem";

  public string Filename { get; }
  public bool IsValid { get; }

  public string ContextName { get; }
  public List<string> AxiomNames { get; }
  public Dictionary<string, bool> IsTheorem { get; } // otherwise Axiom

  public BucProcessor(string filename) {
   Filename = filename.Trim();

   AxiomNames = new List<string>();
   IsTheorem = new Dictionary<string, bool>();

   if (!Filename.EndsWith(FileExtension)) {
    return;
   }

   IsValid = true;

   ContextName = Path.GetFileNameWithoutExtension(Filename);

   XDocument xdoc = XDocument.Load(Filename);

   foreach (XElement el in xdoc.Root.Elements(OrgEventbCoreAxiom)) {
    string name = el.Attribute(OrgEventbCoreLabel).Value;
    AxiomNames.Add(name);

    // axiom or theorem
    bool isTheorem = el.Attribute(OrgEventbCoreTheorem) != null
                &&
                el.Attribute(OrgEventbCoreTheorem).Value == "true";

    IsTheorem.Add(name, isTheorem);
   }
```

```
    }
  }
}
```

A.3.2.9 Program.cs

```
using System;
using System.IO;
using System.Text;

using RodinFileProcessor.ContextProcessing;

namespace RodinFileProcessor {
  internal static class Program {
    private static string workspaceDirectory;
    private static ContextCollection contextCollection;

    public static void Main() {
      do {
        var menuOptionStr = readMenuOption(out var exitApp);

        if (exitApp) break;

        processMenuOption(menuOptionStr);
      } while (true);
    }

    private static string readMenuOption(out bool exitApp) {
      printMenu();

      string menuOptionStr = readUserInput();

      exitApp = menuOptionStr.Trim().ToLower() == "x";

      return menuOptionStr;
    }

    private static void printMenu() {
      StringBuilder sb = new StringBuilder();

      sb.AppendLine("**********************************")
        .AppendLine("*** ***")
        .AppendLine("*** Rodin File Processor ***")
        .AppendLine("*** ***")
        .AppendLine("**********************************")
        .AppendLine("*** ***")
        .AppendLine("*** Menu ***")
        .AppendLine("*** ***")
        .AppendLine("**********************************")
        .AppendLine("*** ***")
        .AppendLine("*** (1) Set Worksapce Path ***")
        .AppendLine("*** ***")
        .AppendLine("*** (2) Print Workspace Stats ***")
        .AppendLine("*** ***")
        .AppendLine("*** (X) Exit ***")
        .AppendLine("*** ***")
        .AppendLine("**********************************")
        .AppendLine();

      Console.Write(sb.ToString());
    }

    private static void processMenuOption(string menuOptionStr) {
      int menuOption;
      try {
```

```
      menuOption = Convert.ToInt32(menuOptionStr);
    }
    catch {
      menuOption = -1;
    }

    switch (menuOption) {
      case 1:
        processWorkspacePath();
        break;

      case 2:
        printWorkspaceStats();
        break;

      default:
        printMessage("Invalid selection, please try again");
        break;
    }
  }

  private static void printMessage(string message) {
    Console.Write("*** ");
    Console.WriteLine(message);
    Console.WriteLine();
  }

  private static string readUserInput() {
    Console.Write("*** ");
    var returnVal = Console.ReadLine() ?? "";
    Console.WriteLine();
    return returnVal;
  }

  private static void processWorkspacePath() {
    printMessage("Please enter the path to the workspace:");

    var oldValue = workspaceDirectory;
    workspaceDirectory = readUserInput();

    if (!Directory.Exists(workspaceDirectory)) {
      printMessage("Invalid path: path does not exist");
      workspaceDirectory = oldValue;
    }
  }

  private static void printWorkspaceStats() {
    contextCollection = new ContextCollection(workspaceDirectory);

    var contextStats = contextCollection?.ToString();
    if (string.IsNullOrWhiteSpace(contextStats)) {
      printMessage("The path you have specified contains no Rodin files to analyze.");
      return;
    }

    printMessage($"Workspace stats (for path '{workspaceDirectory}'):");
    Console.WriteLine(contextStats);
  }
 }
}
```

Appendix B

Rodin Screen Shots

An in-depth discussion of the Rodin tool suite was presented in Chapter 6, but the focus was on the Event-B modelling language as well as Rodin's inference mechanism. In this appendix, we include a number of screen shots of Rodin's user interface. For a detailed discussion of Rodin's user interface, see (Jastram 2012).

B.1 The Rodin User Interfaces

The current section includes two examples of typical user interface screens one would see when creating and editing contexts, and discharging proof obligations.

Figure B.1 shows the *Event-B context editor* and is part of Rodin's *modelling UI*. This interface can be used to create and edit Rodin contexts. The most significant functions of the interface are the '+' signs next to the following headings:

- **EXTENDS**: Allows the user to select another context in the same project that will be extended in the current context;
- **SETS**: Allows the user to introduce an additional carrier set (see Section 6.4.1);
- **CONSTANTS**: Allows the user to introduce a global symbol (constant) that is within the scope of all the axioms and theorems in the AXIOMS section, as well as in the scope of all derived contexts' axioms; and
- **AXIOMS**: Allows the user to add axioms and theorems to the context.

Output from the *static checker* (see Section 6.5.1) is added to the *Rodin Problems* tab and clicking on a symbol on the *Symbols* tab will inject the selected symbol in the selected input control.

Finally, the *Event-B explorer* on the left provides various options to create and manipulate Event-B projects (see Appendix A.1) and contexts.

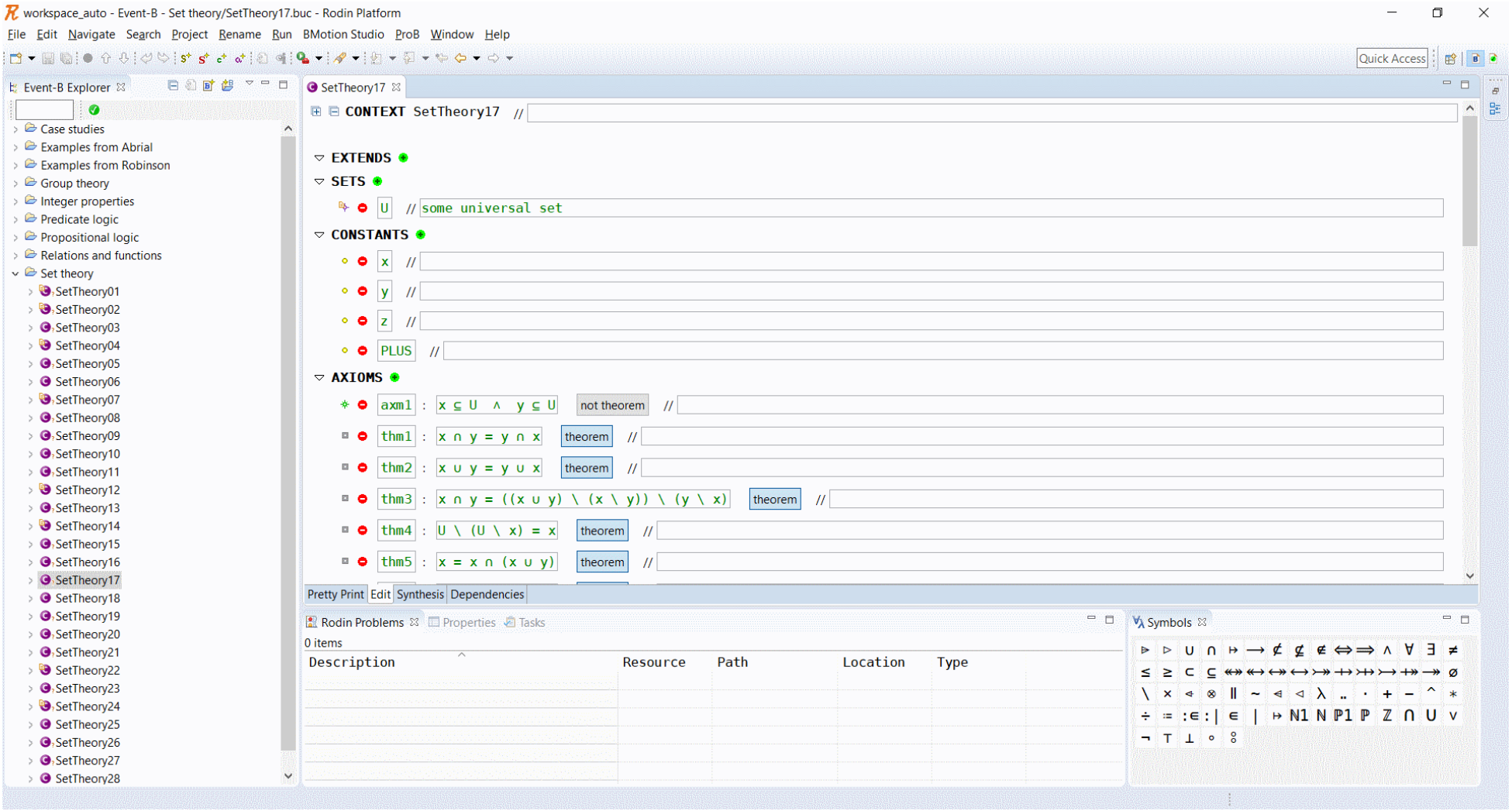


Figure B.1: The Modelling User Interface

Figure B.2 is part of Rodin's *proving UI*. This part of the user interface can be used to inspect and manipulate proof trees, and discharge proof obligations interactively.

The screen is divided into five sections, namely

- The **Proof Tree**: The user can use this control to inspect the proof tree or to manipulate the proof tree. For example, a typical action is to prune the proof tree at the selected proof tree node to attempt a different interactive strategy to discharge a proof obligation;
- The **Selected Hypotheses**: This control lists the hypotheses for the proof tree node that is selected in the *Proof Tree* control. Text in red indicates the availability of a rewrite rule that can be applied. Note that the list of hypotheses can be empty;
- The **Goal**: This control displays the goal for the proof tree node that is selected in the *Proof Tree* control;
- The **Proof Control**: The user can use this control to invoke a specific theorem prover, apply the default auto tactic profile, apply the default post tactic profile or apply *lasso*. The selected action is always performed for the proof tree node that is selected in the *Proof Tree* control; and
- The **Event-B Explorer**: This control provides various options to create and manipulate Event-B projects (see Appendix A.1) and contexts.

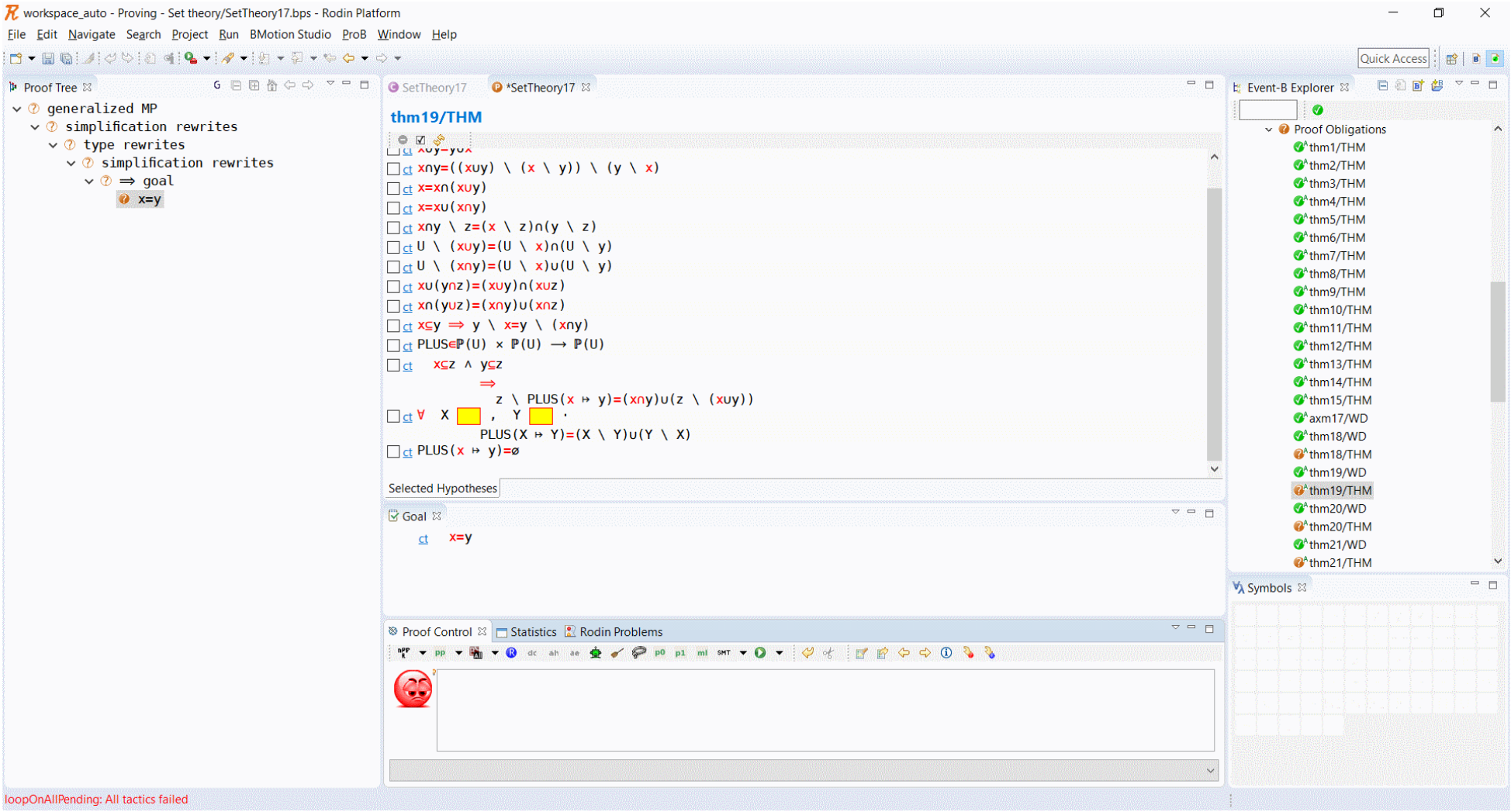


Figure B.2: The Proving User Interface

B.2 Rodin Auto-tactic Profiles

This section includes screen shots of the *Default Auto/Post Tactic Profile* selector and the tactic profile editor. Tactic profiles are discussed in Section 6.11.

Figure B.3 shows the user interface for selecting an auto tactic profile as well as options for enabling and disabling auto tactic profiles.

Figure B.4 shows the user interface for creating a custom tactic profile. Once created, the profile name can be selected from the list of available tactic profiles in Figure B.3.

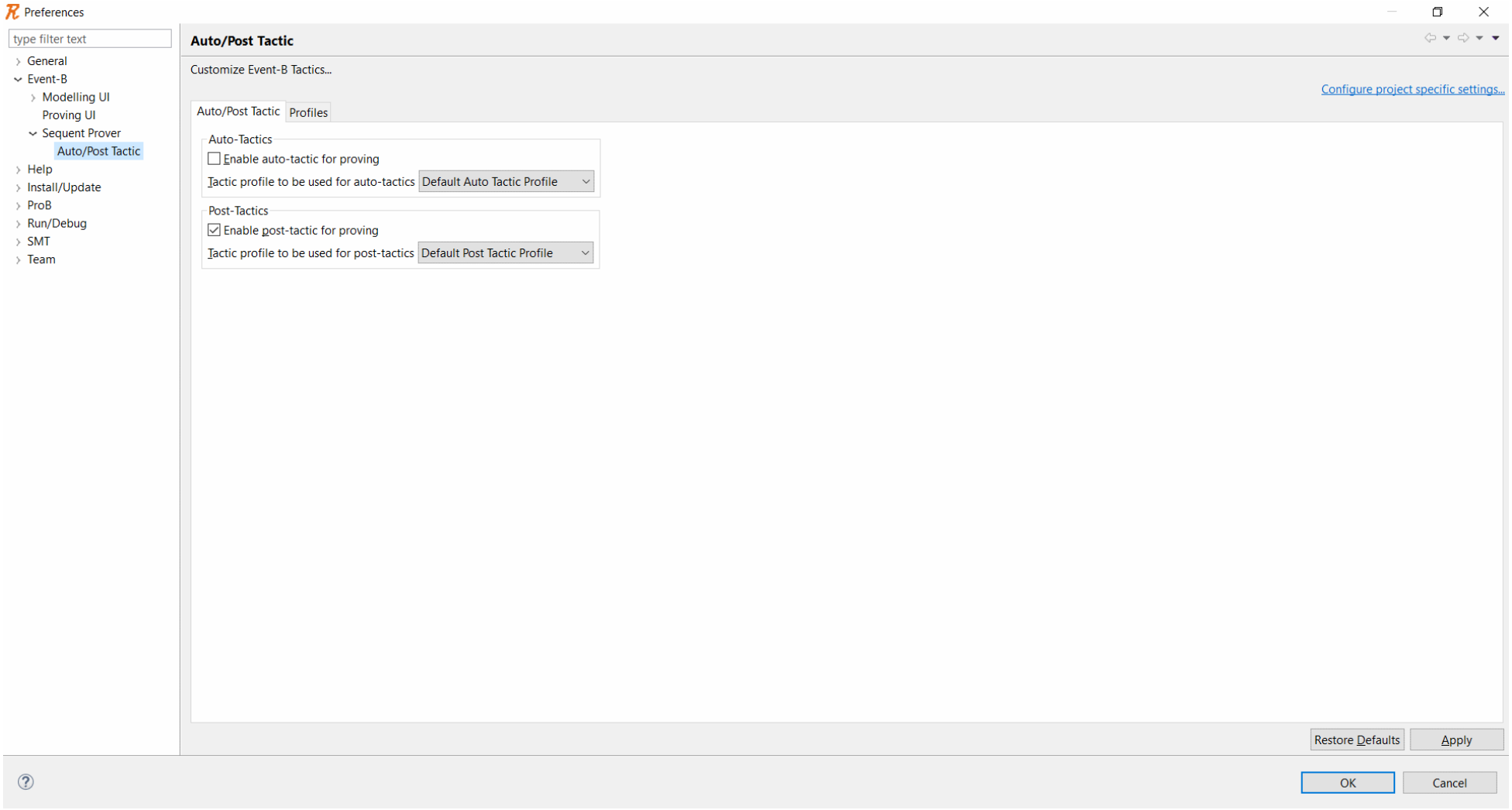


Figure B.3: The Auto/Post Tactic Selector

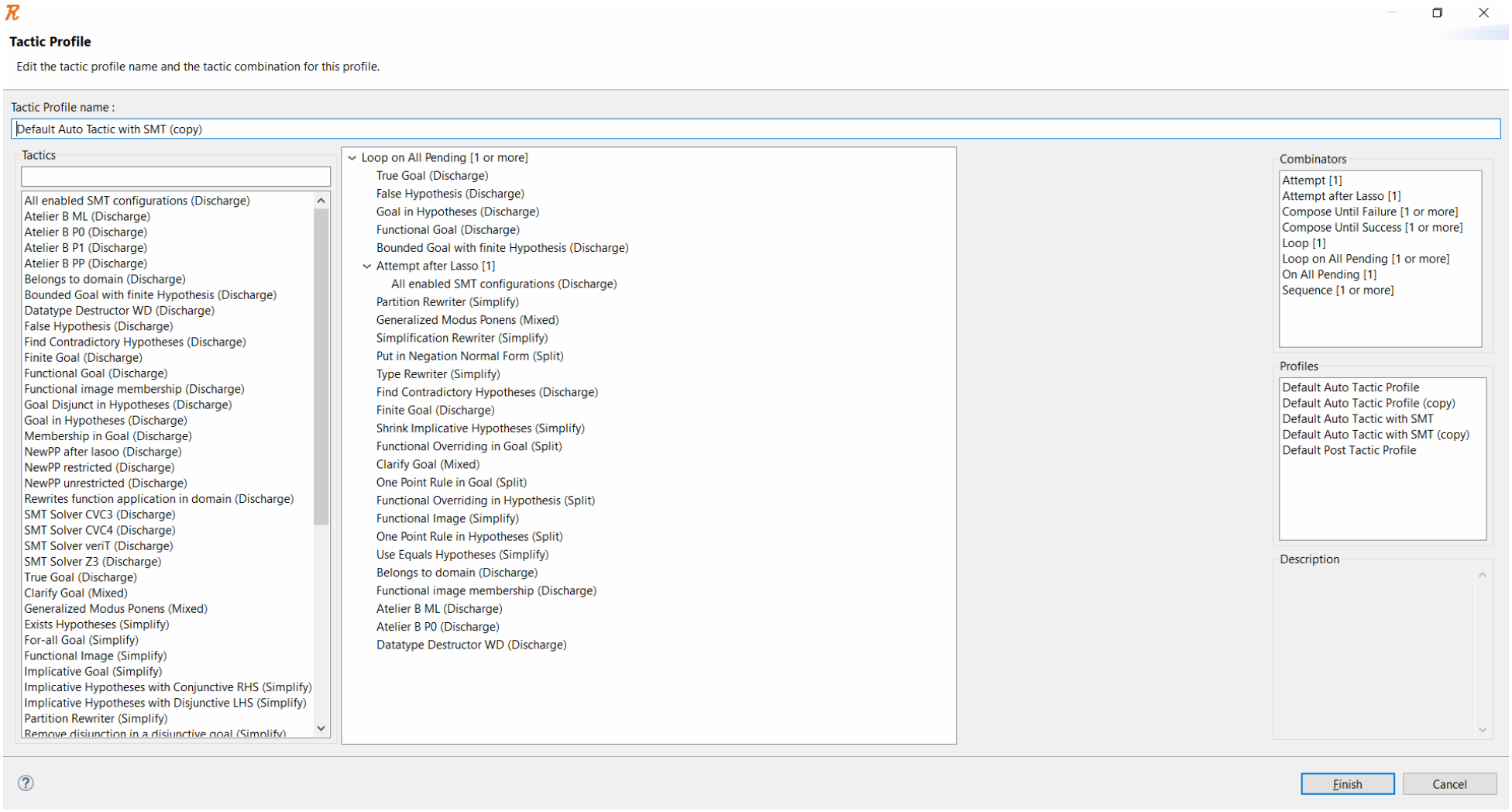


Figure B.4: The Tactic Profile Editor

Appendix C

Rodin Contexts

C.1 Introduction

A total of 75 Rodin contexts, consisting of 215 axioms and 215 theorems, are included in this Appendix. Many of the contexts listed here are referenced in Chapters 7, 8 and 9. All the Rodin contexts listed here were included in the calculation of the execution times of the various Rodin theorem proving strategies discussed in Sections 8.1 and 8.2.

Two tables which show the proof obligation status (automatically discharged, interactively discharged or not discharged) for each of the Rodin WD or THM proof obligations are included in Appendix D. Table D.1 includes the proof status information for the Default Auto Tactic Profile. Table D.2 includes the proof status information for the Default Auto Tactic Profile with SMT ($0.1s$timeout).

The Rodin contexts were converted to LaTeX using the Rodin plug-in "Event B to LaTeX exporter" (version 0.7.0.201607081834) provided by the University of Southampton. Note that comments in Rodin contexts appear in green once exported.

C.2 Case Study

The context in this section is an implementation of the simple telephone network specified by Morgan (1993). The state invariants are expressed as axioms and one of the consequences of the invariant of schema ΔTN is explored in `thm13`.

C.2.1 Telephone Network

Referenced in Section 9.1.2.

CONTEXT TelephoneNetwork

DESCRIPTION

The following example appears in Morgan (1993, p. 29) where a Z specification is presented for a simple telephone network.
The invariants of the Z schemas are included.
Finally, `thm13` explores a consequence of the invariant of schema ΔTN.

`axm1` - `axm4` : Declarations of *CON*, *reqs*, *reqsPrime*, *cons*, *consPrime* and *disjoint*
`axm5` : Definition of *disjoint*
`axm6` - `axm8` : *efficientTN* invariants
`axm9` - `axm11` : $efficientTN'$ invariants
`axm12` : ΔTN invariant
`thm13` : state property implied by the ΔTN invariant

SETS

PHONE e.g., PHONE = {0, 1, 2, 3, ...} = NAT

CONSTANTS

CON

reqs

cons

reqsPrime

consPrime

disjoint

AXIOMS

`axm1`: $CON = \mathbb{P}(PHONE)$
e.g., CON = {{}, {0}, {1}, ..., {0,1}, {3,4},...}

`axm2`:
$reqs \in \mathbb{P}(CON)$
$\land$
$reqsPrime \in \mathbb{P}(CON)$
e.g., reqs = {{{0,1}, {3,4},...}, {{0}, {1}}, ...}

`axm3`:
$cons \in \mathbb{P}(CON)$
$\land$
$consPrime \in \mathbb{P}(CON)$

`axm4`: $disjoint \in \mathbb{P}(\mathbb{P}(CON))$

`axm5`:
$\forall X \cdot X \in disjoint$
$\Rightarrow$
$(\forall x1, x2 \cdot x1 \in X \land x2 \in X$
$\Rightarrow (x1 \neq x2 \Rightarrow x1 \cap x2 = \varnothing))$

`axm6`: $cons \subseteq reqs$
efficientTN 1

`axm7`: $cons \in disjoint$
efficientTN 2

`axm8`:
$\neg (\exists cons0 \cdot cons0 \in \mathbb{P}(CON)$
$\land\ cons \subset cons0$
$\land\ cons0 \subseteq reqs$
$\land\ cons0 \in disjoint)$
efficientTN 3

axm9: $consPrime \subseteq reqsPrime$
efficientTNprime 1

axm10: $consPrime \in disjoint$
efficientTNprime 2

axm11:
$\neg\ (\exists cons0 \cdot cons0 \in \mathbb{P}(CON)$
$\land\ consPrime \subset cons0$
$\land\ cons0 \subseteq reqsPrime$
$\land\ cons0 \in disjoint)$
efficientTNprime 3

axm12:
$\neg\ (\exists cons1 \cdot cons1 \in \mathbb{P}(CON)$
$\land\ cons \setminus cons1 \subset cons \setminus consPrime$
$\land\ ($
$\neg\ (\exists cons0 \cdot cons0 \in \mathbb{P}(CON)$
$\land\ cons1 \subset cons0$
$\land\ cons0 \subseteq reqsPrime$
$\land\ cons0 \in disjoint)$
$))$
delta TN

thm13: ⟨theorem⟩ $\forall x \cdot x \in (cons \cap reqsPrime) \Rightarrow x \in consPrime$

END

C.3 Example from Abrial

The context in this section implements the properties of the irreflexive transitive closure given by Abrial (2010, p. 335). Quantifier instantiation is required to discharge the proof obligations.

C.3.1 From Abrial 1

Referenced in Section 7.2.3.

CONTEXT FromAbrial01

DESCRIPTION

The following example appears in Abrial (2010, p. 335).
Let r be a relation on a set S.
Then the irreflexive transitive closure of r, denoted by $cl(r)$, is also a relation on S.
The relation $cl(r)$ has the following properties:
(1) r is included in $cl(r)$;
(2) The forward composition of $cl(r)$ with r is in $cl(r)$; and
(3) The relation $cl(r)$ is the smallest relation satisfying (1) and (2).

axm1 - axm5 : Declaration of S and r; declaration and definition of $cl(r)$
thm6 : The forward composition of $cl(r)$ with itself is in $cl(r)$
thm7 : $cl(r)$ is the union of r and the forward composition of r and $cl(r)$
thm8 : $cl(r)$ is the union of r and the forward composition of $cl(r)$ and r

SETS

S

CONSTANTS

cl_r

r

AXIOMS

axm1: $r \in S \leftrightarrow S$

axm2: $cl\_r \in S \leftrightarrow S$

axm3: $r \subseteq cl\_r$

axm4: $cl\_r;\ r \subseteq cl\_r$

axm5: $\forall p \cdot r \subseteq p \wedge p;\ r \subseteq p \Rightarrow cl\_r \subseteq p$

thm6: ⟨theorem⟩ $cl\_r;\ cl\_r \subseteq cl\_r$

inst axm5: {x↦ y | cl_r;{x↦ y} ⊆ cl_r}

thm7: ⟨theorem⟩ $cl\_r = r \cup (r;\ cl\_r)$

inst axm5: {x↦ y | {x↦ y} ⊆ r ∪ (r;cl_r)}

thm8: ⟨theorem⟩ $cl\_r = r \cup (cl\_r;\ r)$

inst axm5: {x↦ y | {x↦ y} ⊆ r ∪ (cl_r;r)}

END

C.4 Group Theory

Basic group theoretic properties are explored in this section. A number of structures $\langle G, Op \rangle$ are shown to either be groups or exhibit certain group theoretic properties.

C.4.1 Group Theory 1

Referenced in Section 8.4.2.

CONTEXT GroupTheory01

DESCRIPTION

Let $G \subseteq \mathbb{Z}$ and $Op \in G \times G \rightarrow \mathbb{Z}$.
The four conditions (also called the *group axioms*) that $\langle G, Op \rangle$ must satisfy to be a group, are declared and defined.
The *group axioms* are:
(1) **Closure** - $g1, g2 \in G \ \Rightarrow\ Op(g1, g2) \in G$
(2) **Associativity** - $g1, g2, g3 \in G \ \Rightarrow\ Op(Op(g1, g2), g3) = Op(g1, Op(g2, g3))$
(3) **Identity element** - There exists $e \in G$ such that $Op(g, e) = g = Op(e, g)$ for all $g \in G$
(4) **Inverse element** - For every $g \in G$ there exists $g^{-1} \in G$ such that $Op(g, g^{-1}) = e = Op(g^{-1}, g)$

axm1 - axm2 : Set G, function Op and $e \in G$ are declared
axm3 - axm7 : Five functions are declared: *HasClosure*, *IsAssoc*, *HasId*, *HasInverse* and *IsGroup*
axm8 : Function *HasClosure* defined
axm9 : Function *IsAssoc* defined
axm10 : Function *HasId* defined
axm11 : Function *HasInverse* defined
axm12 : Function *IsGroup* defined

CONSTANTS

G

Op

e

HasClosure

IsAssoc

HasId

HasInverse

IsGroup

AXIOMS

axm1: $G \subseteq \mathbb{Z} \land Op \in G \times G \to \mathbb{Z}$

axm2: $e \in G$

axm3: $HasClosure \in (G \times G \to \mathbb{Z}) \to BOOL$

axm4: $IsAssoc \in (G \times G \to \mathbb{Z}) \to BOOL$

axm5: $HasId \in (G \times G \to \mathbb{Z}) \to BOOL$

axm6: $HasInverse \in (G \times G \to \mathbb{Z}) \to BOOL$

axm7: $IsGroup \in (\mathbb{P}(\mathbb{Z}) \times (G \times G \to \mathbb{Z})) \to BOOL$

axm8:

$(\forall g1, g2 \cdot g1 \mapsto g2 \in dom(Op) \Rightarrow Op(g1 \mapsto g2) \in G)$

$\Leftrightarrow$

$HasClosure(Op) = TRUE$

axm9:

$IsAssoc(Op) = TRUE$

$\Leftrightarrow$

$(HasClosure(Op) = TRUE$

$\land$

$\forall g1, g2, g3 \cdot g1 \in G \land g2 \in G \land g3 \in G$

$\Rightarrow Op(g1 \mapsto Op(g2 \mapsto g3)) = Op(Op(g1 \mapsto g2) \mapsto g3))$

axm10:

$HasId(Op) = TRUE$

$\Leftrightarrow$

$(\forall g \cdot g \in G \Rightarrow (Op(e \mapsto g) = g \land Op(g \mapsto e) = g))$

axm11:

$HasInverse(Op) = TRUE$

$\Leftrightarrow$

$(\forall g \cdot g \in G \Rightarrow$

$(\exists minusg \cdot minusg \in G \land$

$Op(g \mapsto minusg) = e \land$

$Op(minusg \mapsto g) = e))$

axm12:

$IsGroup(G \mapsto Op) = TRUE$

$\Leftrightarrow$

$(dom(Op) = G \times G \land$

$HasClosure(Op) = TRUE \land$

$IsAssoc(Op) = TRUE \land$

$HasId(Op) = TRUE \land$

$HasInverse(Op) = TRUE)$

END

C.4.2 Group Theory 2

Referenced in Section 8.4.2.

CONTEXT GroupTheory02

DESCRIPTION

Extends context `GroupTheory01`.
G is defined as the set $\{1, 2\}$, $e = 1$ and Op is defined on G as multiplication mod 3.
$\langle G, Op \rangle$ satisfies the *group axioms*.

`axm1` : G and e are defined
`axm2` : Op is defined
`thm3` : Lemma to assist with discharging the PO for `thm4`
`thm4` : $\langle G, Op \rangle$ has the closure property
`thm5` : Lemma to assist with discharging the PO for `thm6`
`thm6` : $\langle G, Op \rangle$ has an identity element
`thm7` : Op is associative
`thm8` : Every element in G has an inverse in G
`thm9` : $\langle G, Op \rangle$ is a group

EXTENDS GroupTheory01
AXIOMS

`axm1`: $G = \{1, 2\} \land e = 1$
`axm2`: $Op = \{1 \mapsto 1 \mapsto 1, 1 \mapsto 2 \mapsto 2, 2 \mapsto 1 \mapsto 2, 2 \mapsto 2 \mapsto 1\}$
multiplication mod 3
`thm3`: $\langle$theorem$\rangle$ $\forall x, y \cdot x \in G \land y \in G \Rightarrow Op(x \mapsto y) \in G$
lemma
`thm4`: $\langle$theorem$\rangle$ $HasClosure(Op) = TRUE$
`thm5`: $\langle$theorem$\rangle$ $\forall g \cdot g \in G \Rightarrow (Op(e \mapsto g) = g \land Op(g \mapsto e) = g)$
lemma
`thm6`: $\langle$theorem$\rangle$ $HasId(Op) = TRUE$
`thm7`: $\langle$theorem$\rangle$ $IsAssoc(Op) = TRUE$
`thm8`: $\langle$theorem$\rangle$ $HasInverse(Op) = TRUE$
`thm9`: $\langle$theorem$\rangle$ $IsGroup(G \mapsto Op) = TRUE$

END

.

C.4.3 Group Theory 3

CONTEXT GroupTheory03

DESCRIPTION

The operator Op, defined by the rule $Op(x, y) = x + y$, is evaluated for the closure property.
Op has the closure property when applied to the set of even integers.
Op does not have the closure property when applied to the set of odd integers.

axm1 : Function *IsClosed* is declared
axm2 : Set *Evens* is defined
axm3 : Operator *EvensOp* is defined
axm4 : Function *IsClosed* is defined
thm5 : *EvensOp* is closed, i.e. the sum of two even integers is even
axm6 : Set *Odds* is defined
axm7 : Operator *OddsOp* is defined
thm8 : Lemma to assist with discharging the PO for thm9
thm9 : *OddsOp* is not closed, i.e. the sum of two odd integers is not odd, but even

CONSTANTS

IsClosed
Evens
Odds
EvensOp
OddsOp

AXIOMS

axm1: $IsClosed \in (\mathbb{Z} \times \mathbb{Z} \pfun \mathbb{Z}) \to BOOL$
axm2: $Evens = \{n \cdot n \in \mathbb{Z} \mid 2 * n\}$
axm3: $EvensOp = \{x, y \cdot x \mapsto y \in Evens \times Evens \mid x \mapsto y \mapsto (x + y)\}$
axm4:
$\forall Op \cdot (Op \in dom(IsClosed)$
$\Rightarrow$
$(\forall S, x, y \cdot (S \subseteq \mathbb{Z} \land x \in S \land y \in S \land x \mapsto y \in dom(Op) \Rightarrow Op(x \mapsto y) \in S)$
$\Leftrightarrow$
$IsClosed(Op) = TRUE))$
thm5: ⟨theorem⟩ $IsClosed(EvensOp) = TRUE$
axm6: $Odds = \{n \cdot n \in \mathbb{Z} \mid 2 * n + 1\}$
axm7: $OddsOp = \{x, y \cdot x \mapsto y \in Odds \times Odds \mid x \mapsto y \mapsto (x + y)\}$
thm8: ⟨theorem⟩ $OddsOp \in dom(IsClosed)$
Lemma
thm9: ⟨theorem⟩ $IsClosed(OddsOp) = FALSE$

END

C.4.4 Group Theory 4

Referenced in Sections 7.5.3 and 9.2.

CONTEXT GroupTheory04

DESCRIPTION

Let G be the set of even integers.
Let Op be a function on G given by $Op(x, y) = x + y$.
Then $\langle G, Op \rangle$ is a group.

axm1 : G is the set of even integers
axm2 : Function Op is declared
axm3 : $Op(x, y) = x + y$
axm4 : $g1, g2, g3 \in G$

thm5 : $\langle G, Op \rangle$ has the closure property
thm6 : Op is associative
thm7 : Lemma to assist with discharging thm8
thm8 : $\langle G, Op \rangle$ has an identity element
thm9 : Lemma to assist with discharging thm11
thm10 : Lemma to assist with discharging thm11
thm11 : Every element of G has an inverse in G

CONSTANTS

G
Op
g1
g2
g3

AXIOMS

axm1: $G = \{x \cdot x \in \mathbb{Z} \mid 2 * x\}$
axm2: $Op \in \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$
axm3: $\forall x, y \cdot x \in \mathbb{Z} \wedge y \in \mathbb{Z} \Rightarrow Op(x \mapsto y) = x + y$
axm4: $g1 \in G \wedge g2 \in G \wedge g3 \in G$
thm5: $\langle$theorem$\rangle$ $Op(g1 \mapsto g2) \in G$
thm6: $\langle$theorem$\rangle$ $Op(g1 \mapsto Op(g2 \mapsto g3)) = Op(Op(g1 \mapsto g2) \mapsto g3)$
thm7: $\langle$theorem$\rangle$ $0 \in G$
lemma 1
thm8: $\langle$theorem$\rangle$ $\exists e \cdot e \in G \wedge (\forall g \cdot g \in G \Rightarrow Op(e \mapsto g) = g \wedge Op(g \mapsto e) = g)$
thm9: $\langle$theorem$\rangle$ $\forall x \cdot x + (-x) = 0 \wedge (-x) + x = 0$
lemma 2
thm10: $\langle$theorem$\rangle$ $\forall x \cdot 2 * x + (-2 * x) = 0 \wedge (-2 * x) + 2 * x = 0$
lemma 3
thm11: $\langle$theorem$\rangle$
$\forall g \cdot g \in G \Rightarrow$
$(\exists minusg \cdot Op(g \mapsto minusg) = 0$
$\wedge$
$Op(minusg \mapsto g) = 0)$

END

C.5 Integer Properties

In this section we explore a number of simple properties of the set of integers, i.e. the set $\mathbb{Z}$. The generated proof obligations for the theorems in this section could all be discharged without invoking the SMT provers (see Table D.1). In other words, all the proof obligations in this section could be discharged by the *default auto tactic profile* (see Section 8.1).

C.5.1 Integer Properties 1

CONTEXT IntegerProperties01

DESCRIPTION

Integer property 1: $x * x = 1 \Rightarrow x = 1 \vee x = -1$.

CONSTANTS

x

y

z

AXIOMS

axm1: $x \in \mathbb{Z} \wedge y \in \mathbb{Z} \wedge z \in \mathbb{Z}$

axm2: $x * x = 1$

axm3: $y = 1$

axm4: $z = -1$

thm5: ⟨theorem⟩ $x = y \vee x = z$

END

C.5.2 Integer Properties 2

CONTEXT IntegerProperties02

DESCRIPTION

Integer property 2: $1 \leq x \leq 2 \Rightarrow x = 1 \vee x = 2$.

CONSTANTS

x

y

z

AXIOMS

axm1: $x \in \mathbb{Z} \wedge y \in \mathbb{Z} \wedge z \in \mathbb{Z}$

axm2: $y = 1$

axm3: $z = 2$

axm4: $1 \leq x \wedge x \leq 2$

thm5: ⟨theorem⟩ $x = y \vee x = z$

END

C.5.3 Integer Properties 3

CONTEXT IntegerProperties03

DESCRIPTION

Integer property 3: $x = 2 - 3 \Rightarrow x \leq -1$.

CONSTANTS

x

y

z

AXIOMS

axm1: $x \in \mathbb{Z} \land y \in \mathbb{Z} \land z \in \mathbb{Z}$

axm2: $x = y - z$

axm3: $y \leq 2$

axm4: $z \geq 3$

thm5: ⟨theorem⟩ $x \leq -1$

END

C.5.4 Integer Properties 4

CONTEXT IntegerProperties04

DESCRIPTION

Integer property 4: $(X + Y) + Z = X + (Y + Z)$.

CONSTANTS

X

Y

Z

Z1

Z2

Z3

Z4

AXIOMS

axm1: $\{X, Y, Z\} \subseteq \mathbb{Z}$

axm2: $\{Z1, Z2, Z3, Z4\} \subseteq \mathbb{Z}$

axm3: $X + Y = Z1$

axm4: $Z1 + Z = Z2$

axm5: $Y + Z = Z3$

axm6: $X + Z3 = Z4$

thm7: ⟨theorem⟩ $Z2 = Z4$

END

C.5.5 Integer Properties 5

CONTEXT IntegerProperties05

DESCRIPTION

Integer property 5: $X + Y < X - Y$ does not hold for all integers X and Y.
Counterexample: Let $Y = 0$.

AXIOMS

thm1: ⟨theorem⟩

$\neg\ (\forall X, Y, Z1, Z2 \cdot \{X, Y, Z1, Z2\} \subseteq \mathbb{Z} \land$
$(((X + Y) = Z1) \land ((X - Y) = Z2) \Rightarrow (Z1 < Z2)))$

END

C.5.6 Integer Properties 6

CONTEXT IntegerProperties06

DESCRIPTION

Integer property 6: $(X + Y) = Z \Rightarrow (X < Z) \wedge (Y < Z)$ does not hold for all integers X, Y and Z.
Counterexample: Let $Y = 0$.

AXIOMS

thm1: ⟨theorem⟩
$\neg (\forall X, Y, Z \cdot \{X, Y, Z\} \subseteq \mathbb{Z} \wedge (X + Y) = Z$
$\Rightarrow (X < Z) \wedge (Y < Z))$

END

C.5.7 Integer Properties 7

CONTEXT IntegerProperties07

DESCRIPTION

Integer property 7: $(Y - X) < (X - Y) \Rightarrow -2X < -2Y \Rightarrow Y < X$.

AXIOMS

thm1: ⟨theorem⟩
$\forall X, Y \cdot X \in \mathbb{Z} \wedge Y \in \mathbb{Z} \wedge (Y - X) < (X - Y)$
$\Rightarrow$
$Y < X$

END

C.5.8 Integer Properties 8

CONTEXT IntegerProperties08

DESCRIPTION

Integer property 8: There is no natural number less than 0.

AXIOMS

thm1: ⟨theorem⟩ $\neg (\exists X \cdot X \in \mathbb{N} \wedge X < 0)$

END

C.6 Lasso Operator

Two contexts are defined to illustrate the need for Rodin's *lasso* operator. The base context, `LassoBaseContext`, declares and defines a function f. The symbol f is therefore available in the derived context `LassoDerivedContext`.

However, proof obligations that arise in the derived context do not automatically include the axioms and theorems from the base context. For example, none of the Rodin theorem provers can discharge the proof obligation for `thm6` in context `LassoDerivedContext`. When *lasso* is applied, `axm2` in context `LassoBaseContext` is added to the list of hypotheses. Now the universal quantifier can be instantiated and the proof obligation is discharged.

C.6.1 Lasso Base Context

Referenced in Sections 8.3.5 and A.1.

CONTEXT LassoBaseContext

DESCRIPTION

The two contexts `LassoBaseContext` and `LassoDerivedContext` illustrate the need for the *lasso* operator.
In context `LassoBaseContext`, the function f is declared and defined.

`axm1` : Declaration of function f
`axm2` : Definition of f

CONSTANTS

f

AXIOMS

`axm1`: $f \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \rightarrow \{0,1\}$

`axm2`:

$\forall X, Y \cdot \{X, Y\} \subseteq \mathbb{P}(\mathbb{Z}) \Rightarrow$
$(X \cap Y = \varnothing \Rightarrow f(X \mapsto Y) = 0)$
$\wedge$
$(X \cap Y \neq \varnothing \Rightarrow f(X \mapsto Y) = 1)$

END

C.6.2 Lasso Derived Context

Referenced in Section 8.3.5.

CONTEXT LassoDerivedContext

DESCRIPTION

Context `LassoBaseContext` is extended.
The two contexts `LassoBaseContext` and `LassoDerivedContext` illustrate the need for the *lasso* operator.

In context `LassoDerivedContext`, the function f is applied to a number of ordered pairs.

`thm1` - `thm4` : f applied to various ordered pairs of integer sets
`thm5` : Lemma to assist with discharging the PO for `thm6`
`thm6` : f applied to an ordered pair containing a polynomial. The *lasso* operator must be applied manually in order to discharge the PO

EXTENDS LassoBaseContext

AXIOMS

`thm1`: ⟨theorem⟩ $f(\{1\} \mapsto \{1\}) \neq 0$

`thm2`: ⟨theorem⟩ $f(\{1\} \mapsto \{2\}) \neq 1$

`thm3`: ⟨theorem⟩ $f(\{0, 1, 5, 12\} \mapsto \{2, 22, 222\}) = 0$

`thm4`: ⟨theorem⟩ $f(\{2, 4, 6, 8, 10\} \mapsto \{0, 1, 10, 101, 1010\}) = 1$

`thm5`: ⟨theorem⟩ $\{2, 4, 6, 8, 10\} \cap \{x \cdot x \in \mathbb{Z} \mid x * x\} \neq \varnothing$

Lemma

`thm6`: ⟨theorem⟩ $f(\{2, 4, 6, 8, 10\} \mapsto \{x \cdot x \in \mathbb{Z} \mid x * x\}) = 1$

END

C.7 Predicate Logic

Predicate Logic examples from (Pelletier 1986, p. 196) are presented in this section. Since the type of every symbol in a Rodin context must either be included in an axiom or be derivable through use of the symbol, predicates (boolean-valued functors) are functions mapping from some set U to the predefined type $BOOL = \{TRUE, FALSE\}$.

C.7.1 Predicate Logic 1

CONTEXT PredicateLogic01

DESCRIPTION

Two examples from the *Monadic predicate logic* section in (Pelletier 1986, p. 196).

`axm1` : Declaration of F
`thm2` : Monadic predicate logic 18 in (Pelletier 1986, p. 196)
`axm3` : Declaration of P and Q
`thm4` : Monadic predicate logic 19 in (Pelletier 1986, p. 196)

SETS

U

CONSTANTS

F ∈ U → BOOL

P ∈ U → BOOL

Q ∈ U → BOOL

AXIOMS

`axm1`: $F \in U \rightarrow BOOL$

thm2: ⟨theorem⟩
$\exists y \cdot y \in U \land (\forall x \cdot x \in U \land F(y) = TRUE$
$\Rightarrow F(x) = TRUE)$
axm3: $P \in U \rightarrow BOOL \land Q \in U \rightarrow BOOL$
thm4: ⟨theorem⟩
$\exists x \cdot x \in U \land (\forall y, z \cdot y \in U \land z \in U \land (P(y) = TRUE \Rightarrow Q(z) = TRUE)$
$\Rightarrow (P(x) = TRUE \Rightarrow Q(x) = TRUE))$

END

C.7.2 Predicate Logic 2

CONTEXT PredicateLogic02

DESCRIPTION

Another example from the *Monadic predicate logic* section in (Pelletier 1986, p. 198).

axm1 : Declarations of F, G, H and J
axm2 - axm3 : Properties of functions F and G
thm4 : Monadic predicate logic 29 in (Pelletier 1986, p. 198)

SETS
U
CONSTANTS
F ∈ U → BOOL
G ∈ U → BOOL
H ∈ U → BOOL
J ∈ U → BOOL
AXIOMS
axm1:
$F \in U \rightarrow BOOL$
$\land G \in U \rightarrow BOOL$
$\land H \in U \rightarrow BOOL$
$\land J \in U \rightarrow BOOL$
axm2: $\exists x \cdot x \in U \land F(x) = TRUE$
axm3: $\exists x \cdot x \in U \land G(x) = TRUE$
thm4: ⟨theorem⟩
$((\forall x \cdot x \in U \land F(x) = TRUE \Rightarrow H(x) = TRUE)$
$\land (\forall x \cdot x \in U \land G(x) = TRUE \Rightarrow J(x) = TRUE))$
$\Leftrightarrow$
$(\forall x, y \cdot x \in U \land F(x) = TRUE \land G(y) = TRUE \Rightarrow H(x) = TRUE \land J(y) = TRUE)$

END

C.8 Propositional Logic

Propositional Logic examples from Pelletier (1986, p. 196) are presented in this section. Since the type of every symbol in a Rodin context must either be stated as an axiom or be derivable from use of the symbol, propositions (boolean-valued variables) are constants of the predefined type $BOOL = \{TRUE, FALSE\}$.

C.8.1 Propositional Logic 1

CONTEXT PropositionalLogic01

DESCRIPTION

thm1 to thm8 appear in Pelletier (1986, p. 193-194).
These problems were considered *hard problems for automated theorem provers* at the time.

thm1 : Propositional logic 1 in Pelletier (1986, p. 193)
thm2 : Propositional logic 2 in Pelletier (1986, p. 193)
thm3 : Propositional logic 3 in Pelletier (1986, p. 193)
thm4 : Propositional logic 4 in Pelletier (1986, p. 193)
thm5 : Propositional logic 5 in Pelletier (1986, p. 194)
thm6 : Propositional logic 6 in Pelletier (1986, p. 194)
thm7 : Propositional logic 7 in Pelletier (1986, p. 194)
thm8 : Propositional logic 8 in Pelletier (1986, p. 194)
thm9 : Evaluating truth assignment

CONSTANTS

p
q
r

AXIOMS

thm1: ⟨theorem⟩
$(p = TRUE \Rightarrow q = TRUE)$
$\Leftrightarrow$
$(\neg\ (q = TRUE) \Rightarrow \neg\ (p = TRUE))$

thm2: ⟨theorem⟩ $\neg\ \neg\ (p = TRUE) \Leftrightarrow (p = TRUE)$

thm3: ⟨theorem⟩
$\neg\ (p = TRUE \Rightarrow q = TRUE)$
$\Rightarrow$
$(q = TRUE \Rightarrow p = TRUE)$

thm4: ⟨theorem⟩
$(\neg\ (p = TRUE) \Rightarrow q = TRUE)$
$\Leftrightarrow$
$(\neg\ (q = TRUE) \Rightarrow p = TRUE)$

thm5: ⟨theorem⟩
$(((p = TRUE) \vee (q = TRUE)) \Rightarrow ((p = TRUE) \vee (r = TRUE)))$
$\Rightarrow$
$((p = TRUE) \vee ((q = TRUE) \Rightarrow (r = TRUE)))$

thm6: ⟨theorem⟩ $(p = TRUE) \vee \neg\ (p = TRUE)$

thm7: ⟨theorem⟩ $(p = TRUE) \vee \neg\ \neg\ \neg\ (p = TRUE)$

thm8: ⟨theorem⟩
$(((p = TRUE) \Rightarrow q = TRUE) \Rightarrow (p = TRUE))$
$\Rightarrow (p = TRUE)$

thm9: ⟨theorem⟩ $p = TRUE \Leftrightarrow \neg\ p = FALSE$

END

C.8.2 Propositional Logic 2

CONTEXT PropositionalLogic02

DESCRIPTION

This *negated-conclusion clause form* example appears in Pelletier (1986, p. 193).
The *natural form* equivalent was given in `PropositionalLogic01`, `thm1`.

`axm1` - `thm4` : *Negated-conclusion clause form* of Propositional logic 1 in Pelletier (1986, p. 193)

CONSTANTS

p

q

AXIOMS

`axm1`: $(p = TRUE) \lor (q = TRUE)$

`axm2`: $\neg\,(p = TRUE) \lor (q = TRUE)$

`axm3`: $(p = TRUE) \lor \neg\,(q = TRUE)$

`thm4`: ⟨theorem⟩ $\neg\,(\neg\,(p = TRUE) \lor \neg\,(q = TRUE))$

END

C.9 Relations and Functions

The contexts in this section illustrate various aspects of discharging proof obligations involving relations and functions in Rodin/Event-B. The Event-B language provides extensive support for relations and functions, with symbols for properties such as everywhere defined (versus partial), surjectivity, injectivity and bijectivity.

C.9.1 Relations and Functions 1

Referenced in Sections 8.2.1.1 and 8.3.3.

CONTEXT RelationsAndFunctions01

DESCRIPTION

The function *HasControlValues* maps a family of sets of integers to *TRUE* iff some member set contains all the values in *ControlValues*.
Otherwise the family of sets is mapped to *FALSE*.

`axm1` : *ControlValues* is declared
`axm2` : *HasControlValues* is declared
`axm3` : *HasControlValues* is defined

CONSTANTS

ControlValues

HasControlValues

AXIOMS

axm1: $ControlValues \subseteq \mathbb{Z}$

axm2: $HasControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$

axm3:

$\forall X \cdot X \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \Rightarrow$
$(HasControlValues(X) = TRUE$
$\Leftrightarrow$
$(\exists x \cdot x \in X \land ControlValues \subseteq x))$

END

C.9.2 Relations and Functions 2

Referenced in Section 8.3.3.

CONTEXT RelationsAndFunctions02

DESCRIPTION

Context `RelationsAndFunctions01` is extended.
ControlValues is defined and a number of families of sets are evaluated for the *HasControlValue* property.

`axm1` : *ControlValues* is defined as the set of multiples of 6
`thm2` : The set containing the set of multiples of 3 is evaluated
`thm3` : The set containing the sets of integers $3 * x + 2$ and $2 * x + 1$ respectively is evaluated
`thm4` : The set containing the empty set is evaluated
`thm5` : The set containing the power set of $\mathbb{Z}$ is evaluated

EXTENDS RelationsAndFunctions01
AXIOMS

axm1: $ControlValues = \{x \cdot x \in \mathbb{Z} \mid 6 * x\}$

thm2: ⟨theorem⟩ $HasControlValues(\{\{x \cdot x \in \mathbb{Z} \mid 3 * x\}\}) = TRUE$

thm3: ⟨theorem⟩
$HasControlValues(\{\{x \cdot x \in \mathbb{Z} \mid 3 * x + 2\},$
$\{x \cdot x \in \mathbb{Z} \mid 2 * x + 1\}\}) = FALSE$

thm4: ⟨theorem⟩ $HasControlValues(\varnothing) = FALSE$

thm5: ⟨theorem⟩ $HasControlValues(\mathbb{P}(\mathbb{Z})) = TRUE$

END

C.9.3 Relations and Functions 3

Referenced in Section 8.2.1.1.

CONTEXT RelationsAndFunctions03

DESCRIPTION

Context `RelationsAndFunctions01` is extended.
ControlValues is defined and a number of families of sets are evaluated for the *HasControlValue* property.

`axm1` : *ControlValues* is defined as the set of integers $0, 1, \ldots, 5$
`thm2` : The set containing the set of integers > 0, the set of integers $3 * x + 1$ and the set of roots of $y = 3 * x + 1$

is evaluated
axm3 : The family of sets S is declared and defined
thm4 : S is evaluated

EXTENDS RelationsAndFunctions01

CONSTANTS

tmp1
tmp2
tmp3
tmp4
S

AXIOMS

axm1: $ControlValues = \{0, 1, 2, 3, 4, 5\}$

thm2: ⟨theorem⟩
$HasControlValues(\{\{x \cdot x \in \mathbb{N} \mid x + 1\},$
$\{x \cdot x \in \mathbb{Z} \mid 3 * x + 1\},$
$\{x \cdot 3 * x + 1 = 0 \mid x\}\}) = FALSE$

axm3:
$tmp1 = \{-1, 0, 1\}$
$\wedge$
$tmp2 = \{1, 2\}$
$\wedge$
$tmp3 = \{x \cdot x \in \mathbb{Z} \mid 3 * x + 1\}$
$\wedge$
$tmp4 = \{x \cdot x \in \mathbb{Z} \wedge x \geq 0 \wedge x < 6 \mid x\}$
$\wedge$
$S = \{tmp1, tmp2, tmp3, tmp4\}$

thm4: ⟨theorem⟩ $HasControlValues(S) = TRUE$

END

C.9.4 Relations and Functions 4

CONTEXT RelationsAndFunctions04

DESCRIPTION

The function *HasFirstThreePrimes* maps a triple of integer sets to *TRUE* iff the union of the triple's elements contains $\{2, 3, 5\}$.
Otherwise the triple is mapped to *FALSE*.

axm1 : *HasFirstThreePrimes* is declared
axm2 : *HasFirstThreePrimes* is defined
thm3 - thm8 : Various triples are evaluated

CONSTANTS

HasFirstThreePrimes

AXIOMS

axm1: $HasFirstThreePrimes \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \rightarrow BOOL$

axm2:
$\forall X, Y, Z \cdot X \subseteq \mathbb{Z} \land Y \subseteq \mathbb{Z} \land Z \subseteq \mathbb{Z} \Rightarrow$
$(\mathit{HasFirstThreePrimes}(X \mapsto Y \mapsto Z) = \mathit{TRUE}$
$\Leftrightarrow$
$\{2,3,5\} \subseteq X \cup Y \cup Z)$

thm3: ⟨theorem⟩ $\mathit{HasFirstThreePrimes}(\{1,2,3\} \mapsto \{2,5,8\} \mapsto \{6\}) = \mathit{TRUE}$

thm4: ⟨theorem⟩ $\mathit{HasFirstThreePrimes}(\{3\} \mapsto \{6\} \mapsto \{2,5,8\}) = \mathit{TRUE}$

thm5: ⟨theorem⟩ $\mathit{HasFirstThreePrimes}(\{1,2,3,4,5\} \mapsto \varnothing \mapsto \varnothing) = \mathit{TRUE}$

thm6: ⟨theorem⟩ $\mathit{HasFirstThreePrimes}(\{6\} \mapsto \{1,4,8\} \mapsto \{3\}) = \mathit{FALSE}$

thm7: ⟨theorem⟩ $\mathit{HasFirstThreePrimes}(\{2,5,8\} \mapsto \{1,4,8\} \mapsto \varnothing) = \mathit{FALSE}$

thm8: ⟨theorem⟩ $\mathit{HasFirstThreePrimes}(\varnothing \mapsto \{1,4,8\} \mapsto \varnothing) = \mathit{FALSE}$

END

C.9.5 Relations and Functions 5

CONTEXT RelationsAndFunctions05

DESCRIPTION

The function *HasFirstThreePrimes* maps a 6-tuple of integer sets to *TRUE* iff the union of the 6-tuple's elements contains $\{2,3,5\}$.
Otherwise the 6-tuple is mapped to *FALSE*.
A number of integer sets are declared and evaluated along with $\varnothing$.

axm1 : *HasFirstThreePrimes* is declared
axm2 : *HasFirstThreePrimes* is defined
axm3 - axm8 : Various triples are evaluated
thm9 - thm20 : Various combinations of the sets *A* to *F* and $\varnothing$ are evaluated

CONSTANTS

HasFirstThreePrimes
A
B
C
D
E
F

AXIOMS

axm1: $\mathit{HasFirstThreePrimes} \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \to \mathit{BOOL}$

axm2:
$\forall S, T, U, X, Y, Z \cdot S \subseteq \mathbb{Z} \land T \subseteq \mathbb{Z} \land U \subseteq \mathbb{Z} \land X \subseteq \mathbb{Z} \land Y \subseteq \mathbb{Z} \land Z \subseteq \mathbb{Z} \Rightarrow$
$(\mathit{HasFirstThreePrimes}(S \mapsto T \mapsto U \mapsto X \mapsto Y \mapsto Z) = \mathit{TRUE}$
$\Leftrightarrow$
$\{2,3,5\} \subseteq S \cup T \cup U \cup X \cup Y \cup Z)$

axm3: $A = \{1,2,3\}$

axm4: $B = \{2,5,8\}$

axm5: $C = \{6\}$

axm6: $D = \{1,4,8\}$

axm7: $E = \{3\}$

axm8: $F = \{1, 2, 3, 4, 5\}$

thm9: ⟨theorem⟩ $HasFirstThreePrimes(A \mapsto B \mapsto C \mapsto D \mapsto E \mapsto F) = TRUE$

thm10: ⟨theorem⟩ $HasFirstThreePrimes(E \mapsto C \mapsto B \mapsto \varnothing \mapsto D \mapsto \varnothing) = TRUE$

thm11: ⟨theorem⟩ $HasFirstThreePrimes(F \mapsto \varnothing \mapsto \varnothing \mapsto D \mapsto B \mapsto E) = TRUE$

thm12: ⟨theorem⟩ $HasFirstThreePrimes(\varnothing \mapsto B \mapsto C \mapsto D \mapsto \varnothing \mapsto F) = TRUE$

thm13: ⟨theorem⟩ $HasFirstThreePrimes(A \mapsto \varnothing \mapsto C \mapsto \varnothing \mapsto E \mapsto F) = TRUE$

thm14: ⟨theorem⟩ $HasFirstThreePrimes(\varnothing \mapsto B \mapsto \varnothing \mapsto D \mapsto E \mapsto \varnothing) = TRUE$

thm15: ⟨theorem⟩ $HasFirstThreePrimes(A \mapsto B \mapsto \varnothing \mapsto D \mapsto \varnothing \mapsto F) = TRUE$

thm16: ⟨theorem⟩ $HasFirstThreePrimes(A \mapsto B \mapsto C \mapsto \varnothing \mapsto E \mapsto F) = TRUE$

thm17: ⟨theorem⟩ $HasFirstThreePrimes(C \mapsto D \mapsto C \mapsto B \mapsto D \mapsto C) = FALSE$

thm18: ⟨theorem⟩ $HasFirstThreePrimes(\varnothing \mapsto \varnothing \mapsto \varnothing \mapsto \varnothing \mapsto \varnothing \mapsto \varnothing) = FALSE$

thm19: ⟨theorem⟩ $HasFirstThreePrimes(A \mapsto \varnothing \mapsto \varnothing \mapsto \varnothing \mapsto \varnothing \mapsto \varnothing) = FALSE$

thm20: ⟨theorem⟩ $HasFirstThreePrimes(A \mapsto \varnothing \mapsto C \mapsto D \mapsto E \mapsto \varnothing) = FALSE$

END

C.9.6 Relations and Functions 6

CONTEXT RelationsAndFunctions06

DESCRIPTION

The function *HasControlValue* maps a family of integer sets to *TRUE* iff each element set contains at least one value from *ControlValues*.
Otherwise the family of sets is mapped to *FALSE*.

axm1 : *ControlValues* is defined
axm2 : *HasControlValue* is declared
axm3 : *HasControlValue* is defined
thm4 : A family of integer sets is evaluated
thm5 : A family of integer sets is evaluated

CONSTANTS

ControlValues

HasControlValue

AXIOMS

axm1: $ControlValues = \{0, 1, 2, 3, 4, 5\}$

axm2: $HasControlValue \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \to BOOL$

axm3:
$\forall X \cdot X \subseteq \mathbb{P}(\mathbb{Z}) \Rightarrow$
$(HasControlValue(X) = TRUE$
$\Leftrightarrow$
$(\forall x \cdot x \in X \Rightarrow (\exists e \cdot e \in x \wedge e \in ControlValues)))$

thm4: ⟨theorem⟩
$HasControlValue(\{\{1\}, \{x \cdot x \in \mathbb{Z} \mid x + 1\},$
$\{x \cdot x \in \mathbb{Z} \wedge x > 0 \wedge x < 4 \mid x\}, \{-1, 0, 1\}\}) = TRUE$

thm5: ⟨theorem⟩ $HasControlValue(\{\{1\}, \{2\}, \{5\}, \{0\}\}) = TRUE$

END

C.9.7 Relations and Functions 7

CONTEXT RelationsAndFunctions07

DESCRIPTION

The function *HasControlValue* maps a family of integer sets to *TRUE* iff each element set must contain at least one value from *ControlValues*.
Otherwise the family of sets is mapped to *FALSE*.

axm1 : *ControlValues* is defined
axm2 : *HasControlValue* is declared
axm3 : *HasControlValue* is defined
thm4 : A family of integer sets is evaluated
axm5 : Set *tmp* is defined (needed to discharge the PO automatically)
thm6 : Set *tmp* is evaluated

CONSTANTS

ControlValues
HasControlValue
tmp

AXIOMS

axm1: $ControlValues = \{0, 1, 2, 3, 4, 5\}$
axm2: $HasControlValue \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$
axm3:
$\forall X \cdot X \subseteq \mathbb{P}(\mathbb{Z}) \Rightarrow$
$(HasControlValue(X) = TRUE$
$\Leftrightarrow$
$(\forall x \cdot x \in X \Rightarrow (\exists e \cdot e \in x \wedge e \in ControlValues)))$
thm4: ⟨theorem⟩ $HasControlValue(\{\{19\}\}) = FALSE$
axm5: $tmp = \{19, 21, 36\}$
thm6: ⟨theorem⟩ $HasControlValue(\{\{19, 21, 36\}, \{x \cdot x \in \mathbb{N} \mid x + 7\}\}) = FALSE$

END

C.9.8 Relations and Functions 8

CONTEXT RelationsAndFunctions08

DESCRIPTION

The function *NotAllControlValues* maps a family of integer sets to *TRUE* iff every element set does not contain at least one value from *ControlValues*.
Otherwise the family of sets is mapped to *FALSE*.

axm1 : *ControlValues* is defined
axm2 : *NotAllControlValues* is declared
axm3 : *NotAllControlValues* is defined
thm4 : A family of integer sets is evaluated
thm5 : A family of integer sets is evaluated
axm6 : Sets *tmp*1 and *tmp*2 are defined

thm7 : The set containing *tmp*1 and *tmp*2 is evaluated

CONSTANTS

ControlValues
NotAllControlValues
tmp1
tmp2

AXIOMS

axm1: $ControlValues = \{0, 1, 2, 3, 4, 5\}$
axm2: $NotAllControlValues \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$
axm3:
$\forall X \cdot X \subseteq \mathbb{P}(\mathbb{Z}) \Rightarrow$
$(NotAllControlValues(X) = TRUE$
$\Leftrightarrow$
$(\forall x \cdot x \in X \Rightarrow (\exists e \cdot e \in ControlValues \land e \notin x)))$
thm4: ⟨theorem⟩ $NotAllControlValues(\{\{19\}\}) = TRUE$
thm5: ⟨theorem⟩ $NotAllControlValues(\{\{19, 21, 36\}, \{x \cdot x \in \mathbb{N} \mid x + 2\}\}) = TRUE$
axm6:
$tmp1 = \{0, 1, 2, 3, 4, 5, 6\}$
$\land$
$tmp2 = \{x \cdot x \in \mathbb{N} \mid x + 7\}$
thm7: ⟨theorem⟩ $NotAllControlValues(\{\{0, 1, 2, 3, 4, 5, 6\}, \{x \cdot x \in \mathbb{N} \mid x + 7\}\}) = FALSE$

END

C.9.9 Relations and Functions 9

Referenced in Section 8.2.1.2.

CONTEXT RelationsAndFunctions09

DESCRIPTION

The function *HasElementProperty* maps a family of integer sets to *TRUE* iff every pair of element sets has a non-empty intersection.
Otherwise the family of sets is mapped to *FALSE*.
NOTE: If $\varnothing$ is a member set, the family of sets is mapped to *FALSE*.

axm1 : *HasElementProperty* is declared
axm2 : *HasElementProperty* is defined
thm3 : A set containing $\varnothing$ cannot have the *HasElementProperty* property
thm4 : A set containing a single, non-empty set has the *HasElementProperty* property

CONSTANTS

HasElementProperty

AXIOMS

axm1: $HasElementProperty \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$
axm2:
$\forall S \cdot S \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \Rightarrow$
$(HasElementProperty(S) = TRUE$
$\Leftrightarrow$
$(\forall X, Y \cdot X \in S \land Y \in S \Rightarrow X \cap Y \neq \varnothing))$

thm3: ⟨theorem⟩
$\forall S \cdot S \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \land \varnothing \in S \Rightarrow$
$HasElementProperty(S) = FALSE$

thm4: ⟨theorem⟩
$\forall X \cdot X \subseteq \mathbb{Z} \land X \neq \varnothing \Rightarrow$
$HasElementProperty(\{X\}) = TRUE$

END

C.9.10 Relations and Functions 10

CONTEXT RelationsAndFunctions10

DESCRIPTION

Context `RelationsAndFunctions09` is extended.
Sets *A* and *B* are defined and evaluated.

`axm1` : Set *A* is defined
`thm2` : Set *A* is evaluated
`axm3` : Set *B* is defined
`thm4` : Set *B* is evaluated

EXTENDS RelationsAndFunctions09
CONSTANTS
A
B
AXIOMS
axm1: $A = \{\{1,2\},\{1,4\},\{1,3,5,8\},\{-2,-1,0,1,2\}\}$
thm2: ⟨theorem⟩ $HasElementProperty(A) = TRUE$
axm3:
$B = \{\{4,66,8,1,5,2\},\{2,5,44,79,98,1\},$
$\{123,65,74,5,225,21\},$
$\{225,369,234,2\},\{2,23,21,25\},\{2,5,4\}\}$
thm4: ⟨theorem⟩ $HasElementProperty(B) = TRUE$
END

C.9.11 Relations and Functions 11

CONTEXT RelationsAndFunctions11

DESCRIPTION

Context `RelationsAndFunctions09` is extended.
Two sets are evaluated for the *HasElementProperty* property.

`thm1` : Example of a set with only one, non-empty member which has the *HasElementProperty* property (see `thm4` in `RelationsAndFunctions09`)
`thm2` : Family of sets containing a polynomial evaluated

EXTENDS RelationsAndFunctions09

AXIOMS

thm1: ⟨theorem⟩ *HasElementProperty*($\{\{x \cdot x > 5 \mid 2 * x - 1\}\}$) = *TRUE*

thm2: ⟨theorem⟩
HasElementProperty($\{\{x \cdot x > 5 \mid 2 * x - 1\}$,
$\{3, 5, 7, 11\}$,
$\{5, 15\}\}$) = *TRUE*

END

C.9.12 Relations and Functions 12

CONTEXT RelationsAndFunctions12

DESCRIPTION

Context `RelationsAndFunctions09` is extended.
The set *S* is constructed and evaluated.

`axm1` : Sets $s1$ to $s5$ are defined
`axm2` : The set *S* is defined
`thm3` : The set *S* is evaluated

EXTENDS RelationsAndFunctions09

CONSTANTS

S
s1
s2
s3
s4
s5

AXIOMS

axm1:
$s1 = \{x \cdot x \geq 0 \mid 2 * x + 1\}$
$\wedge$
$s2 = \{0, 1, 2, 4, 8, 16, 32, 64, 128, 512, 1024\}$
$\wedge$
$s3 = \{2, 3, 4, 5, 6, 7\}$
$\wedge$
$s4 = \{20, 21, 23, 39, 8, 1, 7\}$
$\wedge$
$s5 = \{x \cdot x \geq 0 \mid 3 * x\}$

axm2: $S = \{s1, s2, s3, s4, s5\}$

thm3: ⟨theorem⟩ *HasElementProperty*(S) = *TRUE*

END

C.9.13 Relations and Functions 13

CONTEXT RelationsAndFunctions13

DESCRIPTION

The function *HasComplementProperty* maps a family of integer sets to *TRUE* iff for every pair of distinct element sets X and Y neither $X \subseteq Y$ nor $Y \subseteq X$.
Otherwise the family of sets is mapped to *FALSE*.
NOTE: $X \subseteq Y \Rightarrow X \setminus Y = \varnothing$

`axm1` : *HasElementProperty* is declared
`axm2` : *HasElementProperty* is defined

CONSTANTS

HasComplementProperty

AXIOMS

`axm1`: $HasComplementProperty \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \rightarrow BOOL$

`axm2`:

$\forall S \cdot S \in \mathbb{P}(\mathbb{P}(\mathbb{Z})) \Rightarrow$
$(HasComplementProperty(S) = TRUE$
$\Leftrightarrow$
$(\forall X, Y \cdot (X \in S \land Y \in S \land X \neq Y)$
$\Rightarrow (X \setminus Y \neq \varnothing \land Y \setminus X \neq \varnothing)))$

END

C.9.14 Relations and Functions 14

CONTEXT RelationsAndFunctions14

DESCRIPTION

Context `RelationsAndFunctions13` is extended.
The family of sets A is constructed and declared to not have the *HasComplementProperty* property.

`axm1` : Sets $a1$ to $a4$ are declared
`axm2` : Set A is declared
`thm3` : Set A is evaluated

EXTENDS RelationsAndFunctions13

CONSTANTS

A
a1
a2
a3
a4

AXIOMS

`axm1`:

$a1 = \{1, 2\}$
$\land$
$a2 = \{1, 4\}$
$\land$
$a3 = \{1, 3, 5, 8\}$

$\wedge$

$a4 = \{-2, -1, 0, 1, 2\}$

axm2: $A = \{a1, a2, a3, a4\}$

thm3: ⟨theorem⟩ *HasComplementProperty*(A) = *FALSE*

END

C.9.15 Relations and Functions 15

CONTEXT RelationsAndFunctions15

DESCRIPTION

Context `RelationsAndFunctions13` is extended.
The family of sets *B* is constructed and declared to not have the *HasComplementProperty* property.
NOTE: The provers are sensitive to the order in which the sets $b1$ to $b6$ are included in *B*.

`axm1` : Sets $b1$ to $b6$ are declared
`axm2` : Set *B* is declared as the family of sets $b1$ to $b6$
`thm3` : Set *B* is evaluated

EXTENDS RelationsAndFunctions13

CONSTANTS

B

b1

b2

b3

b4

b5

b6

AXIOMS

axm1:

$b1 = \{2, 5\}$

$\wedge$

$b2 = \{4, 66, 8, 12\}$

$\wedge$

$b3 = \{123, 65, 74, 5, 225, 21\}$

$\wedge$

$b4 = \{225, 369, -234, 2\}$

$\wedge$

$b5 = \{2, -23, 21, 25\}$

$\wedge$

$b6 = \{2, 5, 44, 79, 98, 1, 4\}$

axm2: $B = \{b1, b2, b3, b4, b5, b6\}$

thm3: ⟨theorem⟩ *HasComplementProperty*(B) = *FALSE*

END

C.9.16 Relations and Functions 16

CONTEXT RelationsAndFunctions16

DESCRIPTION

Context `RelationsAndFunctions13` is extended.
Several sets are evaluated for the *HasComplementProperty* property.

`axm1` : Set $c1$ is declared
`thm2` : Set $\{c1\}$ is evaluated
`thm3` : Set $\{c1, \{4\}\}$ is evaluated
`axm4` : Set C is declared as the family of element sets $c2$ and $c3$
`thm5` : Set C is evaluated

EXTENDS RelationsAndFunctions13
CONSTANTS
c1
c2
c3
C
AXIOMS
`axm1`: $c1 = \{1, 2, 4\}$
`thm2`: ⟨theorem⟩ $\mathit{HasComplementProperty}(\{c1\}) = \mathit{TRUE}$
`thm3`: ⟨theorem⟩ $\mathit{HasComplementProperty}(\{c1, \{4\}\}) = \mathit{FALSE}$
`axm4`:
$c2 = \{2, 5, 8, 11, 14, 17, 21, 24, 30\}$
$\land$
$c3 = \{12, 5, 8, 11, 14, 17, 21, 24, 30\}$
$\land$
$C = \{c2, c3\}$
`thm5`: ⟨theorem⟩ $\mathit{HasComplementProperty}(C) = \mathit{TRUE}$
END

C.9.17 Relations and Functions 17

CONTEXT RelationsAndFunctions17

DESCRIPTION

Context `RelationsAndFunctions13` is extended.
The set D is constructed and evaluated for the *HasComplementProperty* property.

`axm1` : Set $d1$ is declared
`axm2` : Set $d2$ is declared
`thm3` : A lemma is included to assist with `thm5`
`axm4` : Set D is declared
`thm5` : Set D is evaluated

EXTENDS RelationsAndFunctions13

CONSTANTS

d1

d2

D

AXIOMS

axm1: $d1 = \{x \cdot x \geq 0 \land x \leq 10 \mid 2 * x\}$

d1 = {0,2,4,...,20}

axm2: $d2 = \{x \cdot x < 2 \mid 3 * x + 1\}$

d2 = {...,-5,-2,0,4}

thm3: ⟨theorem⟩ $\exists x \cdot x \in d2 \land x \notin d1$

lemma

axm4: $D = \{d1, d2\}$

thm5: ⟨theorem⟩ $HasComplementProperty(D) = TRUE$

END

C.9.18 Relations and Functions 18

Referenced in Sections 7.3.3 and 7.4.1.3.

CONTEXT RelationsAndFunctions18

DESCRIPTION

A symmetric difference tautology is evaluated.
thm6 appears in Van der Poll (2000, p. 81).

axm1 : Set A is declared
axm2 : Set B is declared
axm3 : Set C is declared
axm4 : Function $PLUS$ is declared
axm5 : Function $PLUS$ is defined as the symmetric difference (+) of two sets
thm6 : $(A + B) + C = A + (B + C)$

CONSTANTS

A

B

C

PLUS

AXIOMS

axm1: $A \subseteq \mathbb{Z}$

axm2: $B \subseteq \mathbb{Z}$

axm3: $C \subseteq \mathbb{Z}$

axm4: $PLUS \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \rightarrow \mathbb{P}(\mathbb{Z})$

axm5:

$\forall x, y \cdot x \subseteq \mathbb{Z} \land y \subseteq \mathbb{Z} \Rightarrow$
$PLUS(x \mapsto y) = (x \setminus y) \cup (y \setminus x)$

thm6: ⟨theorem⟩ $PLUS(PLUS(A \mapsto B) \mapsto C) = PLUS(A \mapsto PLUS(B \mapsto C))$

END

C.9.19 Relations and Functions 19

Referenced in Section 7.3.3.

CONTEXT RelationsAndFunctions19

DESCRIPTION

A symmetric difference tautology is evaluated.
In *RelationsAndFunctions*18, the tautology was stated directly without introducing additional constants for nested functors. In this context, additional constants are declared to avoid nested functors.
`thm10` appears in Van der Poll (2000, p. 82).

`axm1` : Set A is declared
`axm2` : Set B is declared
`axm3` : Set C is declared
`axm4` : Function $PLUS$ is declared
`axm5` : Function $PLUS$ is defined as the symmetric difference of two sets
`axm6` : $A\_plus\_B = A + B$
`axm7` : $AB\_plus\_C = (A + B) + C$
`axm8` : $B\_plus\_C = B + C$
`axm9` : $A\_plus\_BC = A + (B + C)$
`thm10` : $(A + B) + C = A + (B + C)$

CONSTANTS

A
B
C
PLUS
A_plus_B
AB_plus_C
A_plus_BC
B_plus_C

AXIOMS

`axm1`: $A \subseteq \mathbb{Z}$
`axm2`: $B \subseteq \mathbb{Z}$
`axm3`: $C \subseteq \mathbb{Z}$
`axm4`: $PLUS \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \to \mathbb{P}(\mathbb{Z})$
`axm5`:
$\forall x, y \cdot x \subseteq \mathbb{Z} \land y \subseteq \mathbb{Z} \Rightarrow$
$PLUS(x \mapsto y) = (x \setminus y) \cup (y \setminus x)$
`axm6`: $A\_plus\_B = PLUS(A \mapsto B)$
`axm7`: $AB\_plus\_C = PLUS(A\_plus\_B \mapsto C)$
`axm8`: $B\_plus\_C = PLUS(B \mapsto C)$
`axm9`: $A\_plus\_BC = PLUS(A \mapsto B\_plus\_C)$
`thm10`: ⟨theorem⟩ $AB\_plus\_C = A\_plus\_BC$

END

C.9.20 Relations and Functions 20

Referenced in Section 7.10.3.

CONTEXT RelationsAndFunctions20

DESCRIPTION

This example appears in Van der Poll (2000, p. 97).
Let *Persons* be a set of 5-tuples with the properties name, role, department, salary and address.
IDs is a set of positive integers representing employee IDs.
Function *Emp* maps IDs to persons.
Let *p* be the ID of an employee whose salary is changing by *increase*.
The set *Emp* must be updated; let the update be *EmpPrime*.
`axm7` relates *EmpPrime* (the relation *Emp* after the salary increase) to *Emp* in the following way:
Emp = *EmpPrime* except for the ID *p* where the salaries differ by exactly *increase*.

`axm1` : *Persons* is declared
`axm2` : *IDs* is declared
`axm3` : Function *Emp* is declared
`axm4` : Relation *EmpPrime* is declared
`axm5` : *p* is declared
`axm6` : *increase* is declared
`axm7` : The relationship between *Emp* and *EmpPrime* is defined
`thm8` : *EmpPrime* is proved functional

SETS

Names
Roles
Depts
Addresses

CONSTANTS

Persons
IDs
Emp
EmpPrime
p
increase

AXIOMS

`axm1`: $Persons = Names \times Roles \times Depts \times \mathbb{N}_1 \times Addresses$
salary $\in \mathbb{N}_1$

`axm2`: $IDs \subseteq \mathbb{N}_1$
ID is a subset of positive integers

`axm3`: $Emp \in IDs \rightarrow Persons$
Emp is a total function

`axm4`: $EmpPrime \in IDs \nleftrightarrow Persons$
EmpPrime is a total relation (i.e. defined for all IDs);
we want to establish that it is a function given its relationship with Emp

`axm5`: $p \in IDs$
Employee number whose salary is changing

`axm6`: $increase \in \mathbb{Z}$
Salary change amount

`axm7`:
$\forall x, n, r, d, sPrime, a \cdot$
$x \in IDs \Rightarrow$

(
$x \mapsto (n \mapsto r \mapsto d \mapsto sPrime \mapsto a) \in EmpPrime$
$\Leftrightarrow$
$((x \neq p \land Emp(x) = (n \mapsto r \mapsto d \mapsto sPrime \mapsto a))$
$\lor$
$(x = p \land (\exists s \cdot (sPrime = s + increase \land Emp(x) = (n \mapsto r \mapsto d \mapsto s \mapsto a)))))$
)

thm8: ⟨theorem⟩ $EmpPrime \in IDs \rightarrow Persons$

END

C.9.21 Relations and Functions 21

CONTEXT RelationsAndFunctions21

DESCRIPTION

This example appears in Van der Poll (2000, p. 56) to illustrate the resolution rule.

axm1 : Set *People* is defined
axm2 : Function *Parent* is declared
axm3 : Function *Ancestor* is declared
axm4 : Partial definition of *Ancestor* in terms of *Parent*
axm5 : Partial definition of *Ancestor* in terms of *Parent*
axm6 : Adam is a parent of Cain
axm7 : Cain is a parent of Enoch
thm8 : Therefore, Adam is an ancestor of Enoch

SETS

People

CONSTANTS

Adam
Cain
Enoch
Parent
Ancestor

AXIOMS

axm1: $partition(People, \{Adam\}, \{Cain\}, \{Enoch\})$

axm2: $Parent \in People \times People \rightarrow BOOL$

axm3: $Ancestor \in People \times People \rightarrow BOOL$

axm4:
$\forall x, y \cdot x \in People \land y \in People$
$\Rightarrow$
$(Parent(x \mapsto y) = TRUE \Rightarrow Ancestor(x \mapsto y) = TRUE)$

axm5:
$\forall x, y, z \cdot x \in People \land y \in People \land z \in People$
$\Rightarrow$
$(Parent(x \mapsto y) = TRUE \land Ancestor(y \mapsto z) = TRUE$
$\Rightarrow Ancestor(x \mapsto z) = TRUE)$

axm6: $Parent(Adam \mapsto Cain) = TRUE$

axm7: $Parent(Cain \mapsto Enoch) = TRUE$

thm8: ⟨theorem⟩ $Ancestor(Adam \mapsto Enoch) = TRUE$

END

C.9.22 Relations and Functions 22

CONTEXT RelationsAndFunctions22

DESCRIPTION

Various theorems using the definition of *subset* in axm2.

axm1 : Function *subset* is declared
axm2 : *subset* is defined
axm3 : Sets *C* and *D* defined
thm4 : Elements of *D* are subsets of *C*
thm5 - thm10 : Various *subset* theorems

CONSTANTS

C
D
subset
a
b

AXIOMS

axm1: $subset \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \rightarrow \{TRUE, FALSE\}$

axm2:
$\forall A, B \cdot A \subseteq \mathbb{Z} \land B \subseteq \mathbb{Z} \Rightarrow$
(
$(subset(A \mapsto B) = TRUE)$
$\Leftrightarrow$
$(\forall x \cdot x \in A \Rightarrow x \in B)$
)

axm3: $C \subseteq \mathbb{Z} \land D = \mathbb{P}(C)$

thm4: ⟨theorem⟩ $\forall x \cdot x \in D \Leftrightarrow subset(x \mapsto C) = TRUE$

thm5: ⟨theorem⟩ $subset(\{1\} \mapsto \{2\}) = FALSE$

thm6: ⟨theorem⟩
$a = \{0, 1, 5\} \land b = \{0, 1, 2, 4, 5, 10\}$
$\Rightarrow$
$subset(a \mapsto b) = TRUE$

thm7: ⟨theorem⟩ $subset(\varnothing \mapsto \{0, 1\}) = TRUE$

thm8: ⟨theorem⟩ $subset(\{0\} \mapsto \{0, 1\}) = TRUE$

thm9: ⟨theorem⟩ $subset(\{1\} \mapsto \{0, 1\}) = TRUE$

thm10: ⟨theorem⟩ $subset(\{0, 1\} \mapsto \{0, 1\}) = TRUE$

END

C.9.23 Relations and Functions 23

Referenced in Section 7.8.3.

CONTEXT RelationsAndFunctions23

DESCRIPTION

The union of functions that agree on common domain elements is also a function.

axm1 : Function f is declared
axm2 : Function g is declared
axm3 : $f(x) = g(x)$ for common domain elements x
thm4 : The union of f and g is also a function

CONSTANTS

f

g

AXIOMS

axm1: $f \in \mathbb{N} \rightarrow \mathbb{N}$

axm2: $g \in \mathbb{N} \rightarrow \mathbb{N}$

axm3: $\forall x \cdot x \in dom(f) \cap dom(g) \Rightarrow f(x) = g(x)$

thm4: $\langle$theorem$\rangle$ $f \cup g \in \mathbb{N} \rightarrow \mathbb{N}$

END

C.9.24 Relations and Functions 24

Referenced in Section 7.4.2.

CONTEXT RelationsAndFunctions24

DESCRIPTION

Adding a pair $a \mapsto b$ to a partial function f where a is not in the domain of f yields another partial function.

axm1 : Partial function f, and integers a and b are declared
axm2 : a is not in the domain of f
thm3 : $f \cup \{a \mapsto b\}$ is another partial function

CONSTANTS

f

a

b

AXIOMS

axm1: $f \in \mathbb{Z} \pfun \mathbb{Z} \land a \in \mathbb{Z} \land b \in \mathbb{Z}$

axm2: $a \notin dom(f)$

thm3: $\langle$theorem$\rangle$ $f \cup \{a \mapsto b\} \in \mathbb{Z} \pfun \mathbb{Z}$

END

C.9.25 Relations and Functions 25

Referenced in Section 7.4.2.

CONTEXT RelationsAndFunctions25

DESCRIPTION

Let f be a total function from A to B. For a not in the domain of f, we have:
The augmented set $f \cup \{a \mapsto b\}$ is a total function from the augmented domain $A \cup \{a\}$ to the augmented range $B \cup \{b\}$.

axm1 : Elements a and b are declared
axm2 : Sets A and B are declared
axm3 : Total function f is declared
axm4 : a is not in the domain of f
thm5 : $f \cup \{a \mapsto b\}$ is also a total function from $A \cup \{a\}$ to $B \cup \{b\}$

SETS

U

CONSTANTS

f
A
B
a
b

AXIOMS

axm1: $a \in U \land b \in U$
axm2: $A \subseteq U \land B \subseteq U$
axm3: $f \in A \rightarrow B$
axm4: $a \notin dom(f)$
thm5: $\langle$theorem$\rangle$ $f \cup \{a \mapsto b\} \in A \cup \{a\} \rightarrow B \cup \{b\}$

END

C.9.26 Relations and Functions 26

Referenced in Section 7.4.2.

CONTEXT RelationsAndFunctions26

DESCRIPTION

Let f be a total function from A to B, g a total function from C to D. If A and C are disjoint, we have:
The set $f \cup g$ is a total function from $A \cup C$ to $B \cup D$.

axm1 : Sets A and B are declared
axm2 : Total function f is declared
axm3 : Sets C and D are declared
axm4 : A and C are disjoint sets
axm5 : Total function g is declared
thm6 : $f \cup g$ is also a total function from $A \cup C$ to $B \cup D$

SETS

U

CONSTANTS

f

A

B

g

C

D

AXIOMS

axm1: $A \subseteq U \land B \subseteq U$

axm2: $f \in A \rightarrow B$

axm3: $C \subseteq U \land D \subseteq U$

axm4: $A \cap C = \varnothing$

axm5: $g \in C \rightarrow D$

thm6: $\langle$theorem$\rangle$ $f \cup g \in A \cup C \rightarrow B \cup D$

END

C.9.27 Relations and Functions 27

CONTEXT RelationsAndFunctions27

DESCRIPTION

A relation R is declared as a partial relation on $\mathbb{Z}$.
A number of functions on the set of partial relations on $\mathbb{Z}$ are declared and defined:
IsReflexive, *IsSymmetric*, *IsTransitive* and *IsEquivRel* deal with properties of equivalence relations; and
IsIrreflexive, *IsTrichotomous* and *IsLinearOrdering* deal with properties of linear orderings.

axm1 - axm3 : Declaration of relation R
axm4 - axm11 : Declaration and definition of *IsReflexive*, *IsSymmetric*, *IsTransitive* and *IsEquivRel*
axm12 - axm17 : Declaration and definition of *IsIrreflexive*, *IsTrichotomous* and *IsLinearOrdering*

CONSTANTS

U

R R is a Relation on U

IsReflexive

IsSymmetric

IsTransitive

IsEquivRel

IsIrreflexive

IsTrichotomous

IsLinearOrdering

AXIOMS

axm1: $U \subseteq \mathbb{Z}$

axm2: $R \in U \leftrightarrow U$

axm3: $U = dom(R) \cup ran(R)$

axm4: $IsReflexive \in (U \leftrightarrow U) \rightarrow BOOL$

axm5: $IsSymmetric \in (U \leftrightarrow U) \rightarrow BOOL$

axm6: $IsTransitive \in (U \leftrightarrow U) \rightarrow BOOL$

axm7: $IsEquivRel \in (U \leftrightarrow U) \rightarrow BOOL$

axm8:
$(\forall n \cdot n \in U \Rightarrow n \mapsto n \in R)$
$\Leftrightarrow$
$IsReflexive(R) = TRUE$

axm9:
$(\forall m, n \cdot m \mapsto n \in R \Rightarrow n \mapsto m \in R)$
$\Leftrightarrow$
$IsSymmetric(R) = TRUE$

axm10:
$(\forall l, m, n \cdot l \mapsto m \in R \land m \mapsto n \in R \Rightarrow l \mapsto n \in R)$
$\Leftrightarrow$
$IsTransitive(R) = TRUE$

axm11:
$(IsReflexive(R) = TRUE \land IsSymmetric(R) = TRUE \land IsTransitive(R) = TRUE)$
$\Leftrightarrow$
$IsEquivRel(R) = TRUE$

axm12: $IsIrreflexive \in (U \leftrightarrow U) \rightarrow BOOL$

axm13: $IsTrichotomous \in (U \leftrightarrow U) \rightarrow BOOL$

axm14: $IsLinearOrdering \in (U \leftrightarrow U) \rightarrow BOOL$

axm15:
$(\forall n \cdot n \in U \Rightarrow n \mapsto n \notin R)$
$\Leftrightarrow$
$IsIrreflexive(R) = TRUE$

axm16:
$(\forall x, y \cdot x \mapsto y \in R \Rightarrow$
$(x = y$
$\lor$
$(y \mapsto x \notin R)))$
$\Leftrightarrow$
$IsTrichotomous(R) = TRUE$

axm17:
$(IsIrreflexive(R) = TRUE$
$\land$
$IsTransitive(R) = TRUE$
$\land$
$IsTrichotomous(R) = TRUE)$
$\Leftrightarrow$
$IsLinearOrdering(R) = TRUE$

END

C.9.28 Relations and Functions 28

CONTEXT RelationsAndFunctions28

DESCRIPTION

Context `RelationsAndFunctions27` is extended.
Each theorem defines R and evaluates one of the relational properties previously defined.

`thm1` - `thm30` : Relational properties defined in Context `RelationsAndFunctions27` are evaluated for

various values of R

EXTENDS RelationsAndFunctions27

AXIOMS

thm1: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5\} \Rightarrow$
IsTransitive$(R) =$ *FALSE*

thm2: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5\} \Rightarrow$
IsReflexive$(R) =$ *FALSE*

thm3: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5\} \Rightarrow$
IsSymmetric$(R) =$ *FALSE*

thm4: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5\} \Rightarrow$
IsEquivRel$(R) =$ *FALSE*

thm5: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5\} \Rightarrow$
IsIrreflexive$(R) =$ *TRUE*

thm6: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5\} \Rightarrow$
IsTrichotomous$(R) =$ *TRUE*

thm7: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5\} \Rightarrow$
IsLinearOrdering$(R) =$ *FALSE*

thm8: ⟨theorem⟩
$R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 2\} \Rightarrow$
IsTransitive$(R) =$ *TRUE*

thm9: ⟨theorem⟩
$R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 2\} \Rightarrow$
IsReflexive$(R) =$ *TRUE*

thm10: ⟨theorem⟩
$R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 2\} \Rightarrow$
IsSymmetric$(R) =$ *TRUE*

thm11: ⟨theorem⟩
$R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3, 4 \mapsto 4, 3 \mapsto 4, 4 \mapsto 3\} \Rightarrow$
IsEquivRel$(R) =$ *TRUE*

thm12: ⟨theorem⟩
$R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 2\} \Rightarrow$
IsIrreflexive$(R) =$ *FALSE*

thm13: ⟨theorem⟩
$R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 2\} \Rightarrow$
IsTrichotomous$(R) =$ *FALSE*

thm14: ⟨theorem⟩
$R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 2\} \Rightarrow$
IsLinearOrdering$(R) =$ *FALSE*

thm15: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 1 \mapsto 3, 3 \mapsto 4, 1 \mapsto 4, 2 \mapsto 4\} \Rightarrow$
IsTransitive$(R) =$ *TRUE*

thm16: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 1 \mapsto 3, 3 \mapsto 4, 1 \mapsto 4, 2 \mapsto 4\} \Rightarrow$
IsReflexive$(R) =$ *FALSE*

thm17: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 1 \mapsto 3, 3 \mapsto 4, 1 \mapsto 4, 2 \mapsto 4\} \Rightarrow$
$\mathit{IsSymmetric}(R) = \mathit{FALSE}$

thm18: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 1 \mapsto 3, 3 \mapsto 4, 1 \mapsto 4, 2 \mapsto 4\} \Rightarrow$
$\mathit{IsEquivRel}(R) = \mathit{FALSE}$

thm19: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 1 \mapsto 3, 3 \mapsto 4, 1 \mapsto 4, 2 \mapsto 4\} \Rightarrow$
$\mathit{IsIrreflexive}(R) = \mathit{TRUE}$

thm20: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 1 \mapsto 3, 3 \mapsto 4, 1 \mapsto 4, 2 \mapsto 4\} \Rightarrow$
$\mathit{IsTrichotomous}(R) = \mathit{TRUE}$

thm21: ⟨theorem⟩
$R = \{1 \mapsto 2, 2 \mapsto 3, 1 \mapsto 3, 3 \mapsto 4, 1 \mapsto 4, 2 \mapsto 4\} \Rightarrow$
$\mathit{IsLinearOrdering}(R) = \mathit{TRUE}$

thm22: ⟨theorem⟩
$R = \{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3, 2 \mapsto 3, 3 \mapsto 2\} \Rightarrow$
$\mathit{IsEquivRel}(R) = \mathit{TRUE}$

thm23: ⟨theorem⟩
$R = \{4 \mapsto 5, 5 \mapsto 6, 4 \mapsto 6, 7 \mapsto 8\} \Rightarrow$
$\mathit{IsLinearOrdering}(R) = \mathit{TRUE}$

thm24: ⟨theorem⟩
$R = \{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3, 2 \mapsto 3\} \Rightarrow$
$\mathit{IsEquivRel}(R) = \mathit{FALSE}$

thm25: ⟨theorem⟩
$R = \{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3, 4 \mapsto 5, 5 \mapsto 6, 4 \mapsto 6, 4 \mapsto 4, 6 \mapsto 6\} \Rightarrow$
$\mathit{IsReflexive}(R) = \mathit{FALSE}$

thm26: ⟨theorem⟩
$R = \{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3, 4 \mapsto 5, 5 \mapsto 6, 4 \mapsto 6, 4 \mapsto 4, 6 \mapsto 6\} \Rightarrow$
$\mathit{IsEquivRel}(R) = \mathit{FALSE}$

thm27: ⟨theorem⟩
$R = \{4 \mapsto 5, 5 \mapsto 6, 4 \mapsto 6, 7 \mapsto 8, 8 \mapsto 7\} \Rightarrow$
$\mathit{IsLinearOrdering}(R) = \mathit{FALSE}$

thm28: ⟨theorem⟩
$R = \{4 \mapsto 5, 5 \mapsto 6, 4 \mapsto 6, 7 \mapsto 8, 4 \mapsto 4\} \Rightarrow$
$\mathit{IsLinearOrdering}(R) = \mathit{FALSE}$

thm29: ⟨theorem⟩
$R = \{n \cdot n \in \mathbb{Z} \mid n \mapsto n\} \Rightarrow$
$\mathit{IsEquivRel}(R) = \mathit{TRUE}$

thm30: ⟨theorem⟩
$R = \{n, m \cdot n \in \mathbb{Z} \land m \in \mathbb{Z} \land n < m \mid n \mapsto m\} \Rightarrow$
$\mathit{IsLinearOrdering}(R) = \mathit{TRUE}$

END

C.10 Set Theory

The final section of this appendix consists of a collection of contexts that illustrate proof obligations arising from proofs that include basic set-theoretic operators and properties: elementhood, subset, power set, union, intersection, relative complement, generalised union and generalised intersection.

C.10.1 Set Theory 1

Referenced in Sections 7.2.3 and 8.2.3.

CONTEXT SetTheory01

DESCRIPTION

Three examples of power set evaluations.

`thm1` : Power set of {{0}, {1}, {0,2}, {1,2}} evaluated
`thm2` : Power set of {∅, {∅}, {{∅}}} is not empty
`thm3` : Power set of {∅, {∅}, {{∅}}} evaluated

AXIOMS

`thm1`: ⟨theorem⟩
$$\mathbb{P}(\{\{0\},\{1\},\{0,2\},\{1,2\}\}) = \{\varnothing,\{\{0\}\},\{\{1\}\},\{\{0,2\}\},\{\{1,2\}\},\{\{0\},\{1\}\},\{\{0\},\{0,2\}\},\{\{0\},\{1,2\}\},\{\{1\},\{0,2\}\},\{\{1\},\{1,2\}\},\{\{0,2\},\{1,2\}\},\{\{0\},\{1\},\{0,2\}\},\{\{0\},\{1\},\{1,2\}\},\{\{0\},\{0,2\},\{1,2\}\},\{\{1\},\{0,2\},\{1,2\}\},\{\{0\},\{1\},\{0,2\},\{1,2\}\}\}$$

`thm2`: ⟨theorem⟩ $\mathbb{P}(\{\varnothing,\{\varnothing\},\{\{\varnothing\}\}\}) \neq \varnothing \,\wr\, \mathbb{P}(\mathbb{P}(\mathbb{P}(\mathbb{P}(\mathbb{P}(\mathbb{Z})))))$

`thm3`: ⟨theorem⟩
$$\mathbb{P}(\{\varnothing \,\wr\, \mathbb{P}(\mathbb{P}(\mathbb{P}(\mathbb{P}(\mathbb{Z})))),\{\varnothing\},\{\{\varnothing\}\}\}) = \{\varnothing,\{\varnothing\},\{\{\varnothing\}\},\{\{\{\varnothing\}\}\},\{\varnothing,\{\varnothing\}\},\{\varnothing,\{\{\varnothing\}\}\},\{\{\varnothing\},\{\{\varnothing\}\}\},\{\varnothing,\{\varnothing\},\{\{\varnothing\}\}\}\}$$

END

C.10.2 Set Theory 2

Referenced in Sections 7.2.3 and 7.4.1.3.

CONTEXT SetTheory02

DESCRIPTION

Examples of power set evaluations. `thm1` appears in Van der Poll (2000, p. 82).

`thm1` : Power set of {0,1} evaluated
`thm2` : Power set of {0,1,2} evaluated

AXIOMS

`thm1`: ⟨theorem⟩ $\mathbb{P}(\{0,1\}) = \{\varnothing,\{0\},\{1\},\{0,1\}\}$

`thm2`: ⟨theorem⟩
$$\mathbb{P}(\{0,1,2\}) = \{\varnothing,\{0\},\{1\},\{2\},\{0,1\},\{0,2\},\{1,2\},\{0,1,2\}\}$$

END

C.10.3 Set Theory 3

Referenced in Section 7.2.3.

CONTEXT SetTheory03

DESCRIPTION

Evaluation of a generalised intersection of a family of sets of integers.

thm1 : inter({{0,1,2},{1,2,3},{3,1},{4,7,8,1}}) evaluated

AXIOMS

thm1: ⟨theorem⟩ $\mathit{inter}(\{\{0,1,2\},\{1,2,3\},\{3,1\},\{4,7,8,1\}\}) = \{1\}$

END

C.10.4 Set Theory 4

Referenced in Section 7.2.3.

CONTEXT SetTheory04

DESCRIPTION

Power set calculation taken from Van der Poll (2000, p. 61), Example 3.7.

thm1 : Power set of {{1}} evaluated

AXIOMS

thm1: ⟨theorem⟩ $\mathbb{P}(\{\{1\}\}) = \{\varnothing, \{\{1\}\}\}$

END

C.10.5 Set Theory 5

Referenced in Section 8.2.3.

CONTEXT SetTheory05

DESCRIPTION

Example of a power set calculation for a 12-element set of integers.

thm1 : Power set of {0,1,2,3,4,5,6,7,8,9,10,11} evaluated

AXIOMS

thm1: ⟨theorem⟩
$\mathbb{P}(\{0,1,2,3,4,5,6,7,8,9,10,11\})$
$= \{\varnothing, \{0\}, \{1\}, \{0,1\}, \{2\}, \{0,2\}, \{1,2\}, \{0,$
$1,2\}, \{3\}, \{0,3\}, \{1,3\}, \{0,1,3\}, \{2,3\}, \{0,2,3\}, \{1,2,3\}, \{0,1,2,$
$3\}, \{4\}, \{0,4\}, \{1,4\}, \{0,1,4\}, \{2,4\}, \{0,2,4\}, \{1,2,4\}, \{0,1,2,4\},$

$\{3,4\}, \{0,3,4\}, \{1,3,4\}, \{0,1,3,4\}, \{2,3,4\}, \{0,2,3,4\}, \{1,2,3,4\},$
$\ldots,$
$\{1,2,3,5,6,7,8,9,10,11\}, \{0,1,2,3,5,6,7,8,9,10,11\}, \{4,5,6,7,$
$8,9,10,11\}, \{0,4,5,6,7,8,9,10,11\}, \{1,4,5,6,7,8,9,10,11\}, \{0,$
$1,4,5,6,7,8,9,10,11\}, \{2,4,5,6,7,8,9,10,11\}, \{0,2,4,5,6,7,$
$8,9,10,11\}, \{1,2,4,5,6,7,8,9,10,11\}, \{0,1,2,4,5,6,7,8,9,10,$
$11\}, \{3,4,5,6,7,8,9,10,11\}, \{0,3,4,5,6,7,8,9,10,11\}, \{1,3,4,$
$5,6,7,8,9,10,11\}, \{0,1,3,4,5,6,7,8,9,10,11\}, \{2,3,4,5,6,7,$
$8,9,10,11\}, \{0,2,3,4,5,6,7,8,9,10,11\}, \{1,2,3,4,5,6,7,8,9,$
$10,11\}, \{0,1,2,3,4,5,6,7,8,9,10,11\}\}$

END

C.10.6 Set Theory 6

CONTEXT SetTheory06

DESCRIPTION

Example illustrating the application of OTTER Heuristic #1.

`thm1` : $\mathbb{P}(\{\{1\}\})$ and $\{\varnothing, \{\{1\}\}\}$ are proven equal at the level of elementhood.

AXIOMS

`thm1`: ⟨theorem⟩ $\forall x \cdot x \in \mathbb{P}(\{\{1\}\}) \Leftrightarrow x \in \{\varnothing, \{\{1\}\}\}$

END

C.10.7 Set Theory 7

Referenced in Section 7.2.3.

CONTEXT SetTheory07

DESCRIPTION

Examples of generalised intersection using constants A and B.

`axm1` : Constants A and B declared
`thm2` : Generalised intersection of $\{A, B\}$ evaluated

CONSTANTS

A
B

AXIOMS

`axm1`: $A = \{1,2,3\} \land B = \{2,3,4\}$
`thm2`: ⟨theorem⟩ $inter(\{A,B\}) = \{2,3\}$

END

C.10.8 Set Theory 8

CONTEXT SetTheory08

DESCRIPTION

Let *ParFuncsOnU* be a set of partial functions on a set U.
If every pair of functions f and g in *ParFuncsOnU* satisfy the condition $(f = g \lor dom(f) \cap dom(g) = \varnothing)$, then the generalised union of *ParFuncsOnU* is also a partial function of U.

axm1 : *ParFuncsOnU* defined
axm2 : Every pair of functions in *ParFuncsOnU* are either equal or have disjoint domains
thm3 : The generalised union of *ParFuncsOnU* is also a partial function on U

SETS

U

CONSTANTS

ParFuncsOnU

AXIOMS

axm1: $ParFuncsOnU \subseteq U \pfun U$

axm2:

$\forall f, g \cdot (f \in ParFuncsOnU \land g \in ParFuncsOnU)$
$\Rightarrow$
$(f = g \lor dom(f) \cap dom(g) = \varnothing)$

thm3: $\langle$theorem$\rangle$ $union(ParFuncsOnU) \in U \pfun U$

END

C.10.9 Set Theory 9

CONTEXT SetTheory09

DESCRIPTION

Let *Functions* be a set of partial functions on $\mathbb{Z}$.
If every pair of functions f and g in *Functions* satisfy the condition $(f \subseteq g \lor g \subseteq f)$, then the generalised union of *Functions* is also a partial function on $\mathbb{Z}$.

axm1 : *Functions* defined
axm2 : For every pair of functions f and g in *Functions*, either $f \subseteq g$ or $g \subseteq f$ holds
thm3 : The generalised union of *Functions* is again a partial function on $\mathbb{Z}$

CONSTANTS

Functions

AXIOMS

axm1: $Functions \subseteq \mathbb{Z} \pfun \mathbb{Z}$

axm2:

$\forall f, g \cdot f \in Functions \land g \in Functions$
$\Rightarrow$
$f \subseteq g \lor g \subseteq f$

thm3: ⟨theorem⟩ $union(Functions) \in \mathbb{Z} \nrightarrow \mathbb{Z}$

END

C.10.10 Set Theory 10

Referenced in Section 7.7.3.

CONTEXT SetTheory10

DESCRIPTION

Evaluation of the inverse of a function F.

axm1 : Set-theoretic type declaration for F
thm2 : Inverse of F evaluated

SETS

U

CONSTANTS

F

a

b

AXIOMS

axm1: $F \in \mathbb{P}(\mathbb{P}(U)) \rightarrow \mathbb{P}(\mathbb{P}(U))$

thm2: ⟨theorem⟩

$F = \{\varnothing \mapsto a, \{\varnothing\} \mapsto b, a \mapsto b\}$

$\Leftrightarrow$

$F^{-1} = \{a \mapsto \varnothing, b \mapsto \{\varnothing\}, b \mapsto a\}$

END

C.10.11 Set Theory 11

Referenced in Section 7.2.3.

CONTEXT SetTheory11

DESCRIPTION

Evaluation of the power set of $\{a, b\}$ where a and b are unspecified integers.
Note that the case where $a = b$ is not excluded.

axm1 : Set-theoretic type declaration for a and b
thm2 : Power set of $\{a, b\}$ evaluated

CONSTANTS

a

b

AXIOMS

axm1: $a \in \mathbb{Z} \land b \in \mathbb{Z}$

thm2: ⟨theorem⟩ $\mathbb{P}(\{a,b\}) = \{\varnothing, \{a\}, \{b\}, \{a,b\}\}$

END

C.10.12 Set Theory 12

CONTEXT SetTheory12

DESCRIPTION

The definition of set-theoretic equality stated at the level of elementhood. See e.g., Enderton (1977, p. 17).

thm1 : Sets are equal when they contain the same elements

SETS

U

AXIOMS

thm1: ⟨theorem⟩
$\forall A, B \cdot (A \subseteq U \wedge B \subseteq U \Rightarrow$
$(A = B$
$\Rightarrow$
$(\forall x \cdot x \in A \Leftrightarrow x \in B)$
$))$

END

C.10.13 Set Theory 13

CONTEXT SetTheory13

DESCRIPTION

Example from Van der Poll (2000, p. 82) using explicit constructs.

axm1 : Sets A, B and C are defined
axm2 : Set D is defined as the power set of C
axm3 : Set E is defined
thm4 : $D = E$, i.e. the explicitly defined E is also the power set of C

CONSTANTS

A
B
C
D
E

AXIOMS

axm1: $A = \{0\} \wedge B = \{1\} \wedge C = \{0,1\}$
axm2: $D = \mathbb{P}(C)$
axm3: $E = \{\varnothing, A, B, C\}$
thm4: ⟨theorem⟩ $D = E$

END

C.10.14 Set Theory 14

CONTEXT SetTheory14

DESCRIPTION

This example appears in Labuschagne and Van der Poll (1999).

`thm1` : For any sets A and B, if A is a subset of B, then $\mathbb{P}(A)$ is a subset of $\mathbb{P}(B)$

SETS

U

AXIOMS

`thm1`: ⟨theorem⟩
$\forall A, B \cdot A \subseteq U \land B \subseteq U$
$\Rightarrow$
$(A \subseteq B \Leftrightarrow \mathbb{P}(A) \subseteq \mathbb{P}(B))$

END

C.10.15 Set Theory 15

Referenced in Sections 7.4.3 and 8.3.1.

CONTEXT SetTheory15

DESCRIPTION

Two properties of power sets are presented: Let A and B be any sets.
1. The power set of $A \cap B$ is equal to the intersection of $\mathbb{P}(A)$ and $\mathbb{P}(B)$
2. However, the power set of $A \cup B$ is the union of $\mathbb{P}(A)$ and $\mathbb{P}(B)$ iff $A \subseteq B$ or $B \subseteq A$

`axm1` : A and B are declared, both of type
`thm2` : The power set of $A \cap B$ is equal to the intersection of $\mathbb{P}(A)$ and $\mathbb{P}(B)$
`thm3` : Lemma needed to prove `thm4`
`thm4` : The power set of $A \cup B$ is the union of $\mathbb{P}(A)$ and $\mathbb{P}(B)$ iff $A \subseteq B$ or $B \subseteq A$

SETS

U

CONSTANTS

A
B

AXIOMS

`axm1`: $A \subseteq U \land B \subseteq U$
`thm2`: ⟨theorem⟩ $\mathbb{P}(A \cap B) = \mathbb{P}(A) \cap \mathbb{P}(B)$
`thm3`: ⟨theorem⟩
$\forall X, Y \cdot X \subseteq U \land \mathbb{P}(X \cup Y) = \mathbb{P}(X) \cup \mathbb{P}(Y)$
$\Rightarrow X \cup Y \in \mathbb{P}(X) \cup \mathbb{P}(Y)$
Lemma
`thm4`: ⟨theorem⟩ $\mathbb{P}(A \cup B) = \mathbb{P}(A) \cup \mathbb{P}(B) \Leftrightarrow A \subseteq B \lor B \subseteq A$

END

C.10.16 Set Theory 16

Referenced in Section 7.4.1.3.

CONTEXT SetTheory16

DESCRIPTION

Example to illustrate the SMT provers' logic over integers.

axm1 : A is the set of multiples of 4
axm2 : B is the set of multiples of 9
axm3 : C is the set of multiples of 10
thm4 : The intersection of A, B and C is evaluated

CONSTANTS

A
B
C

AXIOMS

axm1: $A = \{x \cdot x \in \mathbb{Z} \mid 4 * x\}$
axm2: $B = \{x \cdot x \in \mathbb{Z} \mid 9 * x\}$
axm3: $C = \{x \cdot x \in \mathbb{Z} \mid 10 * x\}$
thm4: ⟨theorem⟩ $A \cap B \cap C = \{x \cdot x \in \mathbb{Z} \mid 2 * 2 * 3 * 3 * 5 * x\}$

END

C.10.17 Set Theory 17

Referenced in Section 7.2.3.

CONTEXT SetTheory17

DESCRIPTION

Examples of easily proved set-theoretic results.

SETS

U

CONSTANTS

x
y
z
PLUS

AXIOMS

axm1: $x \subseteq U \land y \subseteq U$
thm1: ⟨theorem⟩ $x \cap y = y \cap x$
thm2: ⟨theorem⟩ $x \cup y = y \cup x$
thm3: ⟨theorem⟩ $x \cap y = ((x \cup y) \setminus (x \setminus y)) \setminus (y \setminus x)$
thm4: ⟨theorem⟩ $U \setminus (U \setminus x) = x$

thm5: ⟨theorem⟩ $x = x \cap (x \cup y)$

thm6: ⟨theorem⟩ $x = x \cup (x \cap y)$

thm7: ⟨theorem⟩ $(x \cap y) \setminus z = (x \setminus z) \cap (y \setminus z)$

thm8: ⟨theorem⟩ $U \setminus (x \cup y) = (U \setminus x) \cap (U \setminus y)$

thm9: ⟨theorem⟩ $U \setminus (x \cap y) = (U \setminus x) \cup (U \setminus y)$

thm10: ⟨theorem⟩ $x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$

thm11: ⟨theorem⟩ $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$

thm12: ⟨theorem⟩ $x \subseteq y \Rightarrow x \cup y = y$

thm13: ⟨theorem⟩ $x \subseteq y \Rightarrow x \cap y = x$

thm14: ⟨theorem⟩ $x \subseteq y \Rightarrow x \setminus y = \varnothing$

thm15: ⟨theorem⟩ $x \subseteq y \Rightarrow y \setminus x = y \setminus (x \cap y)$

axm16: $PLUS \in \mathbb{P}(U) \times \mathbb{P}(U) \to \mathbb{P}(U)$

axm17:

$\forall X, Y \cdot X \subseteq U \wedge Y \subseteq U \Rightarrow$
$PLUS(X \mapsto Y) = (X \setminus Y) \cup (Y \setminus X)$

thm18: ⟨theorem⟩ $x \cup y \subseteq z \Rightarrow z \setminus PLUS(x \mapsto y) = (x \cap y) \cup (z \setminus (x \cup y))$

thm19: ⟨theorem⟩ $PLUS(x \mapsto y) = \varnothing \Rightarrow x = y$

thm20: ⟨theorem⟩ $z \setminus PLUS(x \mapsto y) = (x \cap (y \cap z)) \cup (z \setminus (x \cup y))$

thm21: ⟨theorem⟩ $(x \setminus y) \cap PLUS(x \mapsto y) = x \cap (U \setminus y)$

thm22: ⟨theorem⟩ $PLUS(x \mapsto y) = (x \setminus y) \cup (y \setminus x)$

thm23: ⟨theorem⟩

$PLUS(PLUS(x \mapsto y) \mapsto z)$
$=$
$((x \setminus (y \cup z)) \cup (y \setminus (x \cup z)))$
$\cup$
$((z \setminus (x \cup y)) \cup (x \cap (y \cap z)))$

thm24: ⟨theorem⟩ $((x \cup y) \cap ((U \setminus x) \cup z)) = (y \setminus x) \cup (x \cap z)$

thm25: ⟨theorem⟩ $\exists X \cdot X \subseteq U \wedge (((X \cup y) \cap ((U \setminus X) \cup z)) = y \cup z)$

thm26: ⟨theorem⟩

$(x \cap y) \subseteq z \wedge z \subseteq (x \cup y)$
$\Rightarrow$
$((x \cup y) \cap ((U \setminus x) \cup z)) = y \cup z$

thm27: ⟨theorem⟩

$x \subseteq y \Rightarrow$
$(z \setminus x) \setminus y = z \setminus y$

thm28: ⟨theorem⟩

$x \subseteq y \Rightarrow$
$(z \setminus y) \setminus x = z \setminus y$

thm29: ⟨theorem⟩

$x \subseteq y \Rightarrow$
$z \setminus (y \cup x) = z \setminus y$

thm30: ⟨theorem⟩

$x \subseteq y \Rightarrow$
$z \setminus (y \cap x) = z \setminus x$

thm31: ⟨theorem⟩

$x \subseteq y \Rightarrow$
$(z \setminus y) \cap x = \varnothing$

thm32: ⟨theorem⟩

$x \subseteq y \Rightarrow$
$(z \setminus x) \cap y = z \cap (y \setminus x)$

thm33: ⟨theorem⟩
$x \subseteq y \land y \subseteq z \Rightarrow$
$(z \setminus x) \cap y = y \setminus x$

thm34: ⟨theorem⟩ $x \setminus y = x \cap (U \setminus y)$

thm35: ⟨theorem⟩ $x \cap \varnothing = \varnothing$

thm36: ⟨theorem⟩ $x \cup \varnothing = x$

thm37: ⟨theorem⟩
$x \subseteq y \Rightarrow$
$(\exists V \cdot V \subseteq U \land y \setminus V = x)$

END

C.10.18 Set Theory 18

Referenced in Sections 9.3, 9.3.2 and 9.3.2.

CONTEXT SetTheory18

DESCRIPTION

Definitions of polynomials used in extending contexts.

axm1 : $p1(x) = 2x - 1$
axm2 : $p2(x) = 4x + 3$
axm3 : $p3(x) = x^2$
axm4 : $p4(x) = x^2 - 1$
axm5 : $p5(x) = 2x^2 - 1$
axm6 : $p6(x) = -x^4 - x^2 - 1$

CONSTANTS

p1
p2
p3
p4
p5
p6

AXIOMS

axm1: $p1 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \land 2 * x - 1 = y \mid x \mapsto y\}$
axm2: $p2 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \land 4 * x + 3 = y \mid x \mapsto y\}$
axm3: $p3 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \land x * x = y \mid x \mapsto y\}$
axm4: $p4 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \land x * x - 1 = y \mid x \mapsto y\}$
axm5: $p5 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \land 2 * x * x - 1 = y \mid x \mapsto y\}$
axm6: $p6 = \{x, y \cdot x \mapsto y \in \mathbb{Z} \times \mathbb{Z} \land -x * x * x * x - x * x - 1 = y \mid x \mapsto y\}$

END

C.10.19 Set Theory 19

Referenced in Sections 7.4.1.3 and 9.3.1.

CONTEXT SetTheory19

DESCRIPTION

Extends context `SetTheory18`.
Various polynomial intersections are explored.

`thm1` : $p1$ and $p2$ intersect
`thm2` : $p3$ and $p4$ do not intersect
`thm3` : $p1$ and $p4$ intersect
`thm4` : $p1$ and $p3$ intersect
`thm5` : $p3$ and $p5$ intersect
`thm6` : $p3$ and $p6$ do not intersect

EXTENDS SetTheory18
AXIOMS

`thm1`: ⟨theorem⟩ $p1 \cap p2 \neq \varnothing$
Intersecting polynomials, deg 1 and 1

`thm2`: ⟨theorem⟩ $p3 \cap p4 = \varnothing$
Non-intersecting polynomials, deg 2 and 2

`thm3`: ⟨theorem⟩ $p1 \cap p4 \neq \varnothing$
Intersecting polynomials, deg 1 and 2

`thm4`: ⟨theorem⟩ $p1 \cap p3 \neq \varnothing$
Intersecting polynomials, deg 1 and 2

`thm5`: ⟨theorem⟩ $p3 \cap p5 \neq \varnothing$
Intersecting polynomials, deg 2 and 2

`thm6`: ⟨theorem⟩ $p3 \cap p6 = \varnothing$
Non-intersecting polynomials, deg 2 and 4

END

C.10.20 Set Theory 20

Referenced in Sections 7.4.1.3 and 9.3.2.

CONTEXT SetTheory20

DESCRIPTION

Extends context `SetTheory18`.
Various polynomial intersection points are evaluated

`thm1` : $p1$ and $p2$ have intersection point $(-2, 5)$
`thm2` : $p1$ and $p4$ have intersection points $(0, -1)$ and $(2, 3)$
`thm3` : $p1$ and $p3$ have intersection point $(1, 1)$
`thm4` : $p3$ and $p5$ have intersection points $(1, 1)$ and $(-1, 1)$

EXTENDS SetTheory18
AXIOMS

`thm1`: ⟨theorem⟩ $p1 \cap p2 = \{-2 \mapsto -5\}$
Intersecting polynomials, deg 1 and 1

thm2: ⟨theorem⟩ $p1 \cap p4 = \{0 \mapsto -1, 2 \mapsto 3\}$
Intersecting polynomials, deg 1 and 2
thm3: ⟨theorem⟩ $p1 \cap p3 = \{1 \mapsto 1\}$
Intersecting polynomials, deg 1 and 2
thm4: ⟨theorem⟩ $p3 \cap p5 = \{1 \mapsto 1, -1 \mapsto 1\}$
Intersecting polynomials, deg 2 and 2

END

C.10.21 Set Theory 21

CONTEXT SetTheory21

DESCRIPTION

Extends context SetTheory18.
The existence of polynomial intersection points are evaluated

axm1 : *points* is defined as a set of integer ordered pairs (intersection points)
thm2 : $p1 \cap p2 \subseteq$ *points*
thm3 : $p1 \cap p4 \subseteq$ *points*
thm4 : $p1 \cap p3 \subseteq$ *points*
thm5 : $p3 \cap p5 \subseteq$ *points*

EXTENDS SetTheory18
CONSTANTS
points
AXIOMS
axm1: $points = \{1 \mapsto 1, -1 \mapsto 1, -2 \mapsto -5, 0 \mapsto -1, 2 \mapsto 3\}$
thm2: ⟨theorem⟩ $\exists point \cdot point \in points \wedge point \in p1 \cap p2$
Intersecting polynomials, deg 1 and 1
thm3: ⟨theorem⟩ $\exists point \cdot point \in points \wedge point \in p1 \cap p4$
Intersecting polynomials, deg 1 and 2
thm4: ⟨theorem⟩ $\exists point \cdot point \in points \wedge point \in p1 \cap p3$
Intersecting polynomials, deg 1 and 2
thm5: ⟨theorem⟩ $\exists point \cdot point \in points \wedge point \in p3 \cap p5$
Intersecting polynomials, deg 2 and 2

END

C.10.22 Set Theory 22

CONTEXT SetTheory22

DESCRIPTION

Example taken from Van der Poll (2000, p. 96).
In a mathematical forest consisting of (directed) tree graphs, adding a node p below a node q already in a tree of the forest does not change the set of root nodes.

axm1 : Let K be the set of all nodes in the forest

axm2 : Let R be the set of root nodes
axm3 : Let V be the set of vertices (represented as ordered pairs of nodes)
axm4 : The root nodes are the forest nodes with no incoming vertices
axm5 : Let p be a node that is not already in the forest
axm6 : Let q be a node in the forest
axm7 : Let K_pr be the set of nodes K with p added
axm8 : Let V_pr be the set of vertices with (q, p) added
axm9 : Let R_pr be the set of root after p was added
thm10 : The set of roots remains unchanged

SETS

TreeNodes Carrier set for tree nodes

CONSTANTS

K Set of all nodes in forest
K_pr
R Set of all root nodes
R_pr
V Set of all vertices
V_pr
p new node not in K
q existing node in K

AXIOMS

axm1: $K \subseteq TreeNodes$
axm2: $R \subseteq K$
axm3: $V \in K \leftrightarrow K$
axm4: $R = K \setminus ran(V)$
axm5: $p \notin K$
axm6: $q \in K$
axm7: $K\_pr = K \cup \{p\}$
axm8: $V\_pr = V \cup \{q \mapsto p\}$
axm9: $R\_pr = K\_pr \setminus ran(V\_pr)$
thm10: $\langle$theorem$\rangle$ $R = R\_pr$

END

C.10.23 Set Theory 23

CONTEXT SetTheory23

DESCRIPTION

In thm4 a set-theoretic property of any set of integers and family of sets of integers is presented.

axm1 : A is a set of integers, B is a family of sets of integers
axm2 : $PROD$ is declared
axm3 : $PROD$ is defined as the Cartesian product of two sets of integers
thm4 : The Cartesian product of A and the generalised union of B is a subset of the generalised union of the set of Cartesian products of A and elements of B

CONSTANTS

A

B

PROD

AXIOMS

axm1: $A \subseteq \mathbb{Z} \land B \subseteq \mathbb{P}(\mathbb{Z})$

axm2: $PROD \in \mathbb{P}(\mathbb{Z}) \times \mathbb{P}(\mathbb{Z}) \rightarrow \mathbb{P}(\mathbb{Z} \times \mathbb{Z})$

axm3:

$\forall X, Y \cdot X \subseteq \mathbb{Z} \land Y \subseteq \mathbb{Z}$

$\Rightarrow$

$(PROD(X \mapsto Y) = X \times Y)$

thm4: ⟨theorem⟩ $PROD(A \mapsto union(B)) \subseteq union(\{X \cdot X \in B \mid PROD(A \mapsto X)\})$

END

C.10.24 Set Theory 24

Referenced in Section 7.6.3.

CONTEXT SetTheory24

DESCRIPTION

In thm2 a set-theoretic property of any set of integers and family of sets of integers is presented. The theorem appears in Van der Poll (2000, p. 72) as Example 3.11.

axm1 : A is a set of integers and B is a family of sets of integers
thm2 : The Cartesian product of A and the generalised union of B is a subset of the generalised union of the set containing the Cartesian product of A and elements of B

CONSTANTS

A

B

AXIOMS

axm1: $A \subseteq \mathbb{Z} \land B \subseteq \mathbb{P}(\mathbb{Z})$

thm2: ⟨theorem⟩ $A \times union(B) \subseteq union(\{X \cdot X \in B \mid A \times X\})$

END

C.10.25 Set Theory 25

Referenced in Section 7.9.3.

CONTEXT SetTheory25

DESCRIPTION

In thm3 a set-theoretic property of every pair of non-empty families of sets of integers is presented.

axm1 : A and B are families of sets of integers
axm2 : A and B are non-empty

thm3 : The generalised intersection of the union of A and B is a subset of the intersection of the generalised intersection of A and the generalised intersection of B

CONSTANTS

A

B

AXIOMS

axm1: $A \subseteq \mathbb{P}(\mathbb{Z}) \land B \subseteq \mathbb{P}(\mathbb{Z})$

axm2: $A \neq \varnothing \land B \neq \varnothing$

thm3: ⟨theorem⟩ $inter(A \cup B) \subseteq inter(A) \cap inter(B)$

END

C.10.26 Set Theory 26

Referenced in Section 8.3.2.

CONTEXT SetTheory26

DESCRIPTION

In thm2 a set-theoretic property of every non-empty family of sets of integers is presented.

axm1 : A is a non-empty family of sets of integers
thm2 : Then the power set of the generalised intersection of A is equal to the generalised intersection of the set containing the power set of each element of A

CONSTANTS

A

AXIOMS

axm1: $A \subseteq \mathbb{P}(\mathbb{Z}) \land A \neq \varnothing$

thm2: ⟨theorem⟩ $\mathbb{P}(inter(A)) = inter(\{X \cdot X \in A \mid \mathbb{P}(X)\})$

END

C.10.27 Set Theory 27

CONTEXT SetTheory27

DESCRIPTION

Two tautologies that are set-theoretic properties of every non-empty family of sets of integers.

thm1 : Every family of integer sets has the property $inter(\{X\}) = X$
thm2 : Every pair of families of integer sets has the property $inter(\{X, Y\}) = X \cap Y$

AXIOMS

thm1: ⟨theorem⟩ $\forall X \cdot X \subseteq \mathbb{P}(\mathbb{Z}) \Rightarrow inter(\{X\}) = X$

thm2: ⟨theorem⟩ $\forall X, Y \cdot X \subseteq \mathbb{P}(\mathbb{Z}) \land Y \subseteq \mathbb{P}(\mathbb{Z}) \Rightarrow inter(\{X, Y\}) = X \cap Y$

END

Appendix D

Rodin Proof Statuses

In Sections 8.1 and 8.2 of Chapter 8, we discussed the number of automatically discharged proof obligations and compared the discharge rates of the *default auto tactic profile* and the *default auto tactic with SMT* to provide motivation for Rodin Heuristic #1. It is, however, not possible to infer from these discussions the specific proof obligations that were automatically discharged. Tables D.1 and D.2 have therefore been prepared by the researcher to supplement the discussion of tactic profiles in Chapter 8. Every proof obligation that was generated for the Rodin contexts in Appendix C is referenced in these tables.

The proof names and proof statuses are listed by context, and the proof status is one of 'A' or '×': 'A' indicates that the generated proof obligation was automatically discharged and '×' indicates that the proof attempt has failed, i.e. no proof was found.

A side-by-side comparison of Tables D.1 and D.2 will also reveal the proof obligations that could only be discharged by the *default auto tactic with SMT*. These would be the proofs with status '×' in Table D.1, but status 'A' in Table D.2.

Finally, we note that Table D.2 was prepared using a time-out value of 150 milliseconds ($0.15s$) for the SMT provers. This is the time-out value of our preferred *default auto tactic with SMT* — see Section 8.2.2.

D.1 Default Auto Tactic Profile

Table D.1: Rodin Proof Statuses Per Context
Default Auto Tactic Profile
(A - Automatic, × - No Proof)

Context	**Proof → Status**		
FromAbrial01	thm6/THM → ×	thm7/THM → ×	thm8/THM → ×
GroupTheory01	axm8/WD → A	axm9/WD → A	axm10/WD → A
	axm11/WD → A	axm12/WD → A	

Continued on next page

Table D.1 – *Continued* (A - Automatic, × - No Proof)

Context	**Proof → Status**		
GroupTheory02	thm3/WD → ×	thm3/THM → ×	thm4/WD → A
	thm4/THM → ×	thm5/WD → A	thm5/THM → ×
	thm6/WD → A	thm6/THM → A	thm7/WD → A
	thm7/THM → ×	thm8/WD → A	thm8/THM → ×
	thm9/WD → A	thm9/THM → A	
GroupTheory03	axm4/WD → A	thm5/WD → ×	thm5/THM → ×
	thm8/THM → ×	thm9/WD → A	thm9/THM → ×
GroupTheory04	axm3/WD → A	thm5/WD → A	thm5/THM → ×
	thm6/WD → A	thm6/THM → ×	thm7/THM → A
	thm8/WD → A	thm8/THM → ×	thm9/THM → A
	thm10/THM → A	thm11/WD → A	thm11/THM → ×
IntegerProperties01	thm5/THM → A		
IntegerProperties02	thm5/THM → A		
IntegerProperties03	thm5/THM → A		
IntegerProperties04	thm7/THM → A		
IntegerProperties05	thm1/THM → A		
IntegerProperties06	thm1/THM → A		
IntegerProperties07	thm1/THM → A		
IntegerProperties08	thm1/THM → A		
LassoBaseContext	axm2/WD → A		
LassoDerivedContext	thm1/WD → A	thm1/THM → ×	thm2/WD → A
	thm2/THM → ×	thm3/WD → A	thm3/THM → ×
	thm4/WD → A	thm4/THM → ×	thm5/THM → ×
	thm6/WD → A	thm6/THM → ×	
PredicateLogic01	thm2/WD → A	thm2/THM → A	thm4/WD → A
	thm4/THM → A		
PredicateLogic02	axm2/WD → A	axm3/WD → A	thm4/WD → A
	thm4/THM → A		
PropositionalLogic01	thm1/THM → A	thm2/THM → A	thm3/THM → A
	thm4/THM → A	thm5/THM → A	thm6/THM → A
	thm7/THM → A	thm8/THM → A	thm9/THM → A
PropositionalLogic02	thm4/THM → A		
RelationsAndFunctions01	axm3/WD → A		
RelationsAndFunctions02	thm2/WD → A	thm2/THM → ×	thm3/WD → A
	thm3/THM → ×	thm4/WD → A	thm4/THM → A
	thm5/WD → A	thm5/THM → ×	
RelationsAndFunctions03	thm2/WD → A	thm2/THM → ×	thm4/WD → A
	thm4/THM → ×		
RelationsAndFunctions04	axm2/WD → A	thm3/WD → A	thm3/THM → ×
	thm4/WD → A	thm4/THM → ×	thm5/WD → A
	thm5/THM → ×	thm6/WD → A	thm6/THM → ×

Continued on next page

Table D.1 – *Continued* (A - Automatic, × - No Proof)

Context	**Proof → Status**		
	thm7/WD → A thm8/THM → ×	thm7/THM → ×	thm8/WD → A
RelationsAndFunctions05	axm2/WD → A thm10/WD → A thm11/THM → × thm13/WD → A thm14/THM → × thm16/WD → A thm17/THM → × thm19/WD → A thm20/THM → ×	thm9/WD → A thm10/THM → × thm12/WD → A thm13/THM → × thm15/WD → A thm16/THM → × thm18/WD → A thm19/THM → ×	thm9/THM → × thm11/WD → A thm12/THM → × thm14/WD → A thm15/THM → × thm17/WD → A thm18/THM → × thm20/WD → A
RelationsAndFunctions06	axm3/WD → A thm5/WD → A	thm4/WD → A thm5/THM → ×	thm4/THM → ×
RelationsAndFunctions07	axm3/WD → A thm6/WD → A	thm4/WD → A thm6/THM → ×	thm4/THM → ×
RelationsAndFunctions08	axm3/WD → A thm5/WD → A thm7/THM → ×	thm4/WD → A thm5/THM → ×	thm4/THM → × thm7/WD → A
RelationsAndFunctions09	axm2/WD → A thm4/WD → A	thm3/WD → A thm4/THM → ×	thm3/THM → ×
RelationsAndFunctions10	thm2/WD → A thm4/THM → ×	thm2/THM → ×	thm4/WD → A
RelationsAndFunctions11	thm1/WD → A thm2/THM → ×	thm1/THM → ×	thm2/WD → A
RelationsAndFunctions12	thm3/WD → A	thm3/THM → ×	
RelationsAndFunctions13	axm2/WD → A		
RelationsAndFunctions14	thm3/WD → A	thm3/THM → ×	
RelationsAndFunctions15	thm3/WD → A	thm3/THM → ×	
RelationsAndFunctions16	thm2/WD → A thm3/THM → ×	thm2/THM → A thm5/WD → A	thm3/WD → A thm5/THM → ×
RelationsAndFunctions17	thm3/THM → ×	thm5/WD → A	thm5/THM → ×
RelationsAndFunctions18	axm5/WD → A	thm6/WD → A	thm6/THM → A
RelationsAndFunctions19	axm5/WD → A axm8/WD → A	axm6/WD → A axm9/WD → A	axm7/WD → A thm10/THM → ×
RelationsAndFunctions20	axm7/WD → ×	thm8/THM → ×	
RelationsAndFunctions21	axm4/WD → A axm7/WD → A	axm5/WD → A thm8/WD → A	axm6/WD → A thm8/THM → A
RelationsAndFunctions22	axm2/WD → A thm5/WD → A thm6/THM → × thm8/WD → A	thm4/WD → A thm5/THM → × thm7/WD → A thm8/THM → ×	thm4/THM → A thm6/WD → A thm7/THM → × thm9/WD → A

Continued on next page

Table D.1 – *Continued* (A - Automatic, × - No Proof)

Context	**Proof → Status**		
	thm9/THM → ×	thm10/WD → A	thm10/THM → ×
RelationsAndFunctions23	axm3/WD → A	thm4/THM → A	
RelationsAndFunctions24	thm3/THM → A		
RelationsAndFunctions25	thm5/THM → A		
RelationsAndFunctions26	thm6/THM → A		
RelationsAndFunctions27	axm8/WD → A	axm9/WD → A	axm10/WD → A
	axm11/WD → A	axm15/WD → A	axm16/WD → A
	axm17/WD → A		
RelationsAndFunctions28	thm1/WD → A	thm1/THM → ×	thm2/WD → A
	thm2/THM → ×	thm3/WD → A	thm3/THM → ×
	thm4/WD → A	thm4/THM → ×	thm5/WD → A
	thm5/THM → A	thm6/WD → A	thm6/THM → ×
	thm7/WD → A	thm7/THM → ×	thm8/WD → A
	thm8/THM → ×	thm9/WD → A	thm9/THM → A
	thm10/WD → A	thm10/THM → ×	thm11/WD → A
	thm11/THM → ×	thm12/WD → A	thm12/THM → ×
	thm13/WD → A	thm13/THM → ×	thm14/WD → A
	thm14/THM → ×	thm15/WD → A	thm15/THM → ×
	thm16/WD → A	thm16/THM → ×	thm17/WD → A
	thm17/THM → ×	thm18/WD → A	thm18/THM → ×
	thm19/WD → A	thm19/THM → ×	thm20/WD → A
	thm20/THM → ×	thm21/WD → A	thm21/THM → A
	thm22/WD → A	thm22/THM → ×	thm23/WD → A
	thm23/THM → ×	thm24/WD → A	thm24/THM → ×
	thm25/WD → A	thm25/THM → ×	thm26/WD → A
	thm26/THM → ×	thm27/WD → A	thm27/THM → ×
	thm28/WD → A	thm28/THM → ×	thm29/WD → A
	thm29/THM → ×	thm30/WD → A	thm30/THM → ×
SetTheory01	thm1/THM → ×	thm2/THM → A	thm3/THM → ×
SetTheory02	thm1/THM → ×	thm2/THM → ×	
SetTheory03	thm1/WD → A	thm1/THM → ×	
SetTheory04	thm1/THM → A		
SetTheory05	thm1/THM → ×		
SetTheory06	thm1/THM → A		
SetTheory07	thm2/WD → A	thm2/THM → ×	
SetTheory08	thm3/THM → ×		
SetTheory09	thm3/THM → ×		
SetTheory10	thm2/THM → ×		
SetTheory11	thm2/THM → ×		
SetTheory12	thm1/THM → A		
SetTheory13	thm4/THM → ×		

Continued on next page

Table D.1 – *Continued* (A - Automatic, × - No Proof)

Context	**Proof → Status**		
SetTheory14	thm1/THM → A		
SetTheory15	thm2/THM → A	thm3/THM → ×	thm4/THM → ×
SetTheory16	thm4/THM → ×		
SetTheory17	thm1/THM → A	thm2/THM → A	thm3/THM → A
	thm4/THM → A	thm5/THM → A	thm6/THM → A
	thm7/THM → A	thm8/THM → A	thm9/THM → A
	thm10/THM → A	thm11/THM → A	thm12/THM → A
	thm13/THM → A	thm14/THM → A	thm15/THM → A
	axm17/WD → A	thm18/WD → A	thm18/THM → ×
	thm19/WD → A	thm19/THM → ×	thm20/WD → A
	thm20/THM → ×	thm21/WD → A	thm21/THM → ×
	thm22/WD → A	thm22/THM → ×	thm23/WD → A
	thm23/THM → ×	thm24/THM → ×	thm25/THM → ×
	thm26/THM → ×	thm27/THM → ×	thm28/THM → ×
	thm29/THM → A	thm30/THM → A	thm31/THM → ×
	thm32/THM → A	thm33/THM → A	thm34/THM → A
	thm35/THM → A	thm36/THM → A	thm37/THM → ×
SetTheory19	thm1/THM → ×	thm2/THM → A	thm3/THM → ×
	thm4/THM → ×	thm5/THM → ×	thm6/THM → ×
SetTheory20	thm1/THM → ×	thm2/THM → ×	thm3/THM → ×
	thm4/THM → ×		
SetTheory21	thm2/THM → ×	thm3/THM → ×	thm4/THM → ×
	thm5/THM → ×		
SetTheory22	thm10/THM → A		
SetTheory23	axm3/WD → A	thm4/WD → A	thm4/THM → ×
SetTheory24	thm2/THM → ×		
SetTheory25	thm3/WD → A	thm3/THM → A	
SetTheory26	thm2/WD → A	thm2/THM → ×	
SetTheory27	thm1/WD → A	thm1/THM → A	thm2/WD → A
	thm2/THM → A		
TelephoneNetwork	thm13/THM → ×		

D.2 Default Auto Tactic Profile with SMT ($0.15s$ SMT Time-out)

Table D.2: Rodin Proof Statuses Per Context
Default Auto Tactic with SMT
(A - Automatic, × - No Proof)

Context	**Proof → Status**		
FromAbrial01	thm6/THM → ×	thm7/THM → ×	thm8/THM → ×
GroupTheory01	axm8/WD → A	axm9/WD → A	axm10/WD → A
	axm11/WD → A	axm12/WD → A	
GroupTheory02	thm3/WD → A	thm3/THM → A	thm4/WD → A
	thm4/THM → A	thm5/WD → A	thm5/THM → A
	thm6/WD → A	thm6/THM → A	thm7/WD → A
	thm7/THM → A	thm8/WD → A	thm8/THM → A
	thm9/WD → A	thm9/THM → A	
GroupTheory03	axm4/WD → A	thm5/WD → ×	thm5/THM → A
	thm8/THM → ×	thm9/WD → A	thm9/THM → ×
GroupTheory04	axm3/WD → A	thm5/WD → A	thm5/THM → A
	thm6/WD → A	thm6/THM → A	thm7/THM → A
	thm8/WD → A	thm8/THM → ×	thm9/THM → A
	thm10/THM → A	thm11/WD → A	thm11/THM → A
IntegerProperties01	thm5/THM → A		
IntegerProperties02	thm5/THM → A		
IntegerProperties03	thm5/THM → A		
IntegerProperties04	thm7/THM → A		
IntegerProperties05	thm1/THM → A		
IntegerProperties06	thm1/THM → A		
IntegerProperties07	thm1/THM → A		
IntegerProperties08	thm1/THM → A		
LassoBaseContext	axm2/WD → A		
LassoDerivedContext	thm1/WD → A	thm1/THM → A	thm2/WD → A
	thm2/THM → A	thm3/WD → A	thm3/THM → A
	thm4/WD → A	thm4/THM → ×	thm5/THM → ×
	thm6/WD → A	thm6/THM → ×	
PredicateLogic01	thm2/WD → A	thm2/THM → A	thm4/WD → A
	thm4/THM → A		
PredicateLogic02	axm2/WD → A	axm3/WD → A	thm4/WD → A
	thm4/THM → A		
PropositionalLogic01	thm1/THM → A	thm2/THM → A	thm3/THM → A
	thm4/THM → A	thm5/THM → A	thm6/THM → A
	thm7/THM → A	thm8/THM → A	thm9/THM → A
PropositionalLogic02	thm4/THM → A		
RelationsAndFunctions01	axm3/WD → A		

Continued on next page

Table D.2 – *Continued* (A - Automatic, × - No Proof)

Context	**Proof → Status**		
RelationsAndFunctions02	thm2/WD → A thm3/THM → A thm5/WD → A	thm2/THM → × thm4/WD → A thm5/THM → A	thm3/WD → A thm4/THM → A
RelationsAndFunctions03	thm2/WD → A thm4/THM → ×	thm2/THM → ×	thm4/WD → A
RelationsAndFunctions04	axm2/WD → A thm4/WD → A thm5/THM → A thm7/WD → A thm8/THM → ×	thm3/WD → A thm4/THM → A thm6/WD → A thm7/THM → ×	thm3/THM → A thm5/WD → A thm6/THM → × thm8/WD → A
RelationsAndFunctions05	axm2/WD → A thm10/WD → A thm11/THM → A thm13/WD → A thm14/THM → × thm16/WD → A thm17/THM → × thm19/WD → A thm20/THM → ×	thm9/WD → A thm10/THM → A thm12/WD → A thm13/THM → × thm15/WD → A thm16/THM → × thm18/WD → A thm19/THM → ×	thm9/THM → A thm11/WD → A thm12/THM → × thm14/WD → A thm15/THM → × thm17/WD → A thm18/THM → × thm20/WD → A
RelationsAndFunctions06	axm3/WD → A thm5/WD → A	thm4/WD → A thm5/THM → ×	thm4/THM → ×
RelationsAndFunctions07	axm3/WD → A thm6/WD → A	thm4/WD → A thm6/THM → A	thm4/THM → A
RelationsAndFunctions08	axm3/WD → A thm5/WD → A thm7/THM → A	thm4/WD → A thm5/THM → ×	thm4/THM → A thm7/WD → A
RelationsAndFunctions09	axm2/WD → A thm4/WD → A	thm3/WD → A thm4/THM → ×	thm3/THM → A
RelationsAndFunctions10	thm2/WD → A thm4/THM → ×	thm2/THM → ×	thm4/WD → A
RelationsAndFunctions11	thm1/WD → A thm2/THM → ×	thm1/THM → ×	thm2/WD → A
RelationsAndFunctions12	thm3/WD → A	thm3/THM → ×	
RelationsAndFunctions13	axm2/WD → A		
RelationsAndFunctions14	thm3/WD → A	thm3/THM → ×	
RelationsAndFunctions15	thm3/WD → A	thm3/THM → ×	
RelationsAndFunctions16	thm2/WD → A thm3/THM → ×	thm2/THM → A thm5/WD → A	thm3/WD → A thm5/THM → ×
RelationsAndFunctions17	thm3/THM → A	thm5/WD → A	thm5/THM → A
RelationsAndFunctions18	axm5/WD → A	thm6/WD → A	thm6/THM → A
RelationsAndFunctions19	axm5/WD → A	axm6/WD → A	axm7/WD → A

Continued on next page

Table D.2 – *Continued* (A - Automatic, × - No Proof)

Context	**Proof → Status**		
	axm8/WD → A	axm9/WD → A	thm10/THM → ×
RelationsAndFunctions20	axm7/WD → A	thm8/THM → A	
RelationsAndFunctions21	axm4/WD → A	axm5/WD → A	axm6/WD → A
	axm7/WD → A	thm8/WD → A	thm8/THM → A
RelationsAndFunctions22	axm2/WD → A	thm4/WD → A	thm4/THM → A
	thm5/WD → A	thm5/THM → A	thm6/WD → A
	thm6/THM → A	thm7/WD → A	thm7/THM → A
	thm8/WD → A	thm8/THM → A	thm9/WD → A
	thm9/THM → A	thm10/WD → A	thm10/THM → A
RelationsAndFunctions23	axm3/WD → A	thm4/THM → A	
RelationsAndFunctions24	thm3/THM → A		
RelationsAndFunctions25	thm5/THM → A		
RelationsAndFunctions26	thm6/THM → A		
RelationsAndFunctions27	axm8/WD → A	axm9/WD → A	axm10/WD → A
	axm11/WD → A	axm15/WD → A	axm16/WD → A
	axm17/WD → A		
RelationsAndFunctions28	thm1/WD → A	thm1/THM → ×	thm2/WD → A
	thm2/THM → ×	thm3/WD → A	thm3/THM → ×
	thm4/WD → A	thm4/THM → ×	thm5/WD → A
	thm5/THM → A	thm6/WD → A	thm6/THM → A
	thm7/WD → A	thm7/THM → A	thm8/WD → A
	thm8/THM → A	thm9/WD → A	thm9/THM → A
	thm10/WD → A	thm10/THM → A	thm11/WD → A
	thm11/THM → A	thm12/WD → A	thm12/THM → ×
	thm13/WD → A	thm13/THM → ×	thm14/WD → A
	thm14/THM → A	thm15/WD → A	thm15/THM → A
	thm16/WD → A	thm16/THM → ×	thm17/WD → A
	thm17/THM → ×	thm18/WD → A	thm18/THM → A
	thm19/WD → A	thm19/THM → A	thm20/WD → A
	thm20/THM → A	thm21/WD → A	thm21/THM → A
	thm22/WD → A	thm22/THM → A	thm23/WD → A
	thm23/THM → A	thm24/WD → A	thm24/THM → ×
	thm25/WD → A	thm25/THM → ×	thm26/WD → A
	thm26/THM → ×	thm27/WD → A	thm27/THM → ×
	thm28/WD → A	thm28/THM → ×	thm29/WD → A
	thm29/THM → A	thm30/WD → A	thm30/THM → ×
SetTheory01	thm1/THM → ×	thm2/THM → A	thm3/THM → ×
SetTheory02	thm1/THM → A	thm2/THM → A	
SetTheory03	thm1/WD → A	thm1/THM → ×	
SetTheory04	thm1/THM → A		
SetTheory05	thm1/THM → ×		

Continued on next page

Table D.2 – *Continued* (A - Automatic, × - No Proof)

Context	**Proof → Status**		
SetTheory06	thm1/THM → A		
SetTheory07	thm2/WD → A	thm2/THM → A	
SetTheory08	thm3/THM → A		
SetTheory09	thm3/THM → A		
SetTheory10	thm2/THM → A		
SetTheory11	thm2/THM → A		
SetTheory12	thm1/THM → A		
SetTheory13	thm4/THM → A		
SetTheory14	thm1/THM → A		
SetTheory15	thm2/THM → A	thm3/THM → ×	thm4/THM → A
SetTheory16	thm4/THM → A		
SetTheory17	thm1/THM → A	thm2/THM → A	thm3/THM → A
	thm4/THM → A	thm5/THM → A	thm6/THM → A
	thm7/THM → A	thm8/THM → A	thm9/THM → A
	thm10/THM → A	thm11/THM → A	thm12/THM → A
	thm13/THM → A	thm14/THM → A	thm15/THM → A
	axm17/WD → A	thm18/WD → A	thm18/THM → A
	thm19/WD → A	thm19/THM → A	thm20/WD → A
	thm20/THM → A	thm21/WD → A	thm21/THM → A
	thm22/WD → A	thm22/THM → A	thm23/WD → A
	thm23/THM → ×	thm24/THM → A	thm25/THM → A
	thm26/THM → A	thm27/THM → A	thm28/THM → A
	thm29/THM → A	thm30/THM → A	thm31/THM → A
	thm32/THM → A	thm33/THM → A	thm34/THM → A
	thm35/THM → A	thm36/THM → A	thm37/THM → ×
SetTheory19	thm1/THM → ×	thm2/THM → A	thm3/THM → ×
	thm4/THM → ×	thm5/THM → ×	thm6/THM → A
SetTheory20	thm1/THM → A	thm2/THM → ×	thm3/THM → ×
	thm4/THM → ×		
SetTheory21	thm2/THM → A	thm3/THM → ×	thm4/THM → ×
	thm5/THM → ×		
SetTheory22	thm10/THM → A		
SetTheory23	axm3/WD → A	thm4/WD → A	thm4/THM → A
SetTheory24	thm2/THM → ×		
SetTheory25	thm3/WD → A	thm3/THM → A	
SetTheory26	thm2/WD → A	thm2/THM → ×	
SetTheory27	thm1/WD → A	thm1/THM → A	thm2/WD → A
	thm2/THM → A		
TelephoneNetwork	thm13/THM → A		

Bibliography

Abrial, Jean-Raymond (1996). *The B Book: Assigning Programs to Meanings*. Cambridge, England: Cambridge University Press.

Abrial, Jean-Raymond (2005). *The B-book: assigning programs to meanings*. Cambridge University Press.

Abrial, Jean-Raymond (2010). *Modeling in Event-B: system and software engineering*. Cambridge University Press.

Abrial, Jean-Raymond, Michael Butler, Stefan Hallerstede, Thai Son Hoang, et al. (2010). "Rodin: an open toolset for modelling and reasoning in Event-B". In: *STTT* 12.6, pp. 447–466.

Abrial, Jean-Raymond, Michael Butler, Stefan Hallerstede, and Laurent Voisin (2006). "An open extensible tool environment for Event-B". In: *International Conference on Formal Engineering Methods*. Springer, pp. 588–605.

Ballantyne, Michael, Robert S Boyer, and Larry Hines (1996). "Woody Bledsoe: His life and legacy". In: *AI Magazine* 17.1, p. 7.

Barrett, Clark and Sergey Berezin (2004). "CVC Lite: A new implementation of the cooperating validity checker". In: *International Conference on Computer Aided Verification*. Ed. by R. Alur and D.A. Peled. Springer. Springer-Verlag, pp. 515–518.

Barrett, Clark, Christopher L. Conway, et al. (2011). "Cvc4". In: *International Conference on Computer Aided Verification*. Springer, pp. 171–177.

Barrett, Clark, Leonardo De Moura, and Aaron Stump (2005). "SMT-COMP: Satisfiability modulo theories competition". In: *International Conference on Computer Aided Verification*. Springer, pp. 20–23.

Barrett, Clark, Morgan Deters, et al. (2013). "6 Years of SMT-COMP." In: *Journal of Automated Reasoning* 50.3, pp. 243–277.

Barrett, Clark, Pascal Fontaine, and Cesare Tinelli (2015). *The SMT-LIB Standard Version 2.6*. URL: `http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-draft-1.pdf` (visited on 12/01/2016).

Barrett, Clark and Cesare Tinelli (2007). "Cvc3". In: *International Conference on Computer Aided Verification*. Springer, pp. 298–302.

Ben-Ari, Mordechai (2004). *Mathematical logic for Computer Science*. London: Springer–Verlag.

Bledsoe, Woodrow W. (1971). "Splitting and reduction heuristics in automatic theorem proving". In: *Artificial Intelligence* 2.1, pp. 55–77.

Bledsoe, Woodrow W. (1977). "Non-resolution theorem proving". In: *Artificial Intelligence* 9.1, pp. 1–35.

Bledsoe, Woodrow W. (1995). "A precondition prover for analogy". In: *BioSystems* 34.1, pp. 225–247.

Bouton, Thomas et al. (2009). "veriT: an open, trustable and efficient SMT-solver". In: *International Conference on Automated Deduction*. Springer, pp. 151–156.

Bundy, Alan (1999). "A survey of automated deduction". In: *Artificial intelligence today*. Springer, pp. 153–174.

Butler, Michael and Stefan Hallerstede (2007). "The Rodin formal modelling tool". In: *BCS-FACS Christmas 2007 Meeting-Formal Methods In Industry, London.*

Cok, David R., David Déharbe, and Tjark Weber (2016). "The 2014 SMT competition". In: *Journal on Satisfiability, Boolean Modeling and Computation* 9, pp. 207–242.

Cok, David R., Aaron Stump, and Tjark Weber (2015). "The 2013 Evaluation of SMT-COMP and SMT-LIB". In: *Journal of Automated Reasoning* 55.1, pp. 61–90.

Colley, J. (2011a). *ADVANCE Fact Sheet*. URL: http://www.advance-ict.eu/sites/www.advance-ict.eu/files/ADVANCEFactSheet_0.pdf.

Colley, J. (2011b). "Advanced design and verification environment for cyber-physical system engineering - Proposal". In: *ADVANCE Project, Southampton University* 287563.

Colley, J. (2011c). "Advanced design and verification environment for cyber-physical system engineering - Summary". In: *ADVANCE Project, Southampton University* 287563.

CVC4 - The SMT Solver (2017). URL: http://cvc4.cs.stanford.edu/web/ (visited on 04/19/2018).

Davis, Martin, George Logemann, and Donald Loveland (1962). "A machine program for theorem-proving". In: *Communications of the ACM* 5.7, pp. 394–397.

Davis, Martin and Hilary Putnam (1960). "A computing procedure for quantification theory". In: *Journal of the ACM (JACM)* 7.3, pp. 201–215.

De Moura, Leonardo and Nikolaj Bjørner (2008). "Z3: An efficient SMT solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 337–340.

De Moura, Leonardo and Nikolaj Bjørner (2011). "Satisfiability modulo theories: introduction and applications". In: *Communications of the ACM* 54.9, pp. 69–77.

Déharbe, David et al. (2012). "SMT solvers for Rodin". In: *Abstract State Machines, Alloy, B, VDM, and Z*. Springer, pp. 194–207.

DEPLOY (2008). URL: http://www.deploy-project.eu/.

Enderton, Herbert B. (1977). *Elements of set theory*. Academic Press.

Event-B (2016). URL: http://www.event-b.org/ (visited on 12/08/2016).

Ferreirós, José (Dec. 2001). "The road to modern logic - an interpretation". In: *The Bulletin of Symbolic Logic* 7.4, pp. 441–484.

Ferreirós, José (2016). "The Early Development of Set Theory". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2016. Metaphysics Research Lab, Stanford University.

Frege, Gottlob (1879). "Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought". In: *From Frege to Gödel: A source book in mathematical logic* 1931, pp. 1–82.

Gelernter, Herbert (1959). "Realization of a geometry theorem proving machine". In: *IFIP Congress*, pp. 273–281.

Gilmore, Paul C. (1960). "A proof method for quantification theory: its justification and realization". In: *IBM Journal of research and development* 4.1, pp. 28–35.

Guard, James R et al. (1969). "Semi-automated mathematics". In: *Journal of the ACM* 16.1, pp. 49–62.

Hallett, Michael (2016). "Zermelo's Axiomatization of Set Theory". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2016. Metaphysics Research Lab, Stanford University.

Harrison, John, Josef Urban, and Freek Wiedijk (2014). "History of interactive theorem proving". In: *Computational Logic. Handbook of the History of Logic* 9, pp. 135–214.

Hrbacek, Karel and Thomas Jech (1999). *Introduction to set theory*. 3rd ed. New York: Marcel Dekker, Inc.

Hsiang, Jieh et al. (1992). "The term rewriting approach to automated theorem proving". In: *The Journal of Logic Programming* 14.1-2, pp. 71–99.

Hunt, Andrew and David Thomas (2000). *The pragmatic programmer: from journeyman to master*. Addison-Wesley Professional.

Jastram, M., ed. (2012). *Rodin User's Handbook*. English. Version 2.8. Deploy Project.

Jolk, Friederike (2005). "Term Rewriting Systems". In:

Knuth, Donald E. and Peter B. Bendix (1983). "Simple word problems in universal algebras". In: *Automation of Reasoning*. Springer, pp. 342–376.

Kovács, Laura and Andrei Voronkov (2013). "First-order theorem proving and Vampire". In: *International Conference on Computer Aided Verification*. Springer, pp. 1–35.

Krings, Sebastian, Jens Bendisposto, and Michael Leuschel (2015). "From failure to proof: the ProB disprover for B and Event-B". In: *Software Engineering and Formal Methods*. Springer, pp. 199–214.

Labuschagne, W.A. and John Andrew Van der Poll (1999). "Heuristics for resolution-based set-theoretic proofs". In: *South African Computer Journal* 23, pp. 3–17.

Lemor Jr., Luiz Carlos, Simone André Da Costa Cavalheiro, and Luciana Foss (2015). "Proof Tactics for Theorem Proving Graph Grammars through Rodin". In: *Revista de Informática Teórica e Aplicada (RITA)* 22.1, pp. 190–241.

Leuschel, Michael and Michael Butler (2003). "ProB: A model checker for B". In: *International Symposium of Formal Methods Europe*. Springer, pp. 855–874.

Leuschel, Michael and Michael Butler (2008). "ProB: an automated analysis toolset for the B method". In: *International Journal on Software Tools for Technology Transfer* 10.2, pp. 185–203.

Lightfoot, David (1991). *Formal specification using Z*. Macmillan Press.

Lusk, Ewing L. (1992). "Controlling redundancy in large search spaces: Argonne-style theorem proving through the years". In: *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer, pp. 96–106.

Mackenzie, Donald (1995). "The automation of proof: A historical and sociological exploration". In: *IEEE Annals of the History of Computing* 17.3, pp. 7–29.

McCarthy, John (1960). *Programs with common sense*. RLE and MIT Computation Center.

McCune, William (2013). *Prover9*. URL: `http://www.cs.unm.edu/~mccune/prover9/` (visited on 09/24/2013).

McCune, William (1997). "Solution of the Robbins problem". In: *Journal of Automated Reasoning* 19.3, pp. 263–276.

McCune, William (2003). "Otter 3.3 Reference Manual". In: *arXiv preprint cs/0310056*. URL: `https://arxiv.org/abs/cs/0310056`.

McCune, William (2005). "Prover9 is Better Than Otter". In: *Argonne Workshop on Automated Reasoning and Deduction (AWARD)*.

McCune, William and Larry Wos (1997). "Otter: The CADE-13 competition incarnations". In: *Journal of Automated Reasoning* 18.2, pp. 211–220.

Morgan, Carroll (1993). "Telephone Network". In: *Specification case studies*. Ed. by Ian Hayes. 2nd edition. Prentice-Hall International London, pp. 29–38.

Moskewicz, Matthew W. et al. (2001). "Chaff: Engineering an efficient SAT solver". In: *Proceedings of the 38th annual Design Automation Conference*. ACM, pp. 530–535.

Nelson, Greg and Derek C. Oppen (1979). "Simplification by cooperating decision procedures". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 1.2, pp. 245–257.

Newell, Allen and Herbert Simon (1956). "The logic theory machine–A complex information processing system". In: *IRE Transactions on Information Theory* 2.3, pp. 61–79.

Nygaard, M. and E.M. Schmidt (2004). *Algorithms and Data Structures - Transition systems*. Aarhus University - Department of Computer Science. URL: http://cs.au.dk/~gerth/dADS1-09/daimi-fn64.pdf (visited on 01/10/2017).

Pelletier, Francis Jeffry (1986). "Seventy-five problems for testing automatic theorem provers". In: *Journal of automated reasoning* 2.2, pp. 191–216.

Plaisted, David A. (1993). "Equational reasoning and term rewriting systems". In: *Handbook of logic in artificial intelligence and logic programming* 1, pp. 273–364.

Plaisted, David A. (2015). "History and Prospects for First-Order Automated Deduction". In: *International Conference on Automated Deduction*. Springer, pp. 3–28.

Reynolds, Andrew and Viktor Kuncak (2015). "Induction for SMT solvers". In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, pp. 80–98.

Riazanov, Alexandre and Andrei Voronkov (1999). "Vampire". In: *International Conference on Automated Deduction*. Springer, pp. 292–296.

Riazanov, Alexandre and Andrei Voronkov (2001). "Vampire 1.1". In: *International Joint Conference on Automated Reasoning*. Springer, pp. 376–380.

Riazanov, Alexandre and Andrei Voronkov (2002). "The design and implementation of VAMPIRE". In: *AI communications* 15.2, 3, pp. 91–110.

Robinson, George A. and Larry Wos (2000). "Paramodulation and theorem-proving in first-order theories with equality". In: *The Collected Works of Larry Wos: (In 2 Volumes) Volume I: Exploring the Power of Automated Reasoning Volume II: Applying Automated Reasoning to Puzzles, Problems, and Open Questions*. World Scientific, pp. 83–99.

Robinson, John Alan (1963). "Theorem-proving on the computer". In: *Journal of the ACM (JACM)* 10.2, pp. 163–174.

Robinson, John Alan (1965). "A machine-oriented logic based on the resolution principle". In: *Journal of the ACM (JACM)* 12.1, pp. 23–41.

RODIN (2004). *European Project Rodin*. URL: http://rodin.cs.ncl.ac.uk/.

Romanovsky, Alexander and Martyn Thomas (2013). *Industrial deployment of system engineering methods*. Springer.

Russell, Bertrand (1903). *The Principles of Mathematics*. Cambridge University Press.

Siekmann, Jörg and Graham Wrightson, eds. (1983). *Automation of Reasoning: Classical Papers on Computational Logic 1967–1970*. Vol. 2. Berlin: Springer–Verlag.

Spivey, J.M. (1998). *The Z Notation: A Reference Manual*. 2nd ed. Oxford, England: Self published.

Steyn, Paul Stephanus (2009). "Validating reasoning heuristics using next generation theorem provers". MSc dissertation. University of South Africa. URL: http://uir.unisa.ac.za/bitstream/handle/10500/2793/dissertation_steyn_p.pdf (visited on 08/24/2016).

Stump, Aaron, Clark Barrett, and David L. Dill (2002). "CVC: A cooperating validity checker". In: *International Conference on Computer Aided Verification*. Ed. by D. Brinksma and K.G. Larsen. Springer. Springer–Verlag, pp. 500–504.

Sutcliffe, Geoff (2000). "System description: SystemOnTPTP". In: *International Conference on Automated Deduction*. Springer, pp. 406–410.

Sutcliffe, Geoff (2009). "The TPTP problem library and associated infrastructure". In: *Journal of Automated Reasoning* 43.4, pp. 337–362.

Sutcliffe, Geoff (2010). "The TPTP world–infrastructure for automated reasoning". In: *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer, pp. 1–12.

Sutcliffe, Geoff (2016). “The CADE ATP System Competition–CASC”. In: *AI Magazine* 37.2, pp. 99–102.

Sutcliffe, Geoff (2017a). *Proceedings of CASC-26 - the CADE-26 ATP System Competition*. URL: http://www.cs.miami.edu/~tptp/CASC/26/Proceedings.pdf.

Sutcliffe, Geoff (2017b). “The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0”. In: *Journal of Automated Reasoning* 59.4, pp. 483–502.

Sutcliffe, Geoff and C. Suttner (2018). *The TPTP Problem Library for Automated Theorem Proving*. URL: http://www.tptp.org/ (visited on 01/31/2018).

Sutcliffe, Geoff and Josef Urban (2016). “The CADE-25 automated theorem proving system competition–CASC-25”. In: *AI Communications* 29.3, pp. 423–433.

Tammet, Tanel (1997). “Gandalf”. In: *Journal of Automated Reasoning* 18.2, pp. 199–204.

Van der Poll, John Andrew (2000). “Automated support for set-theoretic specifications”. PhD dissertation. University of South Africa.

Van Heijenoort, Jean (1967). *From Frege to Gödel: A source book in mathematical logic, 1879-1931*. Vol. 9. Harvard University Press, pp. 124–128.

Van Lamsweerde, Axel (2000). “Formal specification: a roadmap”. In: *Proceedings of the Conference on the Future of Software Engineering*. ACM, pp. 147–159.

Voronkov, Andrei (2015). *Andrei Voronkov’s Homepage*. URL: http://www.voronkov.com/index.cgi (visited on 04/16/2018).

Wang, Hao (1960). “Proving theorems by pattern recognition I”. In: *Communications of the ACM* 3.4, pp. 220–234.

Wing, Jeannette M. (1990). “A specifier’s introduction to formal methods”. In: *Computer* 23.9, pp. 8–22.

Wos, Larry (2017). *A Summary of Inference Rules Used by Argonne’s Automated Deduction Software*. Argonne National Laboratories. URL: http://www.mcs.anl.gov/research/projects/AR/inf_rules.html#hyper-res (visited on 03/09/2017).

Wos, Larry (1995). “The resonance strategy”. In: *Computers & Mathematics with Applications* 29.2, pp. 133–178.

Wos, Larry (1996). “The power of combining resonance with heat”. In: *Journal of Automated Reasoning* 17.1, pp. 23–81.

Wos, Larry (1998). “Programs that offer fast, flawless, logical reasoning”. In: *Communications of the ACM* 41.6, pp. 87–95.

Wos, Larry (2013). “The legacy of a great researcher”. In: *Automated Reasoning and Mathematics*. Springer, pp. 1–14.

Wos, Larry et al. (1967). “The concept of demodulation in theorem proving”. In: *Journal of the ACM (JACM)* 14.4, pp. 698–709.

Zalta, Edward N. (2016). “Frege’s Theorem and Foundations for Arithmetic”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2016.

Glossary

$\models$ (Semantic consequence) For a finite set of formulas U, $U \models A$ iff $\models A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow A$, where $U = \{A_1, A_2, \ldots, A_n\}$. When U is infinite we have $U \models A$ iff $\models A_1 \wedge A_2 \wedge \cdots \Rightarrow A$, where $U = \{A_1, A_2, \ldots\}$. In other words, if every formula in U is true, then A must be true. Furthermore, $\models A$ denotes that A is a tautology. 23, 28

$\vdash$ (Syntactic consequence) For some given formal system and a finite set of formulas U, $U \vdash A$ iff A is derivable from U in the system. A is a theorem in the system when A can be derived from the empty set of formulas, denoted by $\vdash A$. 25

Argonne National Laboratory The Argonne National Laboratory traces its origins to Enrico Fermi's (recipient of the 1938 Nobel Prize in Physics and founding director of Argonne) secret charge: to create the world's first self-sustaining nuclear reaction. This project is known as the Manhattan Project. In 1942, the "Metallurgical Lab" constructed Chicago Pile-1 (the world's first nuclear reactor) underneath the stands of the University of Chicago's football stadium. Seeing that the experiments posed a serious public risk, operations were moved to a location near Palos Hills and renamed "Argonne" after the surrounding forest.

On the 1st of July 1946, Argonne National Laboratory began (at the request of the U.S. Atomic Energy Commission) developing nuclear reactors for the USA's peaceful nuclear program. For example, the Experimental Breeder Reactor I produced the world's first nuclear-generated electricity in 1951. The designs of most of the commercial reactors currently used throughout the world for electric power generation were influenced and based on the experience and knowledge gained from the Argonne experiments. Today, Argonne National Laboratory is known for numerous initiatives and facilities. Of particular interest to us, is research carried out in the field of automated reasoning. See `https://www.anl.gov/`. 19

Assignment In propositional logic, an assignment is a function $v : \mathcal{P} \to \{T, F\}$, where $\mathcal{P}$ is the set of atomic propositions $\mathcal{P} = \{p, q, r, \ldots\}$. In other words, v assigns one of the truth values T or F to every atom in $\mathcal{P}$. 23, 50, 53

B-method The B-method is a Formal Method originally developed by J.-R. Abrial. The B-method is a theory and methodology for the formal development of computer systems. Event-B (also developed by Abrial) is an evolution of the B-method (Abrial 2005; Leuschel and Butler 2003; *Event-B* 2016). 1, 50, 228

Classical B See **B-method**. 50

Clause A disjunction of literals, often abbreviated to a set containing comma–separated literals. 26

Completeness A deductive system S of first-order logic is *complete* when the following holds: if a formula is logically valid then a finite deduction of the formula exists in S. 14

Conjunctive Normal Form A formula is in conjunctive normal form (CNF) iff it is a conjunction of disjunctions of literals. 40

Formal method "A formal method is a mathematically-based technique used in Computer Science to describe properties of hardware and/or software systems. It provides a framework within which large, complex systems may be specified, developed, and verified in a systematic rather than ad hoc manner. A method is formal if it has a sound mathematical basis, typically given by a formal specification language" (Wing 1990). 55

Formal specification A "*formal specification* is the expression, in some formal language and at some level of abstraction, of a collection of properties some system should satisfy... A specification is *formal* if it is expressed in a language made of three components: rules for determining the grammatical well-formedness of sentences (the syntax); rules for interpreting sentences in a precise, meaningful way within the domain considered (the semantics); and rules for inferring useful information from the specification (the proof theory). The latter component provides the basis for automated analysis of the specification" (Van Lamsweerde 2000). 1

Hybrid theorem proving environment A theorem proving environment is considered a *hybrid* environment when the theorem proving tool applies at least two different inference techniques when discharging proof obligations. 1

Model A satisfying interpretation of a propositional formula A, i.e. $v(A) = T$ for *some* interpretation v of A. 53

Satisfiable A proposition formula A is satisfiable iff $v(A) = T$ for *some* interpretation v. The satisfying interpretation is called a *model* for A. 50

Transition system A transition system consists two sets, namely states and transitions. Transition systems describe the potential behaviour of dynamic processes in terms of states and transitions between states. Transitions specify how to go from state to state without violating any system constraints (Nygaard and Schmidt 2004). 58

Z Z is a formal specification language for describing and modelling computing systems. Z was originally proposed by Jean-Raymond Abrial in 1977 with the help of Steve Schuman and Bertrand Meyer. 1, 78, 123, 127, 133

Abbreviations

AI Artificial Intelligence 17

CADE Conference on Automated Deduction 39, 44

CASC CADE ATP System Competition 38–41, 43, 51

CAV Conference on Computer-Aided Verification 41

CVC Cooperating Validity Checker 51

EU European Union 56

IDE Integrated Development Environment 21, 55, 56, 75

IJCAR International Joint Conference on Automated Reasoning 39

ML Mono-lemma 2, 4, 20, 50, 54, 73–75, 79, 110, 117, 119, 127, 132, 135

nfin Not free in 65–69

PO proof obligation 59, 61–63, 98

POG Proof obligation generator 62

POM Proof obligation manager 63, 70, 71

PP Predicate prover 2–4, 20, 50, 54, 73–75, 82, 111, 119, 127

SAT Satisfiability 50, 53

SC Static checker 62

SMT Satisfiability Modulo Theories iii, 2, 4, 20, 38, 39, 41–43, 50, 51, 53, 54, 58, 75, 127, 130, 135

SMT-COMP Satisfiability Modulo Theories (SMT) Competition 39, 41–43, 51

SMT-LIB Satisfiability Modulo Theories (SMT) Library 39, 41–43, 51

sos Set-of-support 46–48

SPC Specialist Problem Class 41, 135

SVC Standford Validity Checker 51

TPTP Thousands of Problems for Theorem Provers 39–43, 51, 135, 137

UI User interface 63, 72, 154, 156

WD Well-definedness 62

ZFC Zermelo-Fraenkel axioms with Choice 31

Symbols

Index