ANALYZING THE SELECTION OF THE HERBRAND BASE PROCESS FOR BUILDING A SMART SEMANTIC TREE THEOREM PROVER

by

NOURAH A. SHENAIBER

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
October 2015

To my parents, my husband and my beautiful daughters.

Abstract

Traditionally, semantic trees have played an important role in proof theory for validating the unsatisfiability of sets of clauses. More recently, they have also been used to implement more practical tools for verifying the unsatisfiability of clause sets in first-order predicate logic. The method ultimately relies on the Herbrand Base, a set used in building the semantic tree. The Herbrand Base is used together with the Herbrand Universe, which stems from the initial clause set in a particular theorem. When searching for a closed semantic tree, the selection of suitable atoms from the Herbrand Base is very important and should be carried out carefully by educated guesses in order to avoid building a tree using atoms which are irrelevant for the proof. In an effort to circumvent the creation of irrelevant ground instances, a novel approach is investigated in this dissertation. As opposed to creating the ground instances of the clauses in S in a strict syntactic order, the values will be established through calculations which are based on relevance for the problem at hand. This idea has been applied and accordingly tested with the use of the Smart Semantic Tree Theorem Prover (SSTTP), which provides an algorithm for choosing prominent atoms from the Herbrand Base for utilisation in the generation of closed semantic trees. Part of this study is an empirical investigation of this prover performance on first-order problems without equality, as well as whether or not it is able to compete with modern theorem provers in certain niches. The results of the SSTTP are promising in terms of finding proofs in less time than some of the state-of-the-art provers. However, it can not compete with them in terms of the total number of the solved problems.

Acknowledgements

Firstly, I would like to extend my gratitude and heartfelt thanks to my supervisors, Dr Manfred Kerber and Dr Volker Sorge, both of whom have dedicated so much of their time and energy to the supervision of my study. In addition, they have been pivotal in helping me to critically assess my own work and guided me in clearly expressing my findings. I am also grateful to the State of Kuwait and Public Authority for Applied Education and Training, which have provided me with the support and financial resources to facilitate my studies. Lastly, the successes I have experienced are a direct outcome of the love, nurturing and support I have been given by my wonderful family, specifically my parents and my sister Shayma. I dedicate this work and all my love to my husband Abdulaziz Alhajri and to my amazing daughters Sarah and Latifah.

Contents

1 Introduction **1**

2 Related Work **7**
2.1 Herbrand Base Systems . . . 8
2.2 The Model Evolution Calculus . . . 10
2.3 Semantically guided provers . . . 10
2.4 State-of-the-art systems . . . 11
2.4.1 PROVER9 . . . 11
2.4.2 SPASS . . . 13
2.4.3 VAMPIRE . . . 13

3 Fundamentals of theorem proving **15**
3.1 Logic Terminology . . . 15
3.2 Resolution Principle . . . 21
3.2.1 Set-of-support . . . 22

4 Herbrand Theorem **25**
4.1 Herbrand Universe . . . 26
4.2 Herbrand Base . . . 27
4.3 Herbrand Interpretations . . . 27
4.4 Semantic Trees . . . 30
4.5 Herbrand's Theorem . . . 33

5 Smart Semantic Tree Theorem Calculus **35**
5.1 HBG Algorithm . . . 35
5.2 SSTTP Calculus . . . 48
5.3 Variable Instantiation . . . 53
5.3.1 Conditional canonical order grounding . . . 53
5.3.2 Placeholder variable grounding . . . 54

6 Soundness & Completeness **61**
6.1 Soundness proof . . . 61
6.2 Completeness & Fairness . . . 63

7 Implementation, Heuristics, and Experiments **75**
7.1 SSTTP Implementation . . . 76
7.1.1 SSTTP Data Structure . . . 76
7.1.2 SSTTP phases . . . 77
7.2 Heuristics . . . 79
7.2.1 SSTTP-h1: Unit heuristic . . . 79
7.2.2 SSTTP-h2 : Degree order heuristic . . . 86
7.2.3 SSTTP-h3 : Impact heuristic . . . 93
7.2.4 SSTTP-h4 : Meta duplicate elimination heuristic . . . 98
7.3 Heuristics comparison . . . 102

8 Related Systems and Experimental Comparison **105**
8.1 Comparison to the ME calculus . . . 106
8.1.1 ME vs. SSTTP for an Unsatisfiable Problem . . . 108
8.1.2 ME vs. SSTTP for a Satisfiable Problem . . . 110
8.2 Experimental Comparison of SSTTP to State-of-the-Art Provers . . . 113

9 Conclusion and Future Work **121**
9.1 Concluding Remarks . . . 122
9.2 Future Works . . . 123

A SSTTP Output Details from all categories of TPTP library **125**

B Provers Output Details from all categories of TPTP library **147**

List of References **167**

List of Figures

4.1 Closed semantic tree of the problem from example 4.1.1 31

5.5 1^{st} generation of the semantic tree of example 5.3.1 with conditional canonical order grounding. 56
5.6 Closed semantic tree of example 5.3.1 with conditional canonical order grounding after 2^{nd} generation of the *SHB*. 57
5.7 Closed semantic tree of example 5.3.1 with placeholder variable grounding after only 1^{st} generation of the *SHB*. 58

6.1 Closed semantic tree of T_1 and T_2. 66
6.2 Closed semantic tree of T_1' when $l \in \Delta$. 67
6.3 Closed semantic tree of T_1' when $l \notin \Delta$ 67

7.1 Data structure graph for a clause. 77
7.2 Closed semantic tree of the problem from example 7.2.2 applying the unit heuristic h1 which has the advantage that in this example the right subtree can always immediately be closed. Examples in which this is the case have linear complexity. 85
7.3 Closed semantic tree of the problem from example 7.2.3 applying the degree order heuristic h2 which has the advantage that in this example the right subtree can always immediately closed and that gives a linear complexity. The potential of h2 in this example depends on the reordering of the clauses in each node before applying the Build tree rule. h2 shows in this example a better performance than the other heuristics investigated in the dissertation. 92
7.4 Comparison of SSTTP with and without the heuristics in each category of TPTP library. The y-axis represents the number of problems solved. 104

8.1 Comparison of the SSTTP-h4 and three state-of-the-art theorem provers using examples from the TPTP library. The y-axis represents the number of problems solved. 116
8.2 Comparing the SSTTP-h4 with three state-of-the-art provers in case of total solved problems from the TPTP library. 117

List of Tables

4.1 Herbrand interpretations of example 4.1.1 29

5.1 Applying the SSTTP calculus to the clause set of example 5.2.3. 52

7.1 The test results of SSTTP prover, with and without the unit heuristic, in the TPTP library. (T/O means time out and M/O means memory out) . . 81

7.2 Continue Table 7.1, Test results of SSTTP prover with and without the unit heuristic in TPTP library. (T/O means time out and M/O means memory out) 82

7.3 Test results of SSTTP prover with and without the degree order heuristic in TPTP library. (T/O means time out and M/O means memory out) . . 88

7.4 Continue Table 7.3, Test results of SSTTP prover with and without the degree order heuristic in TPTP library. (T/O means time out and M/O means memory out) 89

7.5 Test results of the SSTTP prover with and without the application of the impact heuristic using suitable problems of the TPTP library. 94

7.6 Continuation of Table 7.5: Test results of the SSTTP prover with and without the application of the impact heuristic using suitable problems of the TPTP library. 95

7.7 Test results of SSTTP prover with and without the meta duplicate elimination heuristic in TPTP library. (T/O means time out and M/O means memory out) 100

7.8 Continue Table 7.7, Test results of SSTTP prover with and without the meta duplicate elimination heuristic in TPTP library. (T/O means time out and M/O means memory out) 101

7.9 Analysis of SSTTP output in each category of TPTP library in case of the number of problems solved. 103

8.1 Testing the SSTTP prover on non-theorem problems from the TPTP library. Note that "Model found" means that the prover was able to find out that the input non-theorem was not unsatisfiable, that is, for each of the non-theorems, the theorem prover either could establish that it is satisfiable, or it ran out of time, or out of memory. 106

8.2 Applying the SSTTP calculus to the clause set of example 8.1.3. 112
8.3 Analysis of the SSTTP-h4 and the state-of-the-art provers output from the TPTP library. 115
8.4 Analysis of the SSTTP and the state-of-the-art provers output in each category of the TPTP library. 118
8.5 Continue Table 8.4, Analysis of the SSTTP and the state-of-the-art provers output in each category of the TPTP library. 119

CHAPTER 1

Introduction

The distinction between syntax and semantics is an important tool in the study of language, making the latter an object of investigation itself. While syntax defines a language in terms of its expressions and their grammar, semantics specifies the meanings of these expressions, a distinction that was crucial in the development of the theory of language. But also in the areas of philosophical logic and mathematical reasoning this distinctions allowed for the creation of powerful and rich formalisms that allowed these fields to evolve into the disciplines they are today. But while these formalisms were honed over centuries to simplify and develop computational tasks, when reasoning and proving theorems, mathematicians would typically rely on semantic arguments using examples to illustrate their points. Only in the late 19th and early 20th century serious attempts were made to put mathematical reasoning on a formal logical foundation based on rigorous syntax.

With the advent of computer technology some of the first intelligent systems built were theorem provers that attempted to model the mathematical reasoning process. One of the first attempts at a general system for automated theorem proving was the 1956 Logic Theory Machine of Allen Newell and Herbert Simon [32], a program that sought to establish proofs through symbolic logic by applying chains of possible axioms. However,

whilst the system was successful with a few simple theorems, the searches it had to perform rapidly became too slow. Consequently, a more general system for modelling the mathematical reasoning process was needed, developing rigorous calculi and techniques aiming to reduce the reliance on human-oriented heuristics.

While early techniques were still based to some extent on semantic approaches like exploiting the Herbrand universe, a major breakthrough happened in 1965, when Alan Robinson invented the resolution-refutation principle [39], which significantly improved reasoning by using the purely syntactic tool of unification. In the 1970s, simple versions of the resolution method were incorporated into logic programming languages, of which Prolog is the most prominent example. The big gain in efficiency by these methods meant that human-oriented mathematical theorem proving played only a minor role in the investigation. While there were notable exceptions like the Boyer-Moore theorem prover Nqthm [11], which uses resolution together with methods related to induction in attempting to find proofs of statements (clauses) through the use of a version of Lisp, the majority of work in automated reasoning has been dedicated in an attempt to devise calculi for automated-reasoning systems that can prove theorems by purely syntactic means [38, 46]. At the Argonne National Laboratory, another family of attempts was under development since the early 1960s to find proofs in pure operator (equational) systems (i.e., predicate logic with equations). A very prominent system of this family is OTTER, which was developed in the mid-1980s [29] and later updated and renamed as PROVER9 [27] by McCune, its developer. PROVER9 uses the resolution principle, together with a variety of strategies (such as demodulation, weight and resonance strategies). In the 1990s, at the Max Planck Institute for Computer Science Weidenbach et al. [55] developed SPASS, an automated theorem prover for first-order logic with equality. SPASS was released after experimenting with many prototypes using several data structures, sorted unification, and memory models. Furthermore, Voronkov et al. developed VAMPIRE [36] which is also an

automated theorem prover for first-order logic with equality. Today, all these automated theorem proving systems are still being improved on an annual basis to compete in the CADE ATP System Competitions [48].

While the concentration on purely syntactic reasoning methods has lead to the development of powerful systems, it has also entailed the loss of potential information and to forgo heuristics that allow humans to perform much more advanced reasoning tasks than machines. Consequently, there has been a recent resurgence in attempts to reintroduce semantic techniques into rigorous automated reasoning. Reasoning in our calculus is done by trying to show that it is impossible to construct a model. However, since this is done using a calculus, our system has more in common with the traditional systems mentioned above than with the human-oriented proof planning approach [12, 21].

The goal of this thesis is to contribute to this effort by exploring a semantic oriented calculus that revisits the idea of the Herbrand Universe (HU). In particular, we are interested in building a semantic tree calculus for first-order logic. Semantic trees can be considered as a powerful tool for establishing the unsatisfiability of finite sets of first-order clauses. Building a semantic tree depends on the Herbrand Base (HB), which is generated with the help of the HU; in turn, this stems from the input set of clauses for a given theorem [31]. The elements in the HB can be selected in many different orders for the construction of proofs. A good selection will lead to a shorter proof, whilst a bad selection will lead to a longer proof. Note that, if the selection is not fair, this may mean that no proof can be constructed at all.

The motivation of this work is centred on exploring how to make a smart selection of the atoms from the HB to build a closed semantic tree in an efficient way. This is difficult owing to the fact that the program needs to determine, in an intelligent way, which atoms are useful in the proof construction and which are not. A major novel aspect of this work is to construct the HB in such a way that only elements are added to it that are considered

useful in the proof construction. This refines the work concerned with selecting the proper atoms from the *HB* by providing various heuristic techniques throughout the construction of the tree. The motivation of this work is concerned with establishing a subset of the *HB* for any given problem that can be used to construct a small closed semantic tree. In this thesis, a corresponding theorem prover for building closed semantic trees, called SSTTP (Smart Semantic Tree Theorem Prover), will be presented.

This dissertation follows the idea of incorporating the use of semantics into automated theorem proving by building an efficient smart semantic tree prover. In particular, it makes the following contributions:

1. We adapt the Set-of-Support strategy together with the Herbrand procedure to introduce an algorithm that generates a Smart Herbrand Base (*SHB*) of atoms. The intention is that the algorithm will generate those atoms first that will be useful in building a closed semantic tree. Since we assume that the assumptions for the theorem are consistent it is necessary to use the theorem's clauses in the proof. Just as in the Set-of-Support strategy in resolution theorem proving, our algorithm prefers clauses which either are part of the original theorem clauses or are derived from them.

2. We create a new calculus, called the SSTTP calculus, that is based on the Herbrand Base Generation algorithm (HBG) to build closed semantic trees. The proofs of soundness and completeness of this calculus can be found in Chapter 6.

3. We show two ways to ground atoms inside the *SHB*. The first grounding method is based on the canonical order of the *HU* but with restrictions. The second uses placeholder variables that will substituted during the creation of the closed semantic tree. We will use the second grounding strategy, because – as we will find out empirically in Section 5.3 – it will typically generate semantic tree proofs faster.

4. We give four different heuristics to improve the efficiency in finding proofs. These heuristics try to reorder the atoms inside the *SHB* to prefer the creation and application of useful atoms.

5. We investigate to which degree the SSTTP prover is able to compete with state-of-the-art provers. To this end we make experiments with all suitable problems from the TPTP library. The experimental results can be found in Chapter 8. These results are mixed in that they show that the SSTTP prover can prove fewer theorems only than the other systems, but also that it can prove some of the problems faster than other systems.

The dissertation is organised as follows: Chapter 2 introduces the most important related work. Chapter 3 presents the terminology used in this dissertation, along with an introduction into various useful resolution-refutation principles embedded within SSTTP. Chapter 4 introduces Herbrand's Theorem, the *HU*, the *HB*, as well as Herbrand Interpretations, and further highlights how such concepts are used in building semantic trees. Next, Chapter 5 presents the SSTTP calculus and the structure for effectively creating semantic trees for given sets of clauses by carefully selecting atoms from the *HB*. Following, the soundness and completeness of the calculus are proved in Chapter 6. Subsequently, the heuristics that are implemented in the SSTTP system and their performance are presented in Chapter 7. Finally, we demonstrate in Chapter 8 the effectiveness of this prover by comparing its performance with that of three state-of-the-art theorem provers by testing them on theorems from the TPTP problem collection [47].

CHAPTER 2

Related Work

There is a wide range of approaches to theorem proving, and in the following we will provide a brief overview of systems, which are either most relevant to our work since they are built around semantic methods and the Herbrand theorem, or are state-of-the-art efficient systems with which we will compare our method in Chapter 8. For a general introduction, as well as for a broad and deep exposition of the field, the interested user is pointed to [13, 3, 38]. Provers that build on Herbrand's theorem are HERBY [1, 31], WILLOW [25], and PrHERBY [22]. Also related is the Model Evolution Calculus [6], which is based on the Davis-Putnam-Logemann-Loveland Procedure [49]. The corresponding prover is known as DARWIN [5]. Furthermore, there are various systems that adopt semantic selection techniques so as to guide the search for proofs by resolution. Some of the most efficient existing theorem proving systems are PROVER9 [27], SPASS [55] and VAMPIRE [36], all of which have been regular participants in the CADE competitions for many years. Also, there is the iProver system which is a theorem prover system that uses semantic selection based on the Inst-Gen calculus [23]. The idea of this calculus is to ground clauses of a given set by mapping all variables into a prominent constant to proof the unsatisfiablity of that given set. Moreover, iProver integrates state-of-the-art

implementation techniques such as redundancy elimination, indexing, semantic selection and saturation algorithms. In addition to these techniques, iProver implements the ordered resolution calculus in order to gain simplicity [23]. Furthermore, there is a number of systems that use semantic selection techniques to find proofs not using resolution such as leanTAP [8] and ileanTAP [35]. These systems are tableau-based theorem provers for first-order logic. They use a free variable technique (using rigid variables) to reduce the size of the search space [7].

In the rest of this section, some literature is introduced that is related to our SSTTP system. Section 2.1 discusses some theorem proving systems that are based on the Herbrand Base for solving problems. These systems differ in the way in which they use the heuristics to improve their efficiency. Following, Section 2.2 introduces new research that is based on the DPLL (Davis-Putnam-Logemann-Loveland) Procedure-the so-called ME calculus (Model Evolution Calculus). This new calculus generates a Herbrand Interpretation at the end of the proof for a given problem. We also discuss the relationship between the ME calculus and the SSTTP calculus. Section 2.3 provides a brief introduction on the semantic-based prover. Section 2.4 presents a brief description of the powerful systems that will be compared in Chapter 8 with SSTTP.

2.1 Herbrand Base Systems

HERBY [1, 31] is an automated theorem prover based on the semantic tree developed at McGill University. It implemented in C, and uses Herbrand Base and Universe to obtain a close semantic tree for a given problem. Moreover, it implements various heuristics for choosing the proper atoms from the Herbrand Base during the build of the tree. However, the work done by Almulla and Newborn in building HERBY [1, 31] encountered some difficulties in proving theorems. They said that HERBY needs to develop an atom reordering heuristic and examine how to select atoms for each node in the tree. More-

over, HERBY requires a more sophisticated approach for assigning constants to unground literals in theorems [1, 60].

Another work done by Patrice Lapierre was the building of WILLOW [25], which is an extension of HERBY's semantic tree theorem proving heuristics. WILLOW provides new high-level operations in an effort to simplify the closure of semantic trees. Such operations include the detection of useless and unused atoms, and the reuse of closed subtrees. WILLOW also provides new grounding strategies and an extended set of atoms selection heuristics [25]. WILLOW provides the same grounding method as HERBY [1, 31], which is modulo grounding, along with three other methods. One of its methods is expensive, whereas the other requires a huge amount of computations in building the tree. The third grounding strategy is free variables grounding, which is to replace all local variables in the atom by introducing new free variables. The main problem with the approach is establishing a way of instantiating these variables.

WILLOW [25] provides two ways of instantiation. Unfortunately, however, its approach has a serious drawback: it compromises the completeness of the prover, since failure nodes may not be detected. Lapierre said that the problem could be fixed by rebuilding the subtrees whenever a free variable is instantiated, or otherwise by allowing the duplication of an atom on the same path. But these approaches are not implemented in WILLOW.

In another approach, Newborn and his students started to build programs that work in parallel to prove theorems. One of Newborn's students wrote a paper that discusses a program called PrHERBY [22]. PrHERBY, implemented in C, is a parallel semantic tree theorem prover that combines semantic trees and resolution refutation methods. Moreover, it has a parallel grounding scheme that allows each system to have its own instance of generated atoms. Unfortunately, however, this requires many processors in order to run the program efficiently.

These Herbrand base systems are the systems most related to our SSTTP system. But because their development was discontinued in 2001, it proved exceedingly difficult to get hold of the systems and to make experiments with them.

2.2 The Model Evolution Calculus

The Model Evolution Calculus is a calculus presented by Baumgartner [6]. It is based on the Davis-Putnam-Logemann-Loveland Procedure [49]. Baumgartner presents this calculus as a generalisation of the FDPLL calculus [6, 4]. FDPLL is built from the DPLL procedure by lifting it to the first-order level. The Model Evolution Calculus does not resort to ground instantiations but rather contains a more systematic treatment of universal literals. As a consequence, it has the potential of leading to much faster implementations than FDPLL. The corresponding prover is called DARWIN [3]. The programming language of DARWIN is OCaml. There is some similarity between DARWIN and SSTTP, if we try to show the inconsistency of a consistent set, then both of them may provide a Herbrand model of a given problem. We present examples of this in subsection 8.1.2. The differences between them are based in the choice of the split. SSTTP splits according to the *SHB* [41, 42] and DARWIN splits according to the candidate set that it presents [3]. Section 8.1 describes how ME works, as well as the differences between the ME calculus and the SSTTP calculus.

2.3 Semantically guided provers

SCOTT [43] is an automated theorem prover for first-order logic. It is a variant of OTTER [29] and is based on resolution but with a restriction applied in terms of the inference rules. The programming language of SCOTT is C. SCOTT demands, where one of the parent clauses in each inference step must be false when evaluated in a model. It achieves good results, although the evaluation in a model also requires some degree of

computation. The system has been developed since 1991 and comes in a series of versions, some of them incomplete. Completeness is achieved by introducing several models to select clauses rather than restricting the selection on a single model [19]. Owing to performance issues, a new approach was developed so as to guide selection by soft constraints [45]. This technique provides the system a combination between the speed of a single model and the strength of multiple models. The soft semantic guidance allows the system to solve more problems with equational reasoning as opposed to simple first-order logic.

There are other systems that make attempts to guide clause selection. One of them is the semantic clause graph-prover [14]. This system constructs a clause graph from all the models that are generated from the given theorem. Subsequently, it starts searching for a proof from the links of the graph. The selection requires some heuristics in order to speed-up the performance of the system. A problem with the approach is that it is necessary to generate models from the background theory through the application of a model generator, such as FINDER [44]. Often, this is a time consuming process, particularly if the size of the cardinality is chosen to be bigger than 2 [14].

2.4 State-of-the-art systems

There are many theorem provers that have competed in CASC for several years. Most of them accomplish interesting results in special niches; however, the three systems presented in this section seem to be the most powerful general first-order theorem provers in the field. We provide a brief description of these also as we use them for an evaluation of the strength of SSTTP, as presented in Chapter 8.

2.4.1 PROVER9

PROVER9 [27, 20] is an automated theorem prover for first-order logic that is based on resolution. PROVER9 is a system that was developed from the OTTER prover [29] and

was implemented in C. It uses binary resolution and unit-deletion strategies to find proof. Moreover, it supports reasoning with equality and lists. Its syntax format is user-friendly and easy to use. In order to prove a theorem, PROVER9 seeks to establish a contradiction by negating the goal statement and generating all possible facts from the hypothesis. This strategy is recognised as being quite powerful; occasionally, however, it takes time and may look not as particularly intelligent when compared to the way in which a human mathematician approaches a problem since general purpose first-order theorem provers only very inefficiently deal with arithmetic expressions. A special 'production mode tool' has been developed in PROVER9 to deal with such expressions [40]. The implementation of PROVER9 is similar to OTTER in many ways, especially when considering it uses similar weighting functions that make the decision as to which clauses to select next in the proof search. Normally, PROVER9 cycles through two clause selection functions if there is no semantic guidance. The first one is selecting the oldest clause; the second is selecting the lightest clause. If semantic guidance is used, PROVER9 allows more than one finite interpretation. It evaluates each input or derived clause in its interpretations. The clause is signed as true if it is true in all of the interpretations; if not, it is signed false (except when a parameter showed the evaluation as being expensive, then the clause is signed as true). In other words, PROVER9 cycles through these three clause selection functions if semantic guidance is used: first, the oldest clause is selected; second, the lightest true clause is selected; and third, the lightest false clause is selected. The ratio in which each is of the three functions is chosen is specified by parameters. Examples can be found in [28]. Typically, PROVER9 adopts many strategies to guide the search proof. The most widely implemented strategy is referred to as 'hints', which allows the user to input some clauses to test them with the derived clauses from the search space. The program then gives priority to the matching clauses to continue searching for the result. The only problem associated with strategy is how useful hints can be selected. Ernst [17]

used the data-mining approach on the TPTP library [47] to produce useful hints that can be used to prove more problems.

2.4.2 SPASS

SPASS [55] is an automated theorem prover for first-order logic. It supports equality and various non-classical logics. It is case analysis based on the sorts and splitting rule. The programming language used to implement SPASS is C; this is an easy to use tool and development platform. This prover is regularly updated by new features to adapt with commercial software and to increase the performance. It has a number of modules in its library that deal with different reduction rules and inference selection. In order to run the prover, a set of clauses in a clause normal form is given. Subsequently, the prover attempts to find a proof according to the chosen strategy [55]. When a clause can to be split into two sub-clauses that have at least one positive literal, the SPASS immediately performs the splitting rule. Furthermore, there are different strategies associated with selecting the next splitting clause [56]. In addition to new selection and renaming strategies, SPASS version 3.0 developed a user/machine interface, which handles the formula-clause relationship, the clause set input and the output [58]. The latest enhancements to the prover are sub-term contextual rewriting and improved split backtracking. Furthermore, there are important improvements with the speed of the parser, in an extended sort procedure, with input file commands, and with the TPTP [47] input file syntax [57].

2.4.3 VAMPIRE

VAMPIRE is an automated theorem prover for first-order logic with equality. Initially, the main focus of VAMPIRE is its efficiency. Because of this, VAMPIRE creates huge data structures to index the clauses. Over the years, each version of VAMPIRE achieved higher strength, and more problems in the CADE competition could be solved than before [48]. VAMPIRE is written in C++ and based on two calculi: first, binary resolution with su-

perposition and negative selection; and second, positive and negative hyperresolution but only for logic without equality [36]. Like SPASS [55], this prover implements an OTTER-style saturation algorithm [29] and a DISCOUNT-style algorithm [2, 16]. DISCOUNT is an equational theorem prover that deals with equations derived from the problem. Its idea is based on selecting an unprocessed equation and converting it to normal form, and then using it to produce new unprocessed equations. Subsequently, the unprocessed equation is added to the set of processed equations after it is used for interreduction. This procedure is completed after the proof is found or all equations are processed [15]. In addition, VAMPIRE makes use of an implementation using the so-called limited resource strategy [37]. When the time given for the proof search is limited, this strategy allows the program to create only structures that can be used in order to minimise the necessary resources. VAMPIRE uses splitting without backtracking to avoid wasting time and memory. In VAMPIRE 1.1, many techniques used to split are blocking and parallel splitting, new literals used for splitting and branch rewriting used for splitting [36]. VAMPIRE is not only used as a theorem prover but also as a tool to identify first-order properties automatically. Moreover, it can be used for reasoning with theories and quantifiers because it supports many theory functions on integers, real numbers, strings and arrays. Furthermore, it can analyse many input languages, such as C programs. It has the ability to run several proofs in parallel, if requested by the user [24].

CHAPTER 3

Fundamentals of theorem proving

This chapter presents the basic fundamental concepts in first-order logic and theorem proving that are used in the following and are relevant for the explanation of the SSTTP theorem prover. Section 3.1 defines the logic principles and provides various examples. The next section, Section 3.2, describes the resolution strategies used in the SSTTP prover.

3.1 Logic Terminology

The majority of theorem provers have been devised for first-order logic, with this same group also encompassing the SSTTP prover. In this particular language, a statement is referred to as a well-formed formula (WFF, or sometimes just 'formula') [1, 31, 13]. In this context, the interpretation of a formula is established as devising a statement regarding a particular area of discourse. In an effort to establish formulae syntax, a number of definitions can be provided in relation to literals, logical operators, atoms, quantifiers and terms, as detailed below.

Definition (Logical operators) Logical operators are symbols used to compose more complex formulae from simpler formulae, inductively starting with atomic formulae. In

this dissertation, we define five operators: $\wedge$ (and), | (or), $\sim$ (negation), $\rightarrow$ (if ... then), $\leftrightarrow$ (if and only if).

Definition (Quantifiers) $\forall$ (universal quantifier), $\exists$ (existential quantifier).

Definition (Term) A term is a variable symbol, constant symbol or function symbol with arguments that are also recognised as terms. These are described below.

Definition (Variable) In the specific context of this dissertation, a variable is seen to represent any discourse domain element, and is recognised as being a string of digits, letters or underscores beginning with a letter.

More specifically, a capital letter is used at the beginning of a variable, and is commonly selected from the end of the alphabet,*i.e.* $X, Y, Z, U, V,$ and W. A distinction can be made between constants and variables by considering the context in which they appear (notably, in this dissertation, the first letter is a distinctive factor).

Definition (Constant symbol) A constant symbol is seen to represent a particular aspect of the discourse domain. This is represented through a number of digits, letters or underscores beginning with a letter.

In specific consideration to this dissertation, a constant is initiated with the use of a lower-case letter, which is most commonly selected from the beginning of the alphabet, *i.e.* $a, b, c, ship,$ and $horse$.

Definition (Function symbol) A function symbol represents a function; such as in the case of a map that maps discourse domain elements to other domain elements. This is recognisable through the use of a string of digits, letters or underscores initiated with the use of a letter. Each function symbol is utilised along with a fixed arity. Throughout the course of this research, a function symbol encompasses the same naming criteria as a

constant symbol, whereas a function symbol, on the other hand, comprises one or more arguments, whilst a constant symbol, in contrast, offers zero arguments. Importantly, these arguments are terms.

Some examples of particular function symbols are as follows: $abs(X)$, representing the absolute value function; $minimum(X, Y)$, representing the function mapping X and Y to the minimum of the two.

Definition (Predicate symbol) A predicate symbol stands for a relation on the domain of discourse. This particular relation is either *TRUE* or *FALSE* within the domain, and is represented through the use of a number of digits, letter or underscores, initiated with the use of a letter. A predicate symbol has zero or more arguments. Each predicate symbol goes with a fixed arity and expects as many arguments, which are terms. If P is a predicate symbol and $t_1, \ldots, t_n$ are n terms then $P(t_1, \ldots, t_n)$ is an **atom**. No other expressions are atoms.

Examples of predicates are as follows: $above(a, b)$ (read, "a is above b"), $larger\text{-}or\text{-}equal(square(X), X)$ (read, "The square of X is larger than or equal to X").

Definition (Literal) A literal is an atom or the negation of an atom. If A is an atom, then the two literals A and $\sim A$ are said to be **complements** of each others and the set $\{A, \sim A\}$ is called a complementary pair.

For instance, consider the atom $likes(X, Y)$, then $likes(X, Y)$ and $\sim likes(X, Y)$ are literals.

Definition (Well-formed formula) A well-formed formula **(wff)** is defined recursively as follows:

- An atom is a wff.

- If w and v are wffs, then so are $w|v$, $w \wedge v$, $w \rightarrow v$, and $w \leftrightarrow v$.
- If w is a wff, then, for any variable X, then so are: $\forall X.w$ and $\exists X.w$.

Definition (Clause) A clause is a finite disjunction of zero or more literals. A clause with no literals is called the **empty clause** (denoted by $\square$). A clause with one literal only is called a **unit clause**.

A number of theorem provers adopt a type of formulae that is clause normal in nature as opposed to a particular set of formulae. Notably, [31] provides a conversion algorithm.

Definition (Ground) A clause is ground in the instance that no variables are recognised in any of its literals.

Definition (Interpretation) An interpretation of a formula F in first-order logic comprises a nonempty domain D, and the attribution of 'values' to each of the constant symbols, function symbols and predicate symbols included in F, as shown below:

- An element in D is attributed to each constant symbol.
- A mapping from D^n to D is attributed to each n-ary function symbol.
- A mapping from D^n to $\{TRUE, FALSE\}$ is assigned to each n-ary predicate symbol

Definition (Follows) In the case of a set of wffs, one assigned interpretation makes each wff comprise the logical value of $TRUE$, meaning that particular interpretation fulfils the set of wffs. A wff W is logically **following** from a set Γ of wffs if every interpretation satisfying Γ also satisfies W.

Definition (Satisfiable) A set S of clauses is **satisfiable** if there is a minimum of one interpretation that is seen to fulfil all of the clauses in S. If this does not happen, the set S is **unsatisfiable**.

Definition (Theorem) Let a set of formulae Γ and a formula φ be given. We say that φ is a **theorem** under the assumptions (also called axioms) Γ if the set $\Gamma \cup \{\sim\varphi\}$ is unsatisfiable, that is, if it has no model.

Definition (Substitution) In the discourse domain, a substitution σ is a mapping linking variables to terms. This signifies a finite set of the form $\{t_1/v_1, t_2/v_2, \ldots, t_n/v_n\}$, where every v_i is a variable, every t_i is a term different from v_i and not containing it, and where no two elements are having the same variable following the stroke symbol (/).

Example 3.1.1 *Let us consider the substitution $\{b/Y, f(g(c))/R\}$, here the constant b substitutes for variable Y and the function $f(g(c))$ substitutes for variable R. Please note that upper-case letters are variables, whereas lower-case letters denote functions and constants.*

Definition (Unifier) A substitution σ is referred to as a **unifier** for a set of atoms $\{P_1, \ldots, P_k\}$ if and only if applying σ to P_1 is the same as applying it to P_2 and, in a comparable vein, when applying it to all atoms, *i.e.* $[P_1]\sigma = [P_2]\sigma = \ldots = [P_k]\sigma$ where every P_i is an atom. The set $\{P_1, \ldots, P_k\}$ is recognised as being unifiable if there is an associated unifier.

Definition (Subsumption) Considering that P_1 and P_2 are two atoms, P_1 is subsumes P_2 if there is a substitution σ such that $[P_1]\sigma = P_2$.

Example 3.1.2 *Suppose $P_1 = likes(X, Y)$ and $P_2 = likes(a, b)$. Then, $likes(a, b)$ is subsumed by $likes(X, Y)$ with $\sigma = \{a/X, b/Y\}$.*

Definition (Most general unifier) The most general unifier (mgu), μ, of two atoms instances P_1 and P_2 defined as the unifier that creates a substitution instance P_3 such that P_3 subsumes every other substitution instance of P_1 and P_2; in other words, it creates

the most common instance. When establishing the *mgu*, an algorithm can be used that spans back to Robinson, and is seen in various textbooks, such as [31].

Example 3.1.3 *Suppose* $P_1 = P(X, f(X), f(a))$ *and* $P_2 = P(f(Y), Z, U)$ *are two atoms. Then the mgu of* P_1 *and* P_2 *is* $\mu = \{f(Y)/X, f(f(Y))/Z, f(a)/U\}$*, because* $[P_1]\mu = [P_2]\mu = P(f(Y), f(f(Y)), f(a))$*.*

Definition (Binary resolvent) Consider two clauses $C_1 = l_{11}|l_{12}|\ldots|l_{1n}$ and $C_2 = l_{21}|l_{22}|\ldots|l_{2m}$. Suppose literals l_{1i} and l_{2j} are complements, Then apply resolution rule to get a **binary resolvent** of C_1 and C_2.

$$\frac{C_1, C_2}{[\{C_1 - l_{1i}\}|\{C_2 - l_{2j}\}]\sigma} \text{ If } \sigma = mgu(l_{1i}, l_{2i})$$

Example 3.1.4 *Assume we have two clauses:*

$C_1 : \sim P(a)|Q(a, b)$

$C_2 : \sim Q(X, b)|Q(X, c)$

There is no literal of C_1 *that is complementary to any literal in* C_2*. However, if we substitute a for* X *in* C_2 *then the literals* $Q(a, b)$ *and* $\sim Q(a, b)$ *are complementary literals and they can be resolved away to produce the binary resolvent* C_3*.*

$C_3 : \sim P(a)|Q(a, c)$

Definition (Input set (base set)) The set of base clauses (better known as the base set) are those that make up the axioms and the theorem's negated conclusion following the conversion of wffs into clause form.

Definition (Resultion deduction) Considering a set S of clauses, a resolution deduction of C from S may be recognised as a finite sequence $C_1, C_2, \ldots, C_k$ of clauses, where

each C_i is either a clause from S or a resolvent of preceding clauses, and $C_k = C$. Carrying out the deduction of the empty clause $\square$ from S is referred to as a refutation, or a proof of S.

3.2 Resolution Principle

In an effort to circumvent the creation of ground instance sets, as deemed necessary in Herbrand's process outlined in Chapter 4, there will be some consideration towards resolution of the Robinson principle. This can be recognised as a significant breakthrough owing to the fact it may be directly applied to any particular set S of clauses-notably, not only ground clauses-in an effort to test S unsatisfiability.

The resolution principle has the underlying principle of establishing whether or not S comprises the empty clause $\square$. If S does indeed contain $\square$, S is then considered unsatisfiable; if not, the subsequent stage involves establishing whether $\square$ can be derived from S.

Processes concerned with resolution proof are more effective than any earlier process owing to the fact that an infinitely branching search space is replaced with a finitely branching search space through unification introduction. Following the resolution principle's introduction, a number of changes have been devised and applied in efforts to further increase their efficiency. Some such changes include linear resolution, lock resolution, semantic trees and a set-of-support strategy of Wos [13]. Throughout this dissertation, the resolution principle, set-of-support and semantic trees are combined in a new semantic tree theorem prover, referred to as SSTTP. The set-of-support approach is discussed in the following section.

3.2.1 Set-of-support

Wos, Robinson, and Carson proposed the set-of-support strategy in 1965 [59]. As touched upon in the first chapter, a problem comprises a number of axioms $A_1, A_2, \ldots, A_n$ and a conclusion B. The set-of-support (SoS) strategy involves at least one parent of each resolvent being chosen from amongst the clauses ensuing from the goal negation (negated conclusion) or otherwise from their descendants [34]. In order to ascertain theorem proof, there is a need to establish that $A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \sim B$ is unsatisfiable. If $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ is satisfiable (and typically it is assume that the axioms are satisfiable), where resolving only in this set will not induce the empty clause; therefore, in such an instance, it can be considered a valuable approach to avoid resolving clauses in $A_1 \wedge A_2 \wedge \ldots \wedge A_n$. The set-of-support approach precisely rules out such resolvents.

Definition (*SoS*) A subset T of a set S of clauses is referred to as a set-of-support if $S \backslash T$ is satisfiable. A set-of-support resolution phase is a step involving two clauses where not both are from $S \backslash T$, that is, at least one is from T. A set-of-support deduction may be defined as a deduction where each and every resolution step is a set-of-support resolution step.

Theorem 3.2.1 (Completeness of the *SoS* strategy) *If S is a finite set of unsatisfiable clauses and T is a subset of S such that $S \backslash T$ is satisfiable, then there is a set-of-support deduction of $\square$ [the empty clause] from S with T as set-of-support [13].*

The proof of the completeness of the SoS strategy can be found in Chang textbook page 110 [13].

Example 3.2.2 *Let S be the following set of clauses:*

1. $P(X_1, a, Z_1)$

2. $\sim P(Y_1, f(b), U_1)$

3. $\sim P(X_2, Y_2, U_2) | P(Y_2, Z_2, V) | \sim P(X_2, V, W) | P(U_2, Z_2, W)$

4. $\sim P(k(X_3), X_3, k(X_3))$. $\leftarrow$ *negated conclusion (SoS)*

SoS proof:

5. $\sim P(X_2, Y_2, k(X_3)) | P(Y_2, X_3, V) | \sim P(X_2, V, k(X_3)) \leftarrow$ *a resolvent of literal* $P(U_2, Z_2, W)$ *from 3 and 4*

6. $\sim P(X_2, Y_1, k(f(b))) | \sim P(X_2, U_1, k(f(b))) \leftarrow$ *a resolvent of 2 and literal* $P(Y_2, X_3, V)$ *from 5*

7. $\square \longleftarrow$ *a resolvent of 1 and 6*

CHAPTER 4

Herbrand Theorem

In 1930, Herbrand made a very valuable contribution to mechanical theorem proving. Herbrand was a mathematician of French origin who died young but who provided a number of valuable contribution to mathematical logic. 'Herbrand's Theorem' a well-known theorem, identified a relation between quantification theory and propositional logic [18, 1]. Accordingly, this chapter provides the theoretical basis for establishing theorem proof through creating and devising closed semantic trees. SSTTP, our own program, which creates such trees through a logical and intelligent approach, is discussed in Chapter 5. Moreover, Section 4.1 provides an introduction to the Herbrand universe through a set of clauses, whilst Section 4.2 presents the Herbrand base of the set of clauses. Subsequently, Section 4.3 discusses and defines the Herbrand interpretations of a set of clauses, with semantic trees utilised in mind of establishing the unsatisfiability of sets of clauses detailed in the following two sections. Furthermore, Section 4.5 considers the two different forms of Herbrand's Theorem, as well as their respective application in establishing the unsatisfiability of sets of clauses.

4.1 Herbrand Universe

As has been defined, a set of clauses is unsatisfiable only in the event that it is assigned the *FALSE* value within all domain interpretations. Owing to the fact that it is not possible to take into account all interpretations across all domains, it would be valuable to direct attention to one particular domain H, such that S is unsatisfiable only in the instance that S is *FALSE* within all interpretations over this domain H. Importantly, such a domain does exist, and is referred to as the Herbrand Universe of S, which may be described as discussed in the following paragraph.

The Herbrand Universe of a particular set S of clauses ($HU(S)$ for short) may be defined as a finite set or, otherwise, potentially a countable infinite number of constants and ground terms identified in S. Should S be seen to comprise no function symbols, $HU(S)$ is then recognised as a finite set. Furthermore, should there be no constant encompassed within S, it is then necessary to choose randomly one from the universe of discourse, which then should be a member of the Herbrand Universe.

Definition (Canonical order) The Herbrand Universe is constructed by using a canonical order, where the individual terms are itemised in the following way: The constants come first, with the ground terms listed following, adopting a term depth of 1, followed by those with a term depth of 2, etc. Importantly, ground terms with the same term depth are then listed in line with their arity; those of equal arity are ordered in line with their lexicographical order.

Example 4.1.1 *Let us consider the following problem, in which X is a variable, a is a constant symbol, and f is a function symbol.*

1. $P(X) \longleftarrow$ *negated conclusion (SoS)*
2. $\sim P(a)|Q(X)$

3. $\sim Q(f(X))$

The canonical Herbrand Universe is: $HU = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$

Definition (Ground instance) A ground instance of a clause C is a clause obtained by replacing the variables in C by members of the Herbrand Universe of S.

Example 4.1.2 *Let* $S = \{P(X), Q(f(Y))|R(Y)\}$. *Moreover, let* $C = P(X)$ *be the first clause in* S *and* $HU(S) = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$ *be the Herbrand Universe of* S. *Then the ground instances of* C *are* $P(a)$, $P(f(a))$, $P(f(f(a)))$, *and so on.*

4.2 Herbrand Base

The Herbrand Base of a particular set S of clauses ($HB(S)$ for short) is described as a finite set or, otherwise, potentially a countable infinite number of all ground instances of S clause atoms. The arguments of these atoms are all potential combinations of the terms in Herbrand Universe. When all of the atoms are listed in regard to their arity or according to lexicographical order in the cases of those with equal arity, they are recognised as adopting a canonical order.

Definition (Herbrand Base) Let S be a set of clauses. The set of ground atoms of the form $P^n(t_1, \ldots, t_n)$, for all n-place predicates in S, where $t_1, \ldots, t_n$ are elements of HU, is called the Herbrand Base of S.

Example 4.2.1 *Reconsider the problem from example 4.1.1. Then the canonical ordering of the Herbrand Base is* $HB = \{P(a), Q(a), P(f(a)), Q(f(a)), P(f(f(a))), Q(f(f(a))), \ldots\}$.

4.3 Herbrand Interpretations

SSTTP uses different ways for producing atoms of the Herbrand base. An alternative technique for producing Herbrand base atoms is being suggested in this dissertation, which

is better suited for using the approach to prove theorem directly than listing the Herbrand base atoms in canonical order as described above (or in [31]). This new technique gives rise to the Smart Herbrand Base, which will be presented in Chapter 5.

Definition (Herbrand-interpretation) A Herbrand-interpretation may be defined as the attributing of logical value, either *TRUE* or *FALSE*, to each Herbrand base atom.

Importantly, should the number of Herbrand base atoms be infinite, N, then the value of each of these interpretations is then 2^N. An interpretation does not satisfy (or fails) a clause in the instance that the atoms of various Herbrand Base atom subsets-when assigned *TRUE* or *FALSE* values-are seen to conflict with the clause. Such a conflict may be identified through highlighting that the subset atoms, with their assigned values, resolve with the literals of the clause so as to yield the empty clause. On the other hand, the clause is satisfied by the interpretation [31].

Example 4.3.1 *For the problem in example 4.1.1, any interpretation that assigns TRUE to atom $P(a)$ and FALSE to atom $Q(f(a))$ fails Clause 2 because $P(a)$ resolves with the first literal of Clause 2 (i.e., $\sim P(a)$) to yield the empty clause.*

Definition (Unsatisfiable) A Herbrand interpretation is seen to fail a set of clauses if it is seen to fail at least one of the set's clauses. If this is not the case, the set's clauses are satisfied. A set of clauses is **satisfiable** in the instance that there is at least one Herbrand interpretation that is seen to satisfy each of the clauses. In contrast, the set of clauses is **unsatisfiable**.

One way of identifying the unsatisfiability of the clauses is through the application of a truth table. One alternative approach is through the adoption of semantic trees, as will be discussed in the subsequent section.

Example 4.3.2 *Consider again the problem from example 4.1.1. The canonical Herbrand Base is given as* $\{P(a), Q(a), P(f(a)), Q(f(a)), P(f(f(a))), Q(f(f(a))), \ldots\}$ *in example 4.2.1. In this example, it can be seen that there is an infinite number of Herbrand interpretations, as the following truth table 4.1 shows. Each interpretation is shown to fail at least one clause. The first interpretation fails Clause 3 because the fourth atom* $Q(f(a))$ *resolves with Clause 3 to yield the empty clause. The second interpretation fails Clause 2 because the first and fourth atoms of the interpretation resolve with the literals of the clause* $\sim P(a)|Q(X)$ *to yield the empty clause. The same goes for the rest of interpretation.*

Interpretations on the HB					Unsatisfiable clauses
$P(a)$	$Q(a)$	$P(f(a))$	$Q(f(a))$	...	
T	T	T	T	...	3
T	T	T	F	...	2
T	T	F	T	...	1,3
T	T	F	F	...	2
...	...	...	...	...	
F	F	F	T	...	1,3
F	F	F	F	...	1

Table 4.1: Herbrand interpretations of example 4.1.1

In the Example, all of the interpretations involve the attributing of a *TRUE* or *FALSE* value to an infinite number of atoms; importantly, however, only the first four atoms need to be detailed in canonical order in order to highlight that all of the interpretations are unsuccessful in one clause; in other words, all of the partial interpretations on the first four atoms fails at least one clause of the problem, thus establishing the unsatisfiability of the problem's set of clauses. In actuality, every partial interpretation on just the third and the fourth atoms of the canonical order, namely $P(f(a))$ and $Q(f(a))$, fails at least one clause of the problem, thus suggesting that four partial interpretations are inadequate in terms of identifying the unsatisfiability of the problem's clauses.

4.4 Semantic Trees

Following the discussion of the Herbrand universe and the Herbrand base, semantic trees are now the focus of consideration. It will be seen in the sequel that finding a proof for a set of clauses is equivalent to generating a semantic tree.

Definition (Semantic tree) A semantic tree of a set S of clauses is defined as a binary tree T with root N_0, with branches (edges) labelled by atoms from HB, such that: if N is a node in T, then its two outgoing branches are label with complementary literals hb and $\sim hb$. Let $I(N)$ be the set of literals which are labels along the edges of the path from the root to N. Then for every node N in T, $I(N)$ does not contain complementary literals.

To each node N a set of clauses is assigned, denoted $K(N)$ as follows:

1. To the root node N_0 the given set of clauses S is assigned (i.e. $K(N_0) = S$).

2. For any other node N with the nodes on the path to it labelled $N_0, N_1, \ldots, N_t$ and with the branch leading immediately to it labelled with atom hb or its negation, $\sim hb$, all resolvents of hb or its negation with all clauses in the set $K(N_0), K(N_1), \ldots, K(N_t)$ and with the clauses so generated are assigned to $K(N)$. However, a resolvent is not added to $K(N)$ if it appears already in $K(N_0), K(N_1), \ldots, K(N_t)$ or in $K(N)$.

Definition (failure node) A node N is referred to as a failure node if $I(N)$ makes some ground instance of a clause in S false, but $I(N')$ does not for every ancestor node N' of N.

Definition (Closed semantic tree) A semantic tree is recognised as closed in the instance that every tree branch terminates at a failure node.

Definition (Canonical semantic tree) A canonical semantic tree of depth D is a semantic tree of depth D in which each of the left branches at depth $d \leq D$ is labelled with

the d^{th} atom and where each of the right branches at depth $d \leq D$ is labelled with the complement of the d^{th} atom of the canonical Herbrand Base.

Example 4.4.1 *Let us consider the problem from example 4.1.1. Then from the example we take some atoms from HB which are* $\{P(a), Q(f(a))\}$*, that is, all what we need in building a closed semantic tree. How we can find such a set of atoms we will see in Chapter 5. Figure 4.1 shows a closed semantic tree for the problem from the example 4.1.1.*

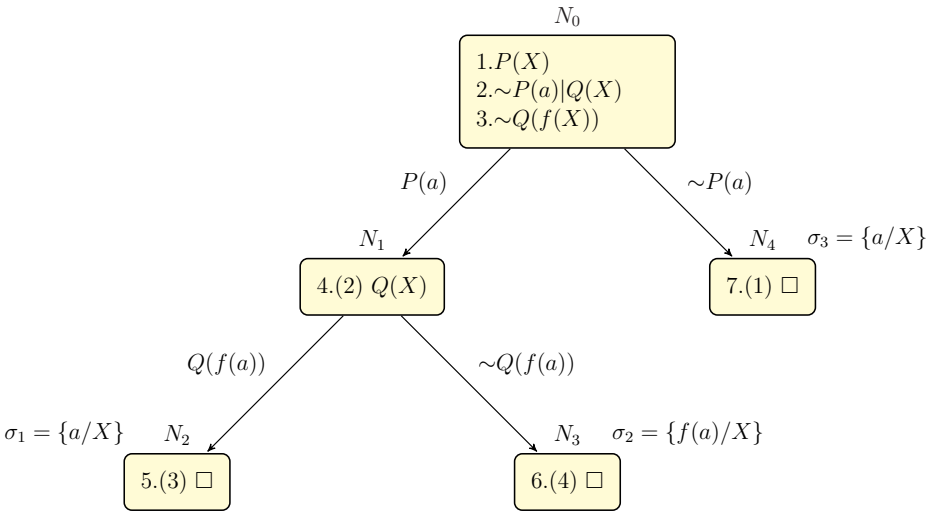

Figure 4.1: Closed semantic tree of the problem from example 4.1.1

The order in which atoms were selected was $hb1 = P(a), hb2 = Q(f(a))$*. First, the root node is assigned the set of clauses S from the problem from example (i.e.* $K(N_0) = S$*) and the depth of the root node is 0. Because the empty clause is not in* $K(N_0)$*, the semantic tree under construction is extended. Then the branch from* N_0 *labelled with* $P(a)$ *is constructed and we arrive at node* N_1*. The set of clauses in* $K(N_1)$ *is found by resolving* $P(a)$ *with the clauses in* $K(N_0)$*:*

$$K(N_1) = \{Q(X)\}$$

In Figure 4.1, the notation describing Clause 4 says that it is formed by resolving the

atom on the branch leading to N_1, that is atom $P(a)$, with the literal in Clause 2 yielding the clause $Q(X)$. Because N_1 is at depth 1 and is not a failure node, D is increased to 2 and the construction continues.

The branch labelled with atom $Q(f(a))$ is constructed, and node N_2 is arrived at next. The set of clauses in $K(N_2)$ is found by resolving $Q(f(a))$ with the clauses in $K(N_0)$ and $K(N_1)$:

$$K(N_2) = \{\square\}$$

In Figure 4.1, Clause 5 is generated by resolving atom $Q(f(a))$ with the literal in Clause 3 yielding the empty clause with $\sigma_1 = \{a/X\}$. Once the empty clause is generated, it is unnecessary to generate other clauses at that node. So, node N_2 is a failure node.

The construction continues by constructing the branch labelled with $\sim Q(f(a))$ and we arrive at node N_3. The set of clauses in $K(N_3)$ is found by resolving $\sim Q(f(a))$ with the clauses in $K(N_0)$ and $K(N_1)$:

$$K(N_3) = \{\square\}$$

In Figure 4.1, Clause 6 is generated by resolving atom $\sim Q(f(a))$ with the literal in Clause 4 yielding the empty clause with $\sigma_2 = \{f(a)/X\}$. Once the empty clause is generated, it is unnecessary to generate other clauses at that node. So, node N_3 is a failure node.

The construction then backtracks to the root and constructs the right branch of N_0, labelling it with $\sim P(a)$ and leading to node N_4, where:

$$K(N_4) = \{\square\}$$

In Figure 4.1, Clause 7 is generated by resolving atom $\sim P(a)$ with the literal in Clause 1 yielding the empty clause with $\sigma_3 = \{a/X\}$. Once the empty clause is generated, it is unnecessary to generate other clauses at that node. So, node N_4 is a failure node, and thus a closed semantic tree is constructed.

4.5 Herbrand's Theorem

In the context of automated theorem proving, Herbrand's Theorem provides a foundation upon which various ATPs (Automated Theorem Provers) rely. In order to establish whether or not a set S of clauses can be considered unsatisfiable, there is a need to take into account only interpretations over the Herbrand universe of S. Should the value be *FALSE* under all interpretations over the Herbrand universe of S, the conclusion can then be drawn to show that S is unsatisfiable. Owing to the fact that here are a countless number of such interpretations, a systematic approach to ordering may be utilised, such as through the application of a semantic tree. Notably, there are two versions of Herbrand's Theorem, as highlighted in the Newborn textbook, [31, p.47-48].

Version I: If S is an unsatisfiable set of clauses, there exists some integer k, such that every partial interpretation over the first k atoms of the canonical Herbrand Base fails S.

Version II: If S is an unsatisfiable set of clauses, there exists some k such that every path in a canonical semantic tree for S beginning at the root and of length at most k leads to a failure node. The semantic tree is said to be closed in this case.

The version most widely cited in the literature is the second one; however, the first of these is recognised as most valuable in the context of this dissertation. Such theorem proof may be identified through reviewing classical textbooks, such as that by Chang and Lee [13, p.61].

Accordingly, this particular theorem may be adopted in different ways, such as by validating the accuracy of the proofs of theorems established through the adoption of resolution-refutation, or otherwise in mind of proving the unsatisfiability of various clause sets, provided there is available a smart approach of choosing atoms from the Herbrand base to be used in mind of creating the closed semantic trees, as will be done throughout the course of this dissertation. The resolution principle of Robinson will be applied,

alongside the set-of-support approach of Wos, in order to effectively create closed semantic trees for unsatisfiable sets of clauses.

CHAPTER 5

Smart Semantic Tree Theorem Calculus

Throughout the course of this research, efforts have been directed towards building a program to prove theorems by constructing a closed semantic tree depending on a Smart Herbrand Base. This Smart Herbrand Base is a subset of the original Herbrand base, which is generated through the application of a smart algorithm. The algorithm is designed to generate the atoms of the Smart Herbrand Base by using a resolution technique. This idea will allow us to use the useful atoms of the Herbrand Base, which will give us an efficient semantic tree theorem proving program (SSTTP). Section 5.1 presents the HBG algorithms with details and examples. Section 5.2 describe the SSTTP calculus and how it works to generate *SHB* atoms and to construct a close semantic tree. Furthermore, in Section 5.3, a brief description is provided of the placeholder variable that the SSTTP program used in order to deal with the variables.

5.1 HBG Algorithm

The algorithm that generates the *SHB* is motivated by the Set-of-Support strategy. We call this algorithm Herbrand Base Generation (HBG). It starts like the original *SoS*, with the input set of clauses $S = \{A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m\}$, where the set

$\{A_1, A_2, \ldots, A_n\}$ is generated from the axiom set and the set $\{B_1, B_2, \ldots, B_m\}$ is generated from the negated conclusion and forms the initial Set-of-Support (SoS). However, it has a different way of using the resolution technique to generate a Smart Herbrand Base in the aim to build closed semantic trees in an effective way. The algorithm uses the resolution rule only to check whether or not the given two literals resolve with each other. Accordingly, it is not actually performing resolution because we do not want to use it for proving; we only use it to help us with the selection of the useful atoms from the Herbrand Base. The steps of the HBG algorithm are shown next:

1. Put the clause(s) of the negated conclusion (denoted by $\{B_1, B_2, \ldots, B_m\}$) in front of the axioms, to build a list $L = (B_1, B_2, \ldots, B_m, A_1, A_2, \ldots, A_n)$.

2. Try to resolve the first literal in the first clause in L denoted by l_1 with all literals in clauses from L occurring after it. If a resolvent occurs then we put the atom of the literal l_1 in the updated SHB after checking that this literal l_1 is a ground atom (i.e., it does not have any variables). If it has variables then we put them in the SHB set after we give them placeholder variables temporarily until they will be substituted by an element from the Herbrand Universe when we use them in the semantic tree.

3. If no resolvent occurs, we then jump to the next literal l_2 in L and repeat step 2. If there is no clause to jump to (i.e., at the end of the list), then exit (stop algorithm).

4. Delete the resolved upon literal l_j and literals that resolved with it from the corresponding clauses in L.

5. Keep repeating the above four steps as long as the input set is not empty.

The outcome of this algorithm is a smart ordering (noncanonical ordering) of atoms in the Herbrand's base (SHB), which, it is hoped, will lead to the faster construction of

closed semantic trees.

Example 5.1.1 *Let us consider the following simple example, let S be the following set of clauses:*

1. $\sim M(f(a)) \longleftarrow$ *negated conclusion (SoS)*
2. $\sim H(X)|M(X)$
3. $\sim H(X)|H(f(X))$
4. $H(a)$

HBG proof (construct the initial SHB):

- *Clause 1 resolves with $M(X)$ from 2*
- *Then put $M(f(a))$ (after substituting X by $f(a)$) in SHB*
- *And delete 1, $M(X)$ from 2*
- *Next, the remaining clauses are:*
  2. $\sim H(X)$
  3. $\sim H(X)|H(f(X))$
  4. $H(a)$
- *Clause 2 resolves with $H(f(X))$ from 3 and with 4*
- *Then put $H(f(H_0))$ (after replacing X with a placeholder H_0) and $H(a)$ (after substituting X by a) in SHB*
- *And delete 2 and $H(f(X)$ from 3 and delete 4*
- *Next, the remaining clauses are*

3. $\sim H(X)$

Stop algorithm. $SHB = \{M(f(a)), H(f(H_0)), H(a)\}$. *The closed semantic tree of this example is shown in Figure 5.1. And during the build of the semantic tree, the placeholder variable* H_0 *will substituted with* a. *Therefore,* $SHB = \{M(f(a)), H(f(a)), H(a)\}$.

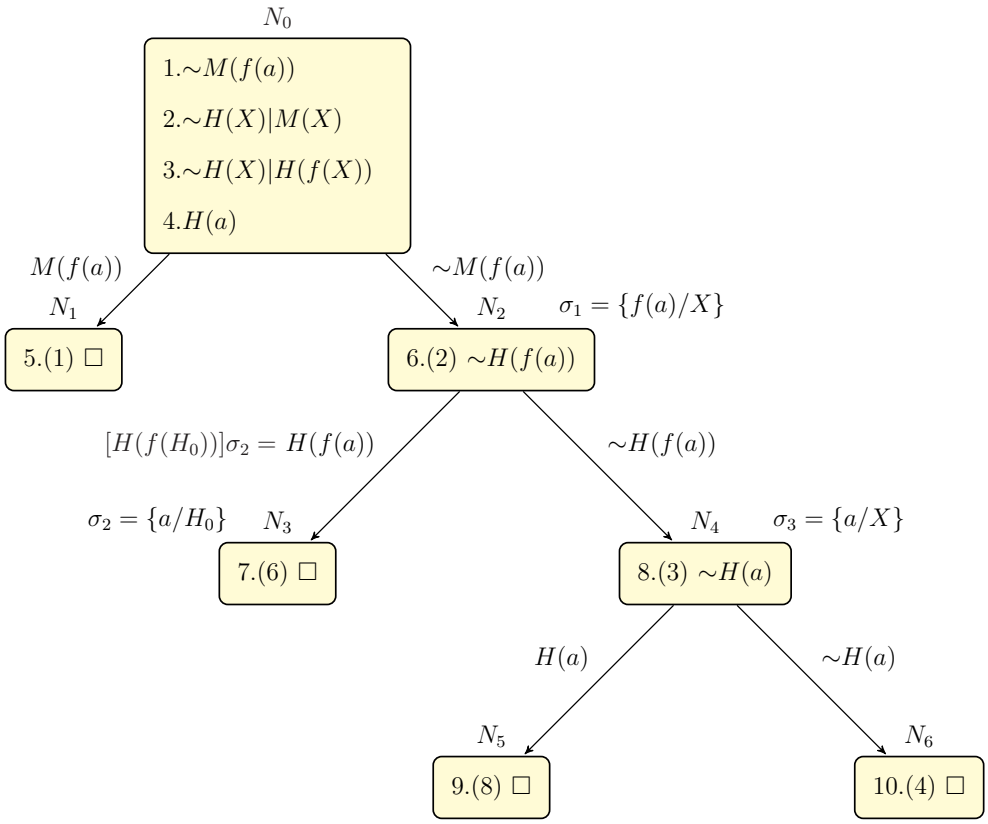

Figure 5.1: Closed semantic tree of the problem from example 5.1.1.

We figure that, in order to make the algorithm complete, the *SHB* needs to be regenerated whenever we reach an open node with an empty *SHB*. Moreover, after we complete the construction of the tree, we will join all the Smart Herbrand Base sets together in one set, which is our smart set that will prove the given theorem. This idea is presented in the following example.

Example 5.1.2 *Let us consider the following simple example. Let S be the following set of clauses:*

1. $P(a) \longleftarrow$ *negated conclusion (SoS)*
2. $\sim P(X) | P(f(X))$
3. $\sim P(f(f(a)))$

HBG proof (construct the initial SHB):

- *Clause 1 resolves with $\sim P(X)$ from 2*
- *Then put $P(a)$ (after substituting X by a) in SHB*
- *And delete 1, $\sim P(X)$ from 2*
- *Next, the remaining clauses are*
  2. $P(f(X))$
  3. $\sim P(f(f(a)))$
- *Clause 2 resolve with clause 3*
- *Then put $P(f(f(a)))$ (after substituting X by $f(a)$) in SHB*
- *And delete 2 and 3*

Stop algorithm. $SHB_1 = \{P(a), P(f(f(a)))\}$. The semantic tree of the first generation of SHB is shown in Figure 5.2.

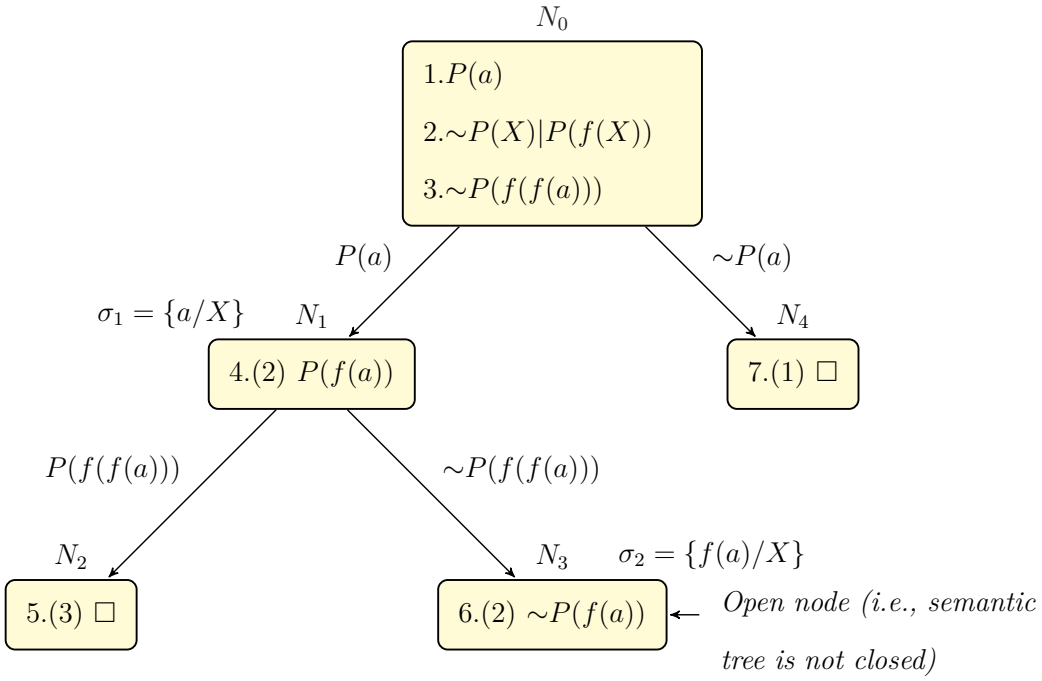

Figure 5.2: Semantic tree of the SHB_1 from example 5.1.2

All elements in the SHB have been used, but no proof has been constructed. So, the HBG algorithm is rerun to generate an extended Smart Herbrand Base.
The algorithm stopped in the open node N_3, let S be the following set of clauses of the path of N_3:

1. $P(a) \longleftarrow$ *negated conclusion (SoS)*
2. $\sim P(X)|P(f(X))$
3. $\sim P(f(f(a)))$
4. $P(f(a))$
6. $\sim P(f(a))$

This set is now used to extend the SHB. HBG proof:

- *Clause 1 resolves with* $\sim P(X)$ *from 2*
- *Then put* $P(a)$ *(after substituting* X *by* a*) in SHB if it is not in the previous one*
- *And delete 1,* $\sim P(X)$ *from 2*
- *Next, the remaining clauses are*

  2. $P(f(X))$

  3. $\sim P(f(f(a)))$

  4. $P(f(a))$

  6. $\sim P(f(a))$

- *Clause 2 resolves with clause 3*
- *Then put* $P(f(f(a)))$ *(after substituting* X *by* $f(a)$*) in SHB if it is not in the previous one*
- *And delete 2 and 3*
- *Next, the remaining clauses are*

  4. $P(f(a))$

  6. $\sim P(f(a))$

- *Clause 4 resolves with clause 6*
- *Then put* $P(f(a))$ *in SHB if it is not in the previous one*

Stop algorithm. The second generation of SHB is $SHB_2 = \{P(f(a))\}$*. Then the algorithms continues building the tree, and forms a closed semantic tree at the end with*

$SHB = SHB_1 \cup SHB_2 = \{P(a), P(f(f(a))), P(f(a))\}$. *The closed semantic tree is shown in Figure 5.3.*

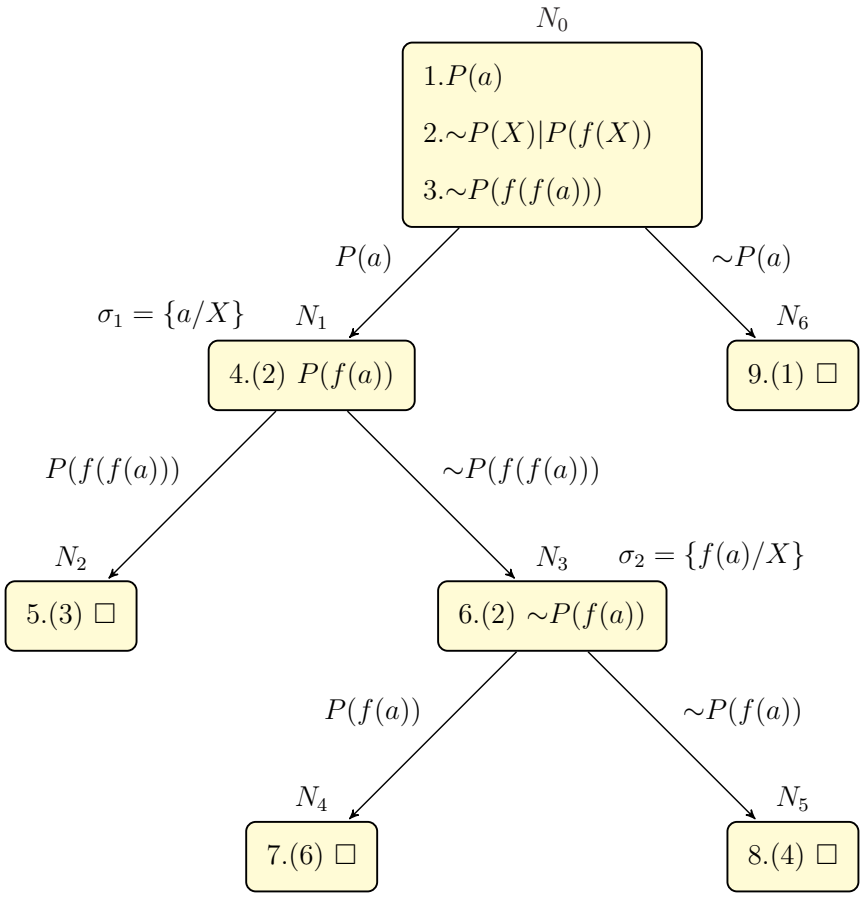

Figure 5.3: Closed semantic tree of example 5.1.2.

The SSTTP prover works by completing multi-resolution steps with the base set of the given theorem, and positions the resolvent atom in the *SHB* as HBG algorithm shows. It then builds the tree with the *SHB* using depth-first search approach. The left-most branch closed first. If this branch cannot be closed, the algorithm has run out of elements from the finite set of *SHB* elements constructed so far. Then, it extends the *SHB* by constructing new elements and continues building the tree until it is closed (or the algorithm runs out

of resources). Subsequently, it backtracks to the previous level and accordingly builds the right branch of the parent node. It continues building the whole tree in a depth-first manner. At the end, if all the leaf nodes of the tree are closed (failed) then it has constructed a proof.

Next, the SSTTP approach is tested with an example selected from the TPTP library.

Example 5.1.3 *Let us consider the following problem from the Analysis domain (ANA013-2.p) in TPTP. Let S be the following set of clauses: (1 and 2 are negated conclusion (SoS))*

1. $\sim P_0(f_3(f_0(c_0, c_1), f_0(f_2(f_1(X_0)), c_1), c_1), f_3(X_0, f_0(f_2(f_1(X_0)), c_1), c_1), c_1)$
2. $P_1(c_1)$
3. $\sim P_2(X_1) | P_0(X_2, X_2, X_1)$
4. $\sim P_1(X_3) | P_2(X_3)$

First, the initial SHB will be constructed from the original set of clauses. Then in a second step a closed semantic tree will be built. We show next the construction of the initial SHB. HBG proof:

- *Clause 1 resolves with* $P_0(X_2, X_2, X_1)$ *from 3*
- *Then put* $P_0(f_3(f_0(c_0, c_1), f_0(f_2(f_1(f_0(c_0, c_1))), c_1), c_1), f_3(f_0(c_0, c_1), f_0(f_2(f_1(f_0(c_0, c_1))), c_1), c_1), c_1)$ *into the SHB after applying the substitution* $\sigma = \{f_3(f_0(c_0, c_1), f_0(f_2(f_1(f_0(c_0, c_1))), c_1), c_1)/X_2, f_0(c_0, c_1)/X_0, c_1/X_1\}$ *to the resolving atom.*
- *And delete 1,* $P_0(X_2, X_2, X_1)$ *from 3*
- *Next, the remaining clauses are*

2. $P_1(c_1)$

3. $\sim P_2(X_1)$

4. $\sim P_1(X_3)|P_2(X_3)$

- *Clause 2 resolve with* $\sim P_1(X_3)$ *from 4*
- *Then put* $P_1(c_1)$ *into the SHB after applying the substitution* $\sigma = \{c_1/X_3\}$ *to resolvent atom.*
- *And delete 2 and* $\sim P_1(X_3)$ *from 4*
- *Next, the remaining clauses are*

  3. $\sim P_2(X_1)$

  4. $P_2(X_3)$

- *Clause 3 resolve with clause 4*
- *Then put* $P_2(X_1)$ *into the SHB after applying the substitution* $\sigma = \{X_1/X_3\}$ *to resolvent atom and because substitution is a variable* X_1 *then we replace it by a placeholder variable* H_0
- *And delete 3 and 4*

Stop algorithm. SHB =
$\{\underbrace{P_0(f_3(f_0(c_0,c_1),f_0(f_2(f_1(f_0(c_0,c_1))),c_1),c_1),f_3(f_0(c_0,c_1),f_0(f_2(f_1(f_0(c_0,c_1))),c_1),c_1),c_1)}_{hb_1},$
$\underbrace{P_1(c_1)}_{hb_2},\underbrace{P_2(H_0)}_{hb_3}\}$. *The closed semantic tree of this example is shown in Figure 5.4.*

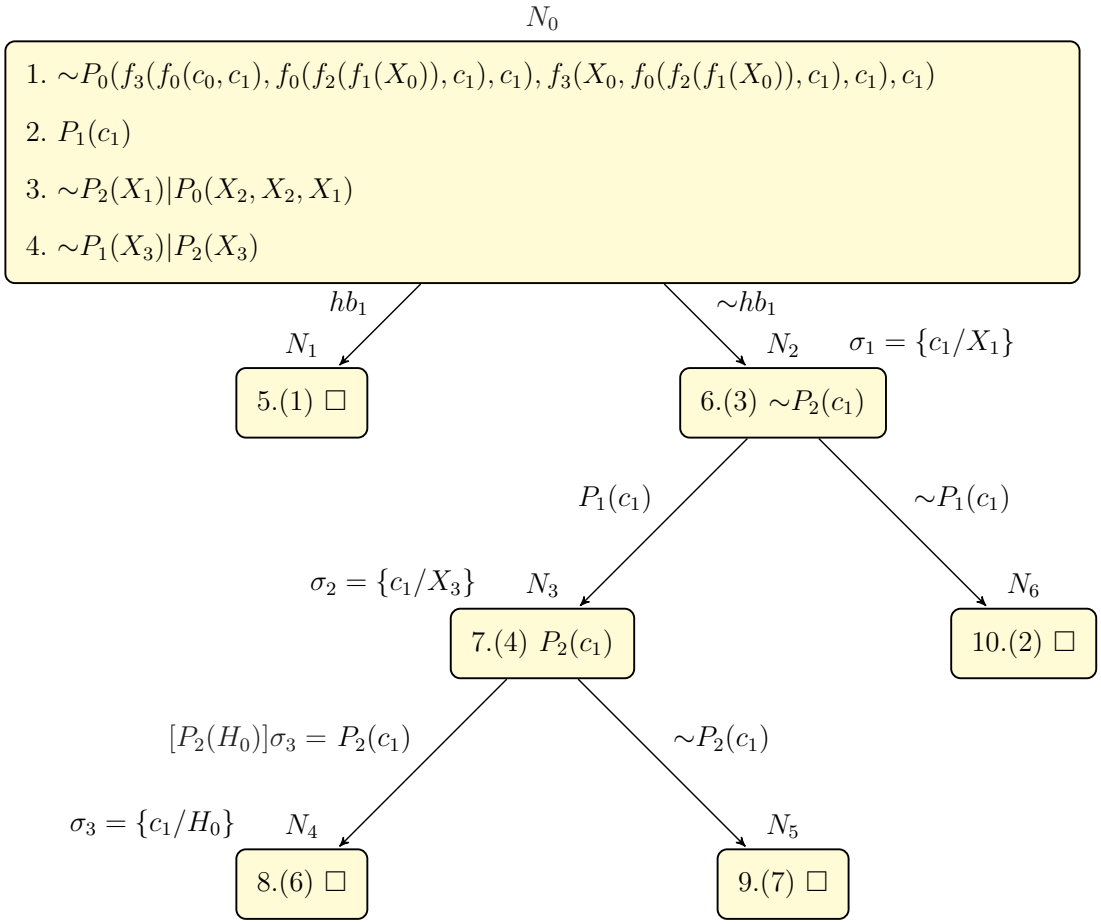

Figure 5.4: Closed semantic tree of the problem (ANA013-2.p) from example 5.1.3

While building of the tree, the placeholder variable H_0 will be substituted by the constant c_1. The following are the steps in the construction of the closed semantic tree: First, the root node is assigned to the set of clauses S from the problem from the example (i.e. $K(N_0) = S$) and the depth of the root node is 0. Because the empty clause is not in $K(N_0)$, the semantic tree under construction is extended. Then the branch from N_0 labeled with hb_1, the first element in SHB, is constructed and we arrive at node N_1. The set of clauses in $K(N_1)$ is found by resolving hb_1 with the clauses in $K(N_0)$:

$$K(N_1) = \{\square\}$$

In Figure 5.4, the notation describing Clause 5 says that it is formed by resolving the atom on the branch leading to N_1, that is atom hb_1, with the literal in Clause 1 yielding the empty clause. Once the empty clause is generated at a node, it is unnecessary to generate other clauses at that node. So, node N_1 is a failure node.

The construction continues by constructing the branch labeled with $\sim hb_1$ and we arrive at node N_2. The set of clauses in $K(N_2)$ is found by resolving $\sim hb_1$ with the clauses in $K(N_0)$:

$$K(N_2) = \{\sim P_2(c_1)\}$$

In Figure 5.4, the notation describing Clause 6 says that it is formed by resolving the atom on the branch leading to N_2, that is atom $\sim hb_1$, with the literal in Clause 3 yielding the clause $\sim P_2(c_1)$ with $\sigma_1 = \{c_1/X_1\}$. Because N_2 is at depth 1 and is not a failure node, D is increased to 2 and the construction continues.

The branch labeled with atom hb_2 is constructed, and node N_3 is arrived at next. The set of clauses in $K(N_3)$ is found by resolving hb_2 with the clauses in $K(N_0)$ and $K(N_2)$:

$$K(N_3) = \{P_2(c_1)\}$$

In Figure 5.4, the notation describing Clause 7 says that it is formed by resolving the atom on the branch leading to N_3, that is atom hb_2, with the literal in Clause 4 yielding the clause $P_2(c_1)$ with $\sigma_2 = \{c_1/X_3\}$. Because N_3 is at depth 2 and is not a failure node, D is increased to 3 and the construction continues.

The branch labeled with atom hb_3 is constructed, and node N_4 is arrived at next. The set of clauses in $K(N_4)$ is found by resolving hb_3 with the clauses in $K(N_0)$, $K(N_2)$, and $K(N_3)$:

$$K(N_4) = \{\square\}$$

In Figure 5.4, the notation describing Clause 8 says that it is formed by resolving the

atom on the branch leading to N_4, that is atom $hb_3 = P_2(H_0)$, with the literal in Clause 6 yielding the empty clause with $\sigma_3 = \{c_1/H_0\}$. And now we replace the placeholder variable H_0 with the constant c_1 globally. So, $hb_3 = P_2(c_1)$. Once the empty clause is generated at a node, it is unnecessary to generate other clauses at that node. So, node N_4 is a failure node.

The construction continues by constructing the branch labeled with $\sim hb_3$ and we arrive at node N_5. The set of clauses in $K(N_5)$ is found by resolving $\sim hb_3$ with the clauses in $K(N_0)$, $K(N_2)$, and $K(N_3)$:

$$K(N_5) = \{\square\}$$

In Figure 5.4, the notation describing Clause 9 says that it is formed by resolving the atom on the branch leading to N_5, that is atom $\sim hb_3 = \sim P_2(c_1)$, with the literal in Clause 7 yielding the empty clause. Because N_5 is a failure node then the construction continues with N_2 and constructs the right branch of N_2, labelling it with $\sim hb_2$ and leading to node N_6, where:

$$K(N_6) = \{\square\}$$

In Figure 5.4, Clause 10 is generated by resolving atom $\sim hb_2$ with the literal in Clause 2 yielding the empty clause. Once the empty clause is generated, it is unnecessary to generate other clauses at that node. So, node N_6 is a failure node, and thus a closed semantic tree is constructed.

5.2 SSTTP Calculus

The SSTTP calculus starts with a clause set Φ, a candidate set and a Smart Herbrand Base set *SHB*. Initially the candidate set is Φ (same as the clause set) and the *SHB* is the empty set (denoted by $\emptyset$). So, the input is of the form [Φ, Φ, $\emptyset$] and, after the derivation rules, we will get the proof of the clause set if all the leaves of the tree are closed; otherwise, the algorithm continues until it runs out of resources (time or space) or no rule is applicable. If there is no proof rule applicable then the *SHB* becomes an actual model of the input clause set.

The calculus is refutationally sound and complete: Φ is a theorem if and only if from [Φ, Φ, $\emptyset$] we can build a tree using the SSTTP calculus rules so that each leaf is closed (that is, of the form [$\Box$, *, *], * stands for an arbitrary set). Chapter 6 will give the soundness and completeness proof of the calculus. In the rules, we represent a clause set Φ as $(\sim l|C, l'|C', \Phi')$, Where $\sim l|C$ and $l'|C'$ are two clauses from Φ and Φ' is the rest of clauses from Φ. Moreover, we denote by $\sim l|C$ a clause D, such that $\sim l$ is a literal of D whilst C is the clause obtained by removing $\sim l$ from D. And s is an atom of the *SHB* set. The SSTTP calculus consists of three rules: HBG, Build tree and Close. All the three rules take three input sets and give three output sets. The first two sets are clause sets and the third one is a set of atoms. In SSTTP, we apply the HBG rule exhaustively. Then, we apply the Build tree rule and the close rule whenever it is possible until we get a proof or *SHB* is empty. If *SHB* is empty, then we apply the HBG rule exhaustively again. So, we continue until a proof is found (that is, a closed semantic tree is constructed). Therefore, the mechanism for the application of rules can be described as follows:

$$(HBG^{+} \circ (\text{Build tree} \mid \text{Close})^{+})^{+}$$

where + means at least one application and $\circ$ means composition.

- HBG:

$$\frac{\Phi \qquad \sim l|C, l'|C', \Phi' \qquad SHB}{\Phi \qquad \sim l|C, C', \Phi' \qquad SHB, [l]\sigma} \quad \text{If} \begin{cases} \sigma = mgu(l, l') \\ [l]\sigma \notin SHB \end{cases}$$

The HBG rule describes the generating procedure of the Smart Herbrand Base (SHB) for a given set of clauses. In this rule, we have three inputs: Φ which is the clause set, $\sim l|C, l'|C', \Phi'$ is the candidate clause set that will be used to generate the Smart Herbrand Base and SHB is the set of the Smart Herbrand Base atoms. Initially, the first and the second inputs are the same (the original clause set) and the SHB is empty. Through this rule, we will use only the second clause set and keep the first clause set unchanged. Then, check the applicability of the resolution rule in the second clause set. If the resolution rule is applicable then we delete the literal l' from the clause C'. After that, the algorithm positions $\sim l|C$ and C' together with Φ in one set and inserts l in the SHB after applying σ to it. The HBG rule is applied repeatedly whenever we need to generate an SHB atom. Example 5.2.1 will exemplify the HBG rule.

- Build tree:

$$\frac{\overbrace{\sim l|C, l'|C', \Phi'}^{\Phi} \qquad \Psi \qquad [s]\sigma, SHB}{\sim l|C, l'|C', \Phi', [C]\sigma \quad \Phi, [C]\sigma \quad SHB \quad | \quad \sim l|C, l'|C', \Phi', [C']\sigma \quad \Phi, [C']\sigma \quad SHB}$$

$$\text{with} \begin{cases} \sigma = mgu(l, l', s) \\ [C]\sigma \notin \Phi \text{ and } [C']\sigma \notin \Phi \end{cases}$$

The Build tree rule describes the derivation process of the semantic tree theorem prover for a given set of clauses. In this rule, we have three inputs: $\Phi = \sim l|C, l'|C', \Phi'$ which is the clause set, Ψ is the candidate clause set and $[s]\sigma, SHB$ is the set of the Smart

Herbrand Base atoms. The candidate set of the premise will not be used in the rule because for the generation of all possible Smart Herbrand Base atoms with the HBG rule it is necessary to use the full clause set Φ, with the new clauses included. Hence, the candidate set of the conclusions (the second component of the conclusions) will be a copy of the clause set (the first component of the conclusions). The Build tree rule works by splitting along the two literals $\sim l$ and l' into two different clauses. Then, resolve $\sim l|C$ with $[s]\sigma$ to get $[C]\sigma$ in the left branch and resolve $l'|C'$ with $[\sim s]\sigma$ to get $[C']\sigma'$ in the right branch. The Build tree rule is applied to construct a semantic tree. Note that, s is always ground or contains placeholder variables which can be instantiated only once. Such an instantiation of a placeholder variable will always happen globally, that is, to the full proof tree. Example 5.2.2 will explain how an instantantiation via $\sigma = mgu(l, l', s)$ is applied.

- Close:

$$\frac{C, \Phi' \qquad \Psi \qquad \text{SHB}}{\Box \qquad \Psi \qquad \text{SHB}} \text{ If } C \text{ is empty clause.}$$

The Close rule describes the closure procedure of the semantic tree theorem prover. If $\Phi = C, \Phi'$ contains the empty clause $\Box$, then this node is a leaf node and it is closed.

Example 5.2.1 *Let* $\Phi = \{\sim P(X_1)|P(f(X_1)), \sim P(a))|\sim P(f(X_2)), P(X_3), P(b)\}$. *Apply the HBG rule to generate the SHB set.*

In the first run of the rule, the SHB $= \{P(H_0)\}$. $P(H_0)$ *comes after resolving* $\sim P(X_1)$ *from the first clause and* $P(X_3)$ *from the third clause with substitution* $\sigma_1 = \{X_1/X_3\}$. *But because the atoms inside the SHB have to be ground, we replace variable* X_1 *by* H_0 *which is a placeholder variable.*

In the second run of the rule, $P(b)$ *comes inside SHB after resolving* $\sim P(X_1)$ *from first clause and* $P(b)$ *from the forth clause with substitution* $\sigma_2 = \{b/X_1\}$.

In the third run of the rule, the SHB $= \{P(H_0), P(b), P(f(H_1))\}$. $P(f(H_1))$ *comes*

after resolving $P(f(X_1))$ from the first clause and $\sim P(f(X_2))$ from the second clause with substitution $\sigma_3 = \{X_1/X_2\}$. But because the atoms inside the SHB have to be ground, we replace variable X_1 by H_1 which is a placeholder variable.

Example 5.2.2 *To explain how $\sigma = mgu(l, l', s)$ is computed in the Build tree rule we consider example 5.2.1. Let $\Phi = \{\sim P(X_1)|P(f(X_1)), \sim P(a))|\sim P(f(X_2)), P(X_3), P(b)\}$ and $s = P(H_0)$ which is the first atom of SHB. Now, apply the Build tree rule.*

In the left hand side, we get a clause $[P(f(H_0))]\sigma$ and a substitution $\{H_0/X_1\}$ after resolving $\sim P(X_1)$ from the first clause and $s = P(H_0)$.

In the right hand side, we get one clause $[\square]\sigma$ and a substitution $\{H_0/X_3\}$ after resolving $P(X_3)$ from the third clause and $s = P(H_0)$.

Thus, $\sigma = mgu(\underbrace{P(X_1)}_{l}, \underbrace{P(X_3)}_{l'}, \underbrace{P(H_0)}_{s}) = \{H_0/X_1, H_0/X_3\}$.

Example 5.2.3 *Consider example 5.1.2. $\Phi = \{P(a), \sim P(X)|P(f(X)), \sim P(f(f(a)))\}$. Table 5.1 shows the steps of applying the SSTTP calculus. Figure 5.3 shows the closed semantic tree of this example. Note that, The SSTTP prover builds the tree using depth-first search approach. To construct this tree according to the SSTTP calculus, we apply the HBG rule two times. So, we get two atoms inside the SHB which are $P(a), P(f(f(a)))$. Then, we apply the Build tree rule twice to generate N_1 then N_2 in the left path as in Figure 5.3, and the close rule to close N_2. In the right, we build N_3. After that, the SHB is empty and the node N_3 is still open. So, we apply the HBG rule exhaustively again with all the clauses in the path of N_3 as an input and a candidate set. Then, we get one atom inside the SHB which is $P(f(a))$. Then, we continue with the build tree rule to get N_4, N_5 and the close rule to close these nodes. Now, we return to the root to build the right node N_6 then apply the close rule to close it. Then we get a proof (closed semantic tree).*

	Clause set	Candidate set	*SHB* set
Initially the *SHB* is empty:			
	Φ	Φ	$\emptyset$
Start with the HBG rule:			
	Φ	$P(f(X)), \sim P(f(f(a)))$	$P(a)$
Apply the HBG rule again:			
	Φ	$\emptyset$	$P(a), P(f(f(a)))$
The HBG rule stops because the candidate clause set is empty.			
Start with the Build tree rule:			
1	Φ	$\emptyset$	$P(a), P(f(f(a)))$
1.1 (left)	$\Phi, P(f(a))$	$\Phi, P(f(a))$	$P(f(f(a)))$
1.1.1 (left)	$\Phi, P(f(a)), \square$	$\Phi, P(f(a)), \square$	$\emptyset$
Now apply the Close rule, because there is an empty clause, to close branch 1.1.1 (left)			
1.1.2 (right)	$\Phi, P(f(a)), \sim P(f(a))$	$\Phi, P(f(a)), \sim P(f(a))$	$\emptyset$
Because the *SHB* is empty and this branch is still open, apply the HBG rule: The first two runs of HBG are redundant. So, the third run will give:			
	$\Phi, P(f(a)), \sim P(f(a))$	$\Phi, P(f(a)), \sim P(f(a))$	$\emptyset$
	$\Phi, P(f(a)), \sim P(f(a))$	$\emptyset$	$P(f(a))$
The HBG rule stops because the candidate clause set is empty.			
Apply the Build tree rule:			
1.1.2.1 (left)	$\Phi, P(f(a)), \sim P(f(a)), \square$	$\Phi, P(f(a)), \sim P(f(a)), \square$	$\emptyset$
Apply the Close rule, because there is an empty clause, to close branch 1.1.2.1 (left)			
1.1.2.2 (right)	$\Phi, P(f(a)), \sim P(f(a)), \square$	$\Phi, P(f(a)), \sim P(f(a)), \square$	$\emptyset$
Apply the Close rule, because there is an empty clause, to close branch 1.1.2.2 (right)			
1.2 (right)	$\Phi, \square$	$\Phi, \square$	$P(f(f(a))), P(f(a))$
Apply the Close rule, because there is an empty clause, to close branch 1.2 (right) The proof is done.			

Table 5.1: Applying the SSTTP calculus to the clause set of example 5.2.3.

5.3 Variable Instantiation

This section presents the methods how SSTTP deals with variables. There are different ways how to do that. In this dissertation, two ways for dealing with variables instantiation are investigated. The first one is the conditional canonical order grounding, which grounds variables according to the Herbrand Universe (HU) order, which is generated applying various conditions. The second method is the placeholder variable grounding, which grounds variables by new free rigid variables.

5.3.1 Conditional canonical order grounding

This method grounds the variables according to the canonical order of the HU but with certain conditions applied. Such conditions depend on the number of the generation of the SHB: for example, the first generation of the SHB picks the first element in the HU; the second generation of the SHB picks the second element in the HU; and so forth. In more detail, let $HU = \{h_1, h_2, h_3, \ldots\}$, and $\Phi = \{P(X_1)|C, \sim P(X_2)|C', \ldots\}$ where C represents the remaining literals of the first clause and C' represents the remaining literals of the second clause. When applying the HBG rule, $[P(X_1)]\sigma$ is added to the SHB. But because the substitution $\sigma = \{X_1/X_2\}$ is not ground, substitute the variable X_1 with the first element in the HU. Therefore, $SHB_1 = \{P(h_1)\}$. Now, start applying the Build tree rule and the Close rule. If the semantic tree is not closed, then apply HBG again to regenerate the SHB. In the second generation of the SHB, substitute the variable X_1 with the second element in the HU. Thus, $SHB_2 = \{P(h_2)\}$ and the building of the semantic tree can continue. If it is not closed, apply HBG again and substitute X_1 with h_3 and so on.

5.3.2 Placeholder variable grounding

This method grounds the variables with new free variables. It is like variable renaming the variables with new names so that it will not conflict with other variables in the tree. Free variables are not local to a given clause, but rather are global to all clauses in the semantic tree. These variables are grounded whenever a unification occurs with one side of a constant or a function from the HU set during the process of building the tree. When one of the free variables is ground, all its occurrence are grounded in the tree with the same substitution. After constructing a closed semantic tree, if there are still free variables in the tree, they will be grounded with an arbitrary constant. This is implemented in the SSTTP program and tested with problems from the TPTP problem library. In more detail, let $\Phi = \{P(X_1)|C, \sim P(X_2)|C', \ldots\}$ where C represents the remaining literals of the first clause and C' represents the remaining literals of the second clause. When applying the HBG rule, $[P(X_1)]\sigma$ is inserted in to the SHB. But because the substitution $\sigma = \{X_1/X_2\}$ is not ground, substitute the variable X_1 with H_1 first then start applying the Build tree rule and the Close rule. This H_1 is a placeholder variable acting as a constant during the building of the semantic tree and will be substituted only when a substitution with a ground term occurs. For example, if $\sigma = \{a/H_1\}$ occurs during the building of the semantic tree then substitute all the occurrence of H_1 inside the semantic tree by a.

The next example 5.3.1 describes the difference between the two grounding methods.

Example 5.3.1 *Let S be the following set of clauses:*

1. $\sim Q(f(a))$
2. $\sim R(f(b))$
3. $\sim P(X_1)|R(f(X_1))$

4. $P(X_2)|Q(X_3)$

Compute the HU of the example:

$HU(S) = \{a, b, f(a), f(b), f(f(a)), f(f(b)), \ldots\}$.

The Smart Herbrand Base that is generated from the HBG rule:

$SHB = \{Q(f(a)), R(f(b)), [P(X_1)]\sigma\}$.

Now, we solve the example according to the two methods:

1. *Conditional canonical order grounding:*

   This grounding method will substitute variables before inserting atoms inside the SHB according to the canonical order of the HU but with one condition. This condition depends on the generation number of the SHB. If the program is in the first generation of the SHB, we then take the first element in the HU which is a and substitute X with a in the SHB. If the program is in the second generation of the SHB, we then take the second element in the HU which is b and substitute X with b in the SHB and so on.

   In example 5.3.1, we start with the first generation of the SHB:

   The substitution $\sigma_i = \{a/X\}$, and SHB $= \{Q(f(a)), R(f(b)), [P(X_1)]\sigma_i\}$

   $= \{Q(f(a)), R(f(b)), P(a)\}$.

   The result of the first generation give us a non-closed semantic tree, as shown in Figure 5.5. Accordingly, we start the second generation of the SHB to get a closed semantic tree, as shown in Figure 5.6:

   The substitution $\sigma_j = \{b/X\}$, and SHB $= \{Q(f(a)), R(f(b)), P(a), [P(X_1)]\sigma_j\}$

   $= \{Q(f(a)), R(f(b)), P(a), P(b)\}$

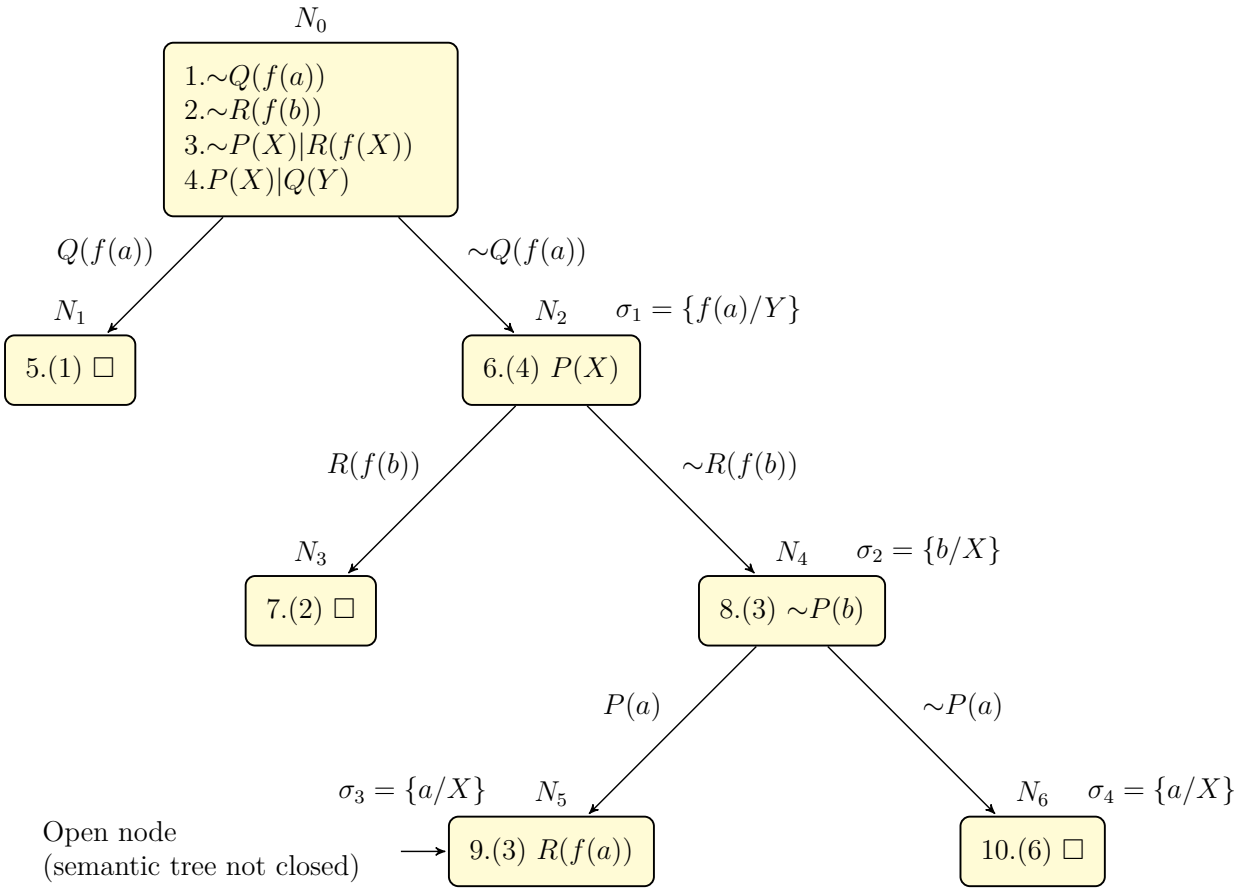

Figure 5.5: 1^{st} generation of the semantic tree of example 5.3.1 with conditional canonical order grounding.

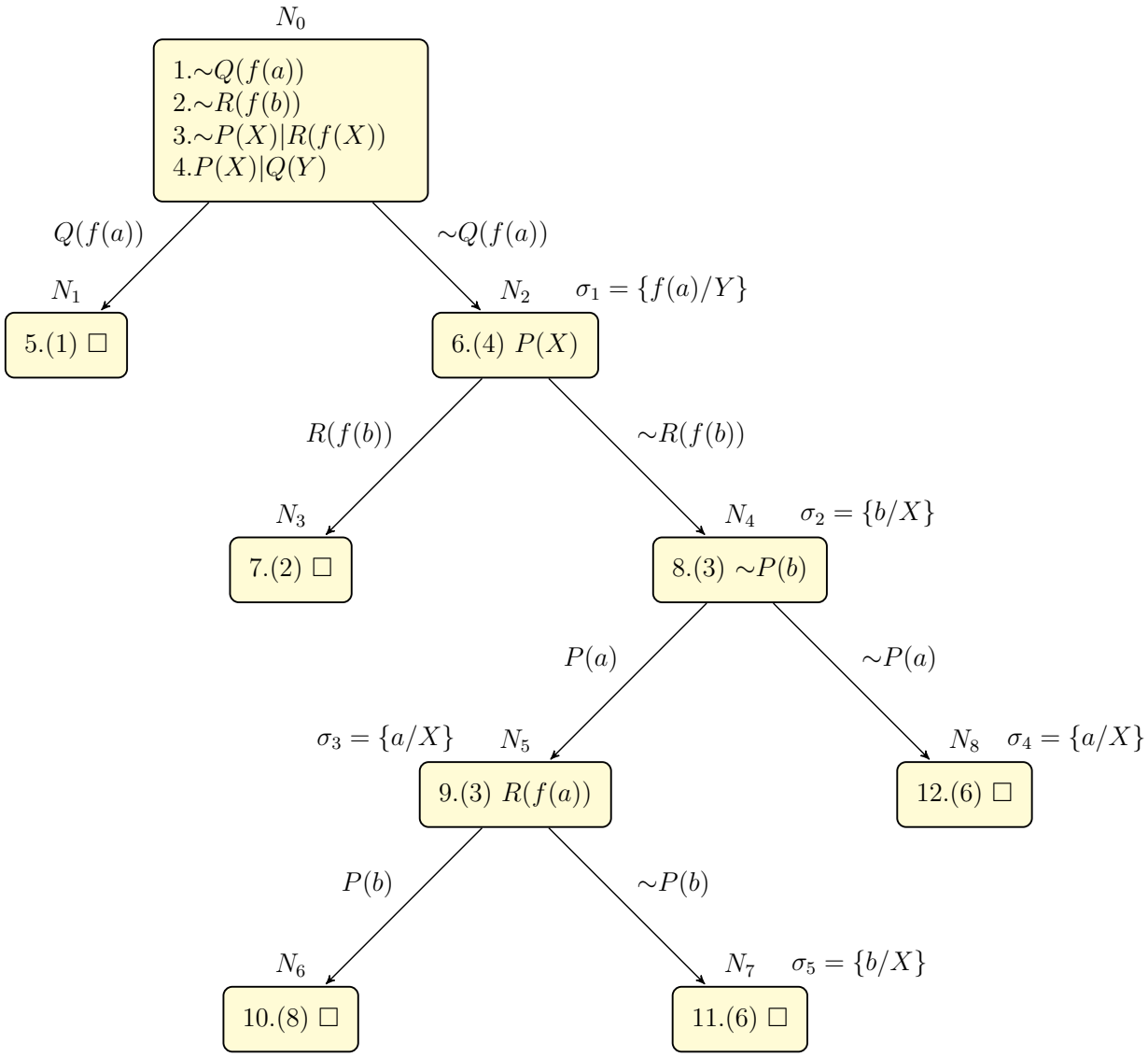

Figure 5.6: Closed semantic tree of example 5.3.1 with conditional canonical order grounding after 2^{nd} generation of the *SHB*.

2. *Placeholder variable grounding:*

   This grounding method will substitute variables before inserting atoms in to the SHB by placeholder variables. Subsequently, they will be substituted according to the unification inside the Build tree rule.

   In example 5.3.1, we start with the first generation of the SHB:

   $SHB = \{Q(f(a)), R(f(b)), P(H_0)\}$.

   We then get a closed semantic tree, as shown in Figure 5.7.

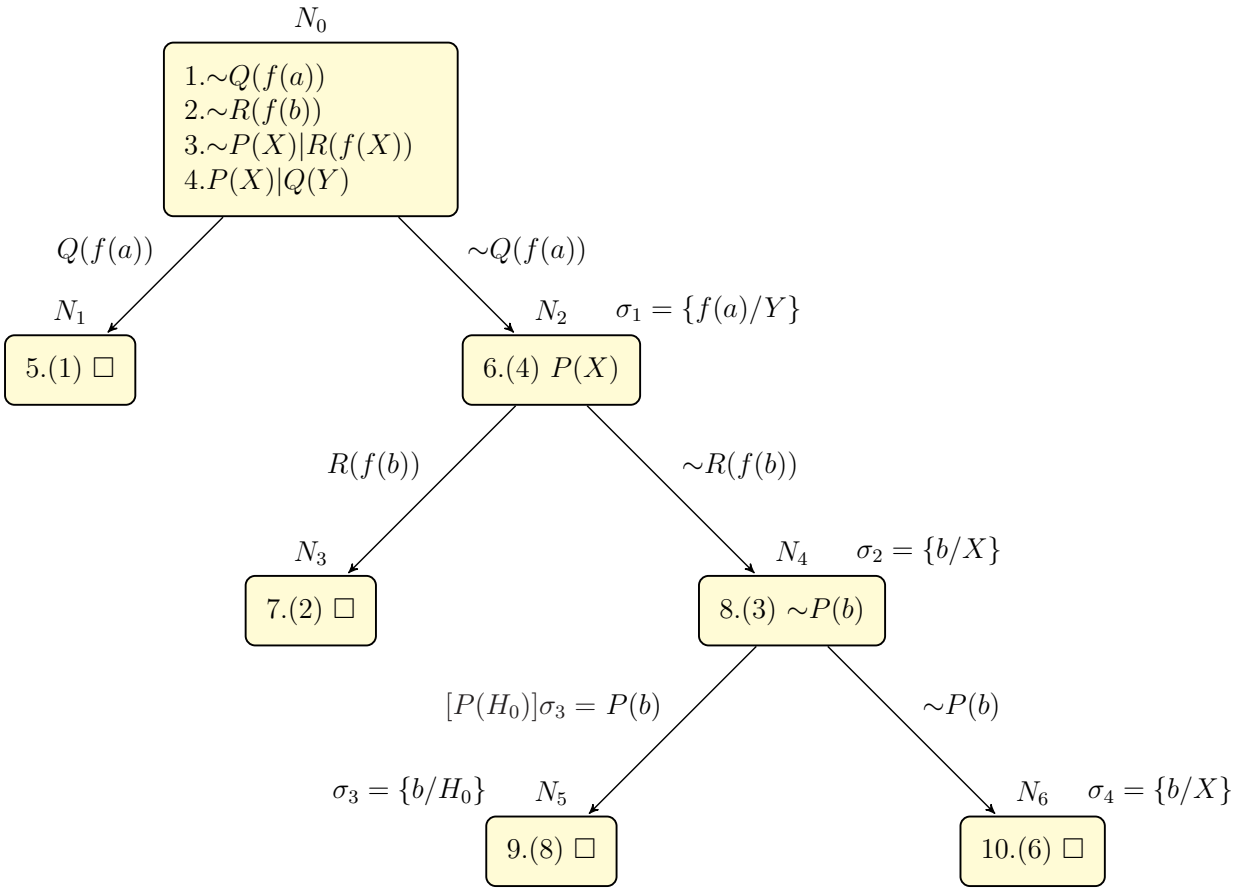

Figure 5.7: Closed semantic tree of example 5.3.1 with placeholder variable grounding after only 1^{st} generation of the *SHB*.

As we can see in example 5.3.1, the conditional canonical order approach can be wasteful, since it first instantiates X by a and then by b, although only the latter is needed. However, the placeholder variable grounding approach replaces X by the placeholder H_0 which is directly instantiated by b. That is, in this example we expect the placeholder variable grounding method to be better in time processing (faster) and in memory usage than the conditional canonical order grounding method as it allows the semantic tree to choose the required substitution as per its need from the unification during the building of the semantic tree.

CHAPTER 6

Soundness & Completeness

The SSTTP theorem prover is based on a combination of the first-order resolution strategy and a Herbrand strategy for introducing a refutation theorem proving procedure by constructing a closed semantic tree. In order to show that the SSTTP calculus is sound, we have to show that for any set of clauses Φ if from Φ a closed semantic tree has been constructed applying the calculus rules that then the set of clauses Φ is unsatisfiable. The corresponding proof is presented in Section 6.1. Moreover, the calculus is complete; that is, if a given set of clauses Φ is unsatisfiable, then it is possible to construct a closed semantic tree using the rules of the SSTTP calculus. In Section 6.2 a completeness proof of the proof procedure is presented. Furthermore we show that, if the SSTTP calculus rules are applied using a fair strategy, then a proof, if it exists, will be found in finitely many steps as shown by Corollary 6.2.3, since a fair strategy cannot postpone the application of a rule indefinitively and other rule applications will not prevent a rule from being applied.

6.1 Soundness proof

To prove the soundness of the SSTTP calculus, it suffices to show the validity of each rule of the SSTTP calculus; that is, to show that, if a clause set is satisfiable and a rule

is applied, at least one of the resulting clause sets is satisfiable (lemma 6.1.1). This corresponds to the contrapositive statement that, if a closed semantic tree can be constructed, then the original clause set must be unsatisfiable (theorem 6.1.2).

Lemma 6.1.1 *For each rule of the SSTTP calculus, if the first component of the premise of the rule is satisfiable (that is, the clause set (1) in the rule below is satisfiable), then one of the first components of its conclusions is satisfiable as well (that is, clause set (2) or clause set (3) is satisfiable).*

Proof The proof consists of three parts, namely showing that each of the three rules in calculus is sound.

- HBG: Since the clause set does not change in the HBG rule in Section 5.2, the property holds trivially.

- Build tree: According to Build tree rule in Section 5.2, let us abbreviate the first component of the premise as (1) and the first components of the conclusions as (2) and (3), respectively.

$$\frac{\overbrace{\overbrace{\Phi}^{(1)},\ \sim l|C,\ l'|C',\ \Phi'}^{(1)} \qquad \Psi \qquad [s]\sigma,\ SHB}{\underbrace{\sim l|C,\ l'|C',\ \Phi',\ [C]\sigma}_{(2)} \quad \Phi,\ [C]\sigma \quad SHB \quad | \quad \underbrace{\sim l|C,\ l'|C',\ \Phi',\ [C']\sigma}_{(3)} \quad \Phi,\ [C']\sigma \quad SHB}$$

$$\text{with } \begin{cases} \sigma = mgu(l, l', s) \\ [C]\sigma \notin \Phi \text{ and } [C']\sigma \notin \Phi \end{cases}$$

Assume (1) is satisfiable; that is, there is a model $\mathcal{M}$ that satisfies (1). Assume furthermore that (2) is unsatisfiable, then $\mathcal{M}$ is not a model of (2). Now we show that $\mathcal{M}$ is a model of (3), hence (3) is satisfiable.

Looking at (3), it consists of four components; the first three are the same as (1), for which $\mathcal{M}$ is a model. On the other hand, if we resolve $(\sim l|C)$ with $(l'|C')$ in (1), we get $[C]\sigma|[C']\sigma$ where σ is always ground even if there are a placeholder variables, i.e., they act as a ground terms. Because resolution is sound, $\mathcal{M}$ will satisfy at least one of $[C]\sigma$ and $[C']\sigma$. Since (2), which is $\sim l|C, l'|C', \Phi', [C]\sigma$, is unsatisfiable and $\mathcal{M}$ is a model of the first three components of (2), $\mathcal{M}$ cannot satisfy $[C]\sigma$. Hence, $\mathcal{M}$ must model the other part of the resolution $[C']\sigma$. Thus, $\mathcal{M}$ satisfies (3).

- Close rule: Since the premise of the Close rule in Section 5.2 contains the empty clause, which is always unsatisfiable, the lemma 6.1.1 is trivially true (the premise of the lemma 6.1.1 is not satisfiable). ∎

Theorem 6.1.2 (Soundness) *For all clauses in* Φ*, if* Φ *has a closed semantic tree, then* Φ *is unsatisfiable.*

Proof Let T be a subtree of a semantic tree and let N be its root. If T is a one-node tree, then N can only have an empty clause ($\square$), which is trivially unsatisfiable. If T has more than one node, we can assume by induction that all the children nodes of N are unsatisfiable. Furthermore, we can then also conclude that N is unsatisfiable by the contrapositive of Lemma 6.1.1. ∎

6.2 Completeness & Fairness

In this section we will first see a proof of the completeness of the SSTTP calculus. The idea of this proof is to show that any resolution proof, which must exist because of the completeness of resolution, can be transformed into an SSTTP proof by using a complexity measure for the clause set. The complexity, $c(C)$, of a clause, C, is defined as the number of literals, l_i, in C. We prove the completeness by induction on the complexity of the

clause set Φ. Furthermore we show in this section that a fair strategy will eventually construct a proof if it exists.

Definition (Complexity) Let Φ be a set of clauses. Thus, Φ is either the empty clause, $\Box$, or it is a conjunction of clauses, $\Phi = C_1, \ldots, C_n$. We define the complexity, $c(C)$, of a clause, C, as follows:

$$c(C) = \begin{cases} 0 & \text{If } C = \Box \\ m - 1 & \text{If } C = l_1 | \ldots | l_m \quad \text{where} \quad m \geq 1 \end{cases}$$

If Φ is a conjunction of clauses then

$$c(\Phi) = \sum_{i=1}^{n} c(C_i) \quad \text{for } \Phi = C_1, \ldots, C_n$$

Remark Since the *SHB* is generated by binary resolution steps, at any stage in the proof construction it contains only finitely many elements. Hence the HBG rule can be applied only finitely often before the *SHB* will be regenerated.

Theorem 6.2.1 (Completeness) *For all clauses in Φ, if Φ is unsatisfiable, then there exists a proof (closed semantic tree) using rules of the SSTTP calculus. Therefore, the SSTTP calculus is complete.*

Proof We prove the theorem by induction on $c(\Phi)$. Let $\Phi = C_1, \ldots, C_n$. Assume $\Phi \neq \{\Box\}$, since the case $\Phi = \{\Box\}$ is trivial.

- Base case: If $c(\Phi) = 0$, then every clause C_i contains a single literal and if Φ is unsatisfiable, then there must be two complementary clauses, $C_i = \sim l$ and $C_j = l'$ in Φ with $\sigma = mgu(l, l')$. Thus, by the HBG rule, we add $[l]\sigma$ to the *SHB* set. Then, by the Build tree rule, we build a semantic tree using $[l]\sigma$ from the *SHB*.

Accordingly, we split the tree with the positive $[l]\sigma$ in the left branch in order to get the empty clause by resolving $[l]\sigma$ with the clause C_i. The other split part consists of the negative $[\sim l]\sigma$ in the right branch to get the empty clause by resolving $[\sim l]\sigma$ with the clause C_j. Finally, by the Close rule, we close the left branch because of the empty clause and we close the right branch because of the empty clause.

Concretely, consider Φ is an unsatisfiable set of clauses. Let C_0 be the clause containing $\sim l$ and C_1 the clause containing l'. The rest of the clauses in Φ is called Φ'. The following is a SSTTP calculus proof:

$$\frac{\Phi \qquad C_0, C_1, \Phi' \qquad SHB}{\Phi \qquad \square, \Phi' \qquad SHB, [l]\sigma}\ \text{HBG rule}$$

$$\frac{}{C_0, C_1, \Phi', \square \quad \Phi, \square \quad SHB \ \mid\ C_0, C_1, \Phi', \square \quad \Phi, \square \quad SHB}\ \text{Build tree rule}$$

$$\frac{}{\square \quad \square \quad SHB \ \mid\ \square \quad \square \quad SHB}\ \text{Close rule}$$

- Step case: Let the complexity for $c(\Phi)$ be greater than 0. We assume that the completeness of SSTTP for all clause sets with complexity smaller than $c(\Phi)$ holds. We have to show that we can apply SSTTP rules such that each subproblem has a complexity smaller than $c(\Phi)$. Since $c(\Phi) > 0$ there are some clauses in Φ that contain at least two literals, and since Φ is an unsatisfiable set, there exists a resolution proof for Φ. In each step of the resolution proof, we have a cut literal. Consider two clauses $(A \cup \{l\})$ and $(B \cup \{l'\})$, where l and l' are two complementary literals with $\sigma = mgu(l, l')$, then the resolvent is $([A]\sigma|[B]\sigma)$ and the cut part is $[l]\sigma$. This cut part $[l]\sigma$ from the resolution proof is generated by the HBG rule, which accumulates this $[l]\sigma$ in the *SHB*.

  Now, pick this l and the clause C that contains l. Write $C = A \cup \{l\}$ where $l \notin A$ and

$A \neq \{\square\}$, since C contains at least two literals. Write $\Phi = \Delta, C$ (note that Δ cannot be empty since Φ is unsatisfiable). As $\Phi = \Delta, A \cup \{l\}$ is unsatisfiable, both Δ, A and Δ, l must be unsatisfiable. However, observe that $c(\Delta, A) < c(\Phi)$ (this follows from the definition of c since $\Phi = \Delta, A \cup \{l\}$) and $c(\Delta, l) < c(\Phi)$. Therefore, by the induction hypothesis, the SSTTP calculus produces two closed semantic trees, T_1 and T_2, with sets of premises Δ, A and Δ, l, respectively as shown in Figure 6.1.

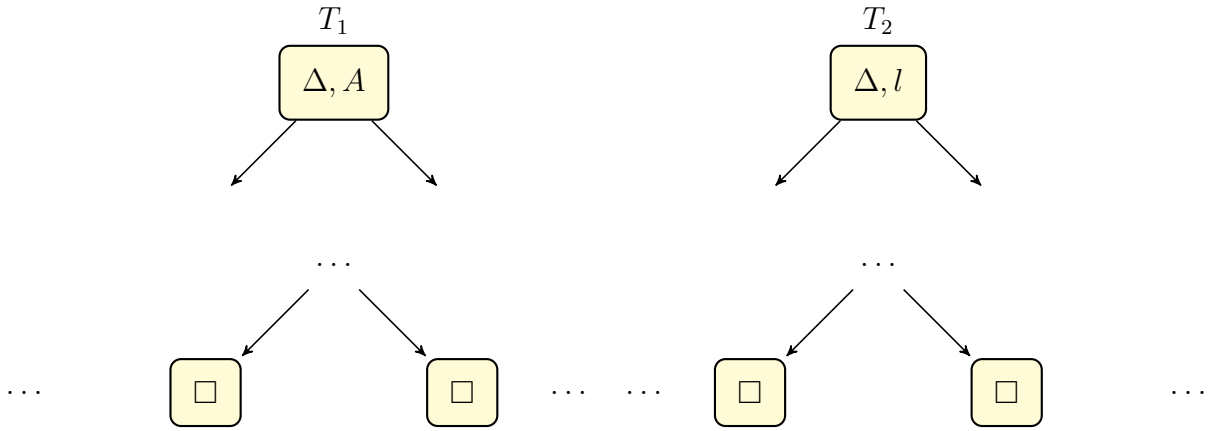

Figure 6.1: Closed semantic tree of T_1 and T_2.

Now, consider the semantic tree T_1', obtained from T_1 by adding l to the clause A and letting l percolate down to the leaves.

Observe that in T_1', every clause that is a descendant of the premise $A \cup \{l\}$ is of the form $C \cup \{l\}$, where C is the corresponding clause in T_1. Therefore, the leaves of the new T_1' obtained from T_1 either contain the empty clause (when another clause already contains l), or l.

- Case 1: If $l \in \Delta$. A leaf of T_1' contains the empty clause:

  T_1' is a closed semantic tree by the Close rule and we are done. Figure 6.2 shows the closed semantic tree of T_1' , if $l \in \Delta$.

- Case 2: If $l \notin \Delta$. A leaf of T_1' contains l:

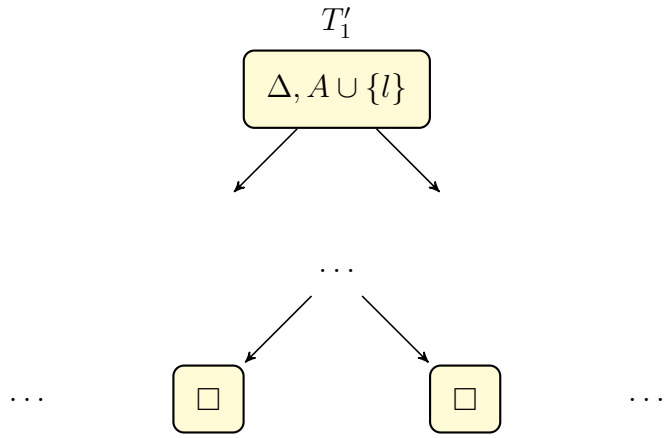

Figure 6.2: Closed semantic tree of T_1' when $l \in \Delta$.

We can combine T_1' and T_2 using both *SHB* sets of T_1 and T_2. Since the leaves of T_1' also contain l, one of the premises of T_2. We obtain a closed semantic tree for Φ. Figure 6.3 shows the closed semantic tree of T_1' , if $l \notin \Delta$.

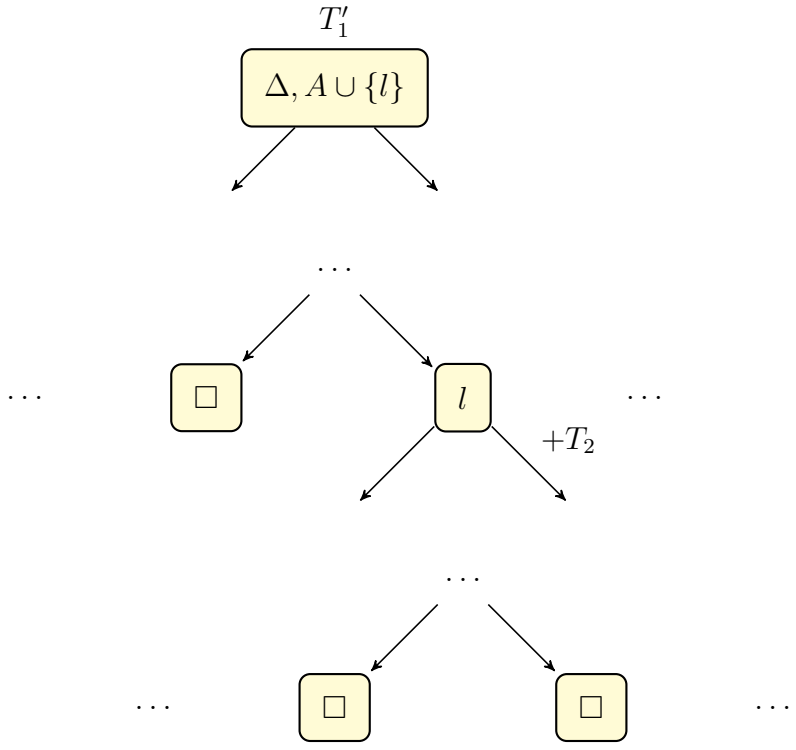

Figure 6.3: Closed semantic tree of T_1' when $l \notin \Delta$

This concludes the proof. ∎

Whilst the completeness property guarantees that, for each unsatisfiable clause set there is an SSTTP proof, fairness in the application of the rules guarantees that such a proof will eventually be found. Therefore, any applicable rule from the SSTTP proof sequence must not be postponed indefinitely.

Definition (Fairness) Fairness means that whenever a concrete rule application is possible, it will not be postponed indefinitely.

Definition $(R - sequence)$ $R = \{r_1, \ldots, r_n\}$ is a sequence of rules from SSTTP that consists of three different rules:

- HBG rule: r_i where $0 \leq i \leq n$.
- Build tree rule: r_j where $0 \leq j \leq n$.
- Close rule: r_m where $0 \leq m \leq n$.

Definition (π) A permutation π on a set R is a bijective (one-to-one) map from R to itself.

In order to show the fairness result, it is important to know that a permutation of a given sequence of rules from SSTTP can also produce a proof of Φ, i.e., if we reorder the applicable rules then we still can get a proof if it exists. The next lemma will make this formal.

Lemma 6.2.2 *If* $R = \{r_1, \ldots, r_n\}$ *is a proof of* Φ *from SSTTP and* $\pi(R) = \{r_{\pi_1}, \ldots, r_{\pi_n}\}$ *is an applicable sequence of rules, then* $\pi(R)$ *is a proof of* Φ *from SSTTP.*

Proof Assume Φ is an unsatisfiable set of clauses and R is a set of applicable rules that form an SSTTP proof of Φ. Remember the SSTTP calculus consists of three rules:

the HBG rule, the Build tree rule and the Close rule. To ensure the fairness of the SSTTP proof, we have to show that, if we reorder any two applicable rules from R, we still have a proof.

Since the sequence R is a proof, one of the rules r_i must be the Close rule. It is trivial that r_i can be reordered with any rule and postponed until the end of the set because the Close rule terminates any node that is applicable to use the Close rule. So, it does not matter if we use the Close rule first or another applicable rule first (assumed that the Close rule remains applicable).

Now, we have to show that if we reorder the HBG rule and the Build tree rule, the proof is still fulfilled. Assume r_i comes before r_j in the sequence R, then we show that if we reorder R by a permutation π so that r_j comes before r_i in $\pi(R)$, then the sequence $\pi(R)$ is still a proof of Φ. Since R is a combination of the HBG rule, the Build tree rule and the close rule, and the reorder of the close rule is trivial, as stated earlier, we need to consider only the following cases:

Case 1:	r_i and r_j are both HBG rules.
Case 2:	r_i is a HBG rule and r_j is a Build tree rule.
Case 3:	r_i and r_j are both Build tree rules.

- Case 1: There are at least four literals in Φ. This gives us two resolvents. Accordingly, in this case we can carry out two resolution steps: first, r_i will produce $[l_k]\sigma_k$ and put it in the *SHB*; and second, r_j will produce $[l_m]\sigma_m$ and put it in the *SHB*. Therefore, $SHB = \{[l_k]\sigma_k, [l_m]\sigma_m\}$. Moreover, if we change the order of the rules–that is, if we start with r_j first then r_i–then we have $SHB' = \{[l_m]\sigma_m, [l_k]\sigma_k\}$. Since, $SHB = SHB'$, $\pi(R)$ is also a proof.

- Case 2: Since both of them are applicable at this stage, $SHB \neq \{\square\}$. If we first execute r_i then we will add a new atom $[l_k]\sigma_k$ to the *SHB*. Therefore, there will

be at least two atoms in the *SHB*. Then the execution of r_j will not depend on r_i because it will use one of the atoms that is already inside the *SHB* to split the tree and it is not necessary to be $[l_k]\sigma_k$. Since r_i and r_j do not depend on each other, $\pi(R)$ is also a proof.

- Case 3: In this case, both rules are applications of the Build tree rule. Since the Build tree rule depends on resolution between two literals, there should be two resolution steps (one for r_i and the other for r_j). Therefore, the set of clauses consists of two complementary pairs of literals $\{\sim l_k, l'_k\}$ and $\{\sim l_m, l'_m\}$. So, the set of clauses Φ can be one of the following cases:

  Case 3-1: each literal of complementary pair located in separate clause.
  $\Phi = \sim l_k | C_k,\ l'_k | C'_k,\ \sim l_m | C_m,\ l'_m | C'_m,\ \Phi'$

  Case 3-2: literals complementary pair located in two clauses.
  $\Phi = \sim l_k | \sim l_m | C_k,\ l'_k | l'_m | C'_k,\ \Phi'$

  Case 3-3: literals complementary pair located in three clauses.
  $\Phi = \sim l_k | C_k,\ l'_k | \sim l_m | C_m,\ l'_m | C'_m,\ \Phi'$

  Now, start proving each case separately:

  - Case 3-1: Assume $\Phi = \sim l_k | C_k,\ l'_k | C'_k,\ \sim l_m | C_m,\ l'_m | C'_m,\ \Phi'$ and $SHB = \{[s_i]\sigma_i, [s_j]\sigma_j, [s_t]\sigma_t, \ldots, [s_n]\sigma_n\}$. The r_i will split using $[s_i]\sigma_i$ atom and the r_j will split using $[s_j]\sigma_j$ atom. If we start first with r_i we get the following two branches:

$$r_i \ \frac{\begin{array}{cccccccc} & & \Phi & & \Psi & & \{[s_i]\sigma_i, [s_j]\sigma_j, [s_t]\sigma_t, \ldots\} & \\ \underbrace{\Phi,\ [C_k]\sigma_i}_{N_1} & N_1 & \{[s_j]\sigma_j, \ldots\} & | & \underbrace{\Phi,\ [C'_k]\sigma_i}_{N_2} & N_2 & \{[s_j]\sigma_j, \ldots\} & \end{array}}{}$$

If we split now with r_j we get the following four branches:

$$r_j \;\frac{\underbrace{\Phi,\ [C_k]\sigma_i,\ [C_m]\sigma_j}_{N_3}\quad N_3\quad \{[s_t]\sigma_t,\dots\}\quad |\quad \underbrace{\Phi,\ [C_k]\sigma_i,\ [C'_m]\sigma_j}_{N_4}\quad N_4\quad \{[s_t]\sigma_t,\dots\}}{N_1 \qquad N_1 \qquad \{[s_j]\sigma_j,\dots\}}$$

$$r_j \;\frac{\underbrace{\Phi,\ [C'_k]\sigma_i,\ [C_m]\sigma_j}_{N_5}\quad N_5\quad \{[s_t]\sigma_t,\dots\}\quad |\quad \underbrace{\Phi,\ [C'_k]\sigma_i,\ [C'_m]\sigma_j}_{N_6}\quad N_6\quad \{[s_t]\sigma_t,\dots\}}{N_2 \qquad N_2 \qquad \{[s_j]\sigma_j,\dots\}}$$

Now let us see the permutation, which will start first with r_j and get the following two branches:

$$r_j \;\frac{\underbrace{\Phi,\ [C_m]\sigma_j}_{N'_1}\quad N'_1\quad \{[s_i]\sigma_i,\dots\}\quad |\quad \underbrace{\Phi,\ [C'_m]\sigma_i}_{N'_2}\quad N'_2\quad \{[s_i]\sigma_i,\dots\}}{\Phi \qquad \Psi \qquad \{[s_j]\sigma_j,[s_i]\sigma_i,[s_t]\sigma_t,\dots\}}$$

If we split now with r_i we get the following four branches:

$$r_i \;\frac{\underbrace{\Phi,\ [C_m]\sigma_j,\ [C_k]\sigma_i}_{N_3}\quad N_3\quad \{[s_t]\sigma_t,\dots\}\quad |\quad \underbrace{\Phi,\ [C_m]\sigma_j,\ [C'_k]\sigma_i}_{N_5}\quad N_5\quad \{[s_t]\sigma_t,\dots\}}{N'_1 \qquad N'_1 \qquad \{[s_i]\sigma_i,\dots\}}$$

$$r_i \;\frac{\underbrace{\Phi,\ [C'_m]\sigma_j,\ [C_k]\sigma_i}_{N_4}\quad N_4\quad \{[s_t]\sigma_t,\dots\}\quad |\quad \underbrace{\Phi,\ [C'_m]\sigma_j,\ [C'_k]\sigma_i}_{N_6}\quad N_6\quad \{[s_t]\sigma_t,\dots\}}{N'_2 \qquad N'_2 \qquad \{[s_i]\sigma_i,\dots\}}$$

Since r_i and r_j do not depend on each other, i. e., r_i used $\sim l_k|C_k,\ l'_k|C'_k$ to split with $[s_i]\sigma_i$ and r_j used $\sim l_m|C_m,\ l'_m|C'_m$ to split with $[s_j]\sigma_j$, the derivation steps of the rules give the same results after reordering the rules. Hence $\pi(R)$ is also a proof.

– Case 3-2: Assume $\Phi = \sim l_k|\sim l_m|C_k,\ l'_k|l'_m|C'_k,\ \Phi'$ and $SHB = \{[s_i]\sigma_i, [s_j]\sigma_j, [s_t]\sigma_t, \dots, [s_n]\sigma_n\}$. The r_i will split using $[s_i]\sigma_i$ atom and the r_j will split using $[s_j]\sigma_j$ atom. If we start first with r_i we get the following

two branches:

$$r_i \;\frac{\Phi \quad \Psi \quad \{[s_i]\sigma_i, [s_j]\sigma_j, [s_t]\sigma_t, \dots\}}{\underbrace{\Phi,\ [\sim l_m|C_k]\sigma_i}_{N_1} \quad N_1 \quad \{[s_j]\sigma_j, \dots\} \quad | \quad \underbrace{\Phi,\ [l'_m|C'_k]\sigma_i}_{N_2} \quad N_2 \quad \{[s_j]\sigma_j, \dots\}}$$

If we split now with r_j we get the following four branches:

$$r_j \;\frac{N_1 \quad N_1 \quad \{[s_j]\sigma_j, \dots\}}{\underbrace{N_1,\ [\sim l_k|C_k]\sigma_j}_{N_3} \quad N_3 \quad \{[s_t]\sigma_t, \dots\} \quad | \quad \underbrace{N_1,\ [l'_k|C'_k]\sigma_j}_{N_4} \quad N_4 \quad \{[s_t]\sigma_t, \dots\}}$$

$$r_j \;\frac{N_2 \quad N_2 \quad \{[s_j]\sigma_j, \dots\}}{\underbrace{N_2,\ [\sim l_k|C_k]\sigma_j}_{N_5} \quad N_5 \quad \{[s_t]\sigma_t, \dots\} \quad | \quad \underbrace{N_2,\ [l'_k|C'_k]\sigma_j}_{N_6} \quad N_6 \quad \{[s_t]\sigma_t, \dots\}}$$

Now let us see the permutation, which will start first with r_j then r_i. We get the following two branches:

$$r_j \;\frac{\Phi \quad \Psi \quad \{[s_j]\sigma_j, [s_i]\sigma_i, [s_t]\sigma_t, \dots\}}{\underbrace{\Phi,\ [\sim l_k|C_k]\sigma_j}_{N'_1} \quad N'_1 \quad \{[s_i]\sigma_i, \dots\} \quad | \quad \underbrace{\Phi,\ [l'_k|C'_k]\sigma_j}_{N'_2} \quad N'_2 \quad \{[s_i]\sigma_i, \dots\}}$$

If we split now with r_i we get the following four branches:

$$r_i \;\frac{N'_1 \quad N'_1 \quad \{[s_i]\sigma_i, \dots\}}{\underbrace{N'_1,\ [\sim l_m|C_k]\sigma_i}_{N_3} \quad N_3 \quad \{[s_t]\sigma_t, \dots\} \quad | \quad \underbrace{N'_1,\ [l'_m|C'_k]\sigma_i}_{N_5} \quad N_5 \quad \{[s_t]\sigma_t, \dots\}}$$

$$r_i \;\frac{N'_2 \quad N'_2 \quad \{[s_i]\sigma_i, \dots\}}{\underbrace{N'_2,\ [\sim l_m|C_k]\sigma_i}_{N_4} \quad N_4 \quad \{[s_t]\sigma_t, \dots\} \quad | \quad \underbrace{N'_2,\ [l'_m|C'_m]\sigma_i}_{N_6} \quad N_6 \quad \{[s_t]\sigma_t, \dots\}}$$

Since firstly r_i and r_j are not dependent on each other because they use different substitutions, i. e., r_i used $[s_i]\sigma_i$ and r_j used $[s_j]\sigma_j$, and since secondly the derivation steps of the rules give the same results after reordering the rules, it follows that $\pi(R)$ is also a proof.

– Case 3-3: Assume $\Phi = \sim l_k|C_k,\ l'_k|\sim l_m|C_m,\ l'_m|C'_m,\ \Phi'$ and $SHB = \{[s_i]\sigma_i, [s_j]\sigma_j, [s_t]\sigma_t, \ldots, [s_n]\sigma_n\}$. The r_i will split using $[s_i]\sigma_i$ atom and the r_j will split using $[s_j]\sigma_j$ atom. If we start first with r_i we get the following two branches:

$$r_i \ \frac{\underbrace{\Phi,\ [C_k]\sigma_i}_{N_1} \quad N_1 \quad \{[s_j]\sigma_j, \ldots\} \quad | \quad \underbrace{\Phi,\ [\sim l_m|C_m]\sigma_i}_{N_2} \quad N_2 \quad \{[s_j]\sigma_j, \ldots\}}{\Phi \quad \Psi \quad \{[s_i]\sigma_i, [s_j]\sigma_j, [s_t]\sigma_t, \ldots\}}$$

If we split now with r_j we get the following four branches:

$$r_j \ \frac{\underbrace{N_1,\ [l'_k|C_m]\sigma_j}_{N_3} \quad N_3 \quad \{[s_t]\sigma_t, \ldots\} \quad | \quad \underbrace{N_1,\ [C'_m]\sigma_j}_{N_4} \quad N_4 \quad \{[s_t]\sigma_t, \ldots\}}{N_1 \quad N_1 \quad \{[s_j]\sigma_j, \ldots\}}$$

$$r_j \ \frac{\underbrace{N_2,\ [l'_k|C_m]\sigma_j}_{N_5} \quad N_5 \quad \{[s_t]\sigma_t, \ldots\} \quad | \quad \underbrace{N_2,\ [C'_m]\sigma_j}_{N_6} \quad N_6 \quad \{[s_t]\sigma_t, \ldots\}}{N_2 \quad N_2 \quad \{[s_j]\sigma_j, \ldots\}}$$

Now let us see the permutation, which will start first with r_j we get the following two branches:

$$r_j \ \frac{\underbrace{\Phi,\ [l'_k|C_m]\sigma_j}_{N'_1} \quad N'_1 \quad \{[s_i]\sigma_i, \ldots\} \quad | \quad \underbrace{\Phi,\ [C'_m]\sigma_j}_{N'_2} \quad N'_2 \quad \{[s_i]\sigma_i, \ldots\}}{\Phi \quad \Psi \quad \{[s_j]\sigma_j, [s_i]\sigma_i, [s_t]\sigma_t, \ldots\}}$$

If we split now with r_i we get the following four branches:

$$r_i \ \frac{\underbrace{N'_1,\ [C_k]\sigma_i}_{N_3} \quad N_3 \quad \{[s_t]\sigma_t, \ldots\} \quad | \quad \underbrace{N'_1,\ [\sim l_m|C_m]\sigma_i}_{N_5} \quad N_5 \quad \{[s_t]\sigma_t, \ldots\}}{N'_1 \quad N'_1 \quad \{[s_i]\sigma_i, \ldots\}}$$

$$r_i \ \frac{\underbrace{N'_2,\ [C_k]\sigma_i}_{N_4} \quad N_4 \quad \{[s_t]\sigma_t, \ldots\} \quad | \quad \underbrace{N'_2,\ [\sim l_m|C_m]\sigma_i}_{N_6} \quad N_6 \quad \{[s_t]\sigma_t, \ldots\}}{N'_2 \quad N'_2 \quad \{[s_i]\sigma_i, \ldots\}}$$

Since r_i and r_j do not depend on each other because they used different substitutions, i. e., r_i used $[s_i]\sigma_i$ and r_j used $[s_j]\sigma_j$, the derivation steps of the rules give the same results after reordering the rules. That is, $\pi(R)$ is also a proof. ∎

Corollary 6.2.3 (Fairness) *If the strategy for the application of SSTTP rules is fair then for any unsatisfiable clause set an SSTTP proof will be constructed eventually.*

The corollary implies that as long as the SSTTP rules guarantee fairness throughout the process of implementing the applicable rules, the order is not important in terms of gathering the SSTTP proof of a given unsatisfiable clause set if the postponed rules are executed eventually.

CHAPTER 7

Implementation, Heuristics, and Experiments

For efficient implementation, SSTTP requires heuristics and a proper way of dealing with variables. In case of variables instantiation, Bjork [9, 10] introduced a rule known as the dilemma rule, which deals with rigid variables in a tableaux procedure. His system does not allow any unification of rigid variables, except when the dilemma branches are merged. Bjork was interested to find the instances of rigid variables that can help his system progress further. Moreover, Bjork describes the difference between the rigid variables and the constants. He said that the difference arises when we define substitutions, which are finite mappings from both universal and rigid variable symbols to terms. In other work, Voronkov [51] presents various strategies that deal with rigid variables. He studied the complexity of methods using rigid variables, such as the method of matings or the tableau method, on a decidable subclass of the predicate calculus with equality. He also dealt with strategies for increasing multiplicity in rigid-variable methods. In this dissertation, the placeholder variables are acting as rigid variables. They are used to replace the variables in the *SHB* and they are treated as constants until they are intantiated with a ground terms globally during the building of a semantic tree. In the SSTTP prover, the placeholder variables are used to deal with variable instantiation, as described in Section 5.3.

In the following, we will first briefly describe the implementation of the SSTTP prover and then we add the heuristics to the SSTTP prover in order to improve its overall performance. Such heuristic techniques help us to effectively select atoms from the Herbrand base to build the *SHB*. Section 7.2 introduces the heuristics that are implemented and tested in our prover. After that, Section 7.3 presents the comparison between the heuristics.

7.1 SSTTP Implementation

In this section, we briefly describe the data structure that is used in the SSTTP program and the phases the program goes through in order to generate the proof in Subsections 7.1.1 and 7.1.2, respectively.

7.1.1 SSTTP Data Structure

This subsection introduces the data structure of the SSTTP program. The data structure presents the clause set as a structure of a link list of clauses. Each clause is built as a link list of terms, which is constructed as a tree of a child and a sibling. For example, a clause $P_0(X_0, f_0(c_0, X_1))|P_0(c_0, f_1(f_1(c_1)))$ will be presented in a tree structure as shown in Figure 7.1.

In addition, a parser is built to convert the CNF format from the TPTP library to a clause set by using the Bison and Flex software. Furthermore, the SSTTP functions are constructed.

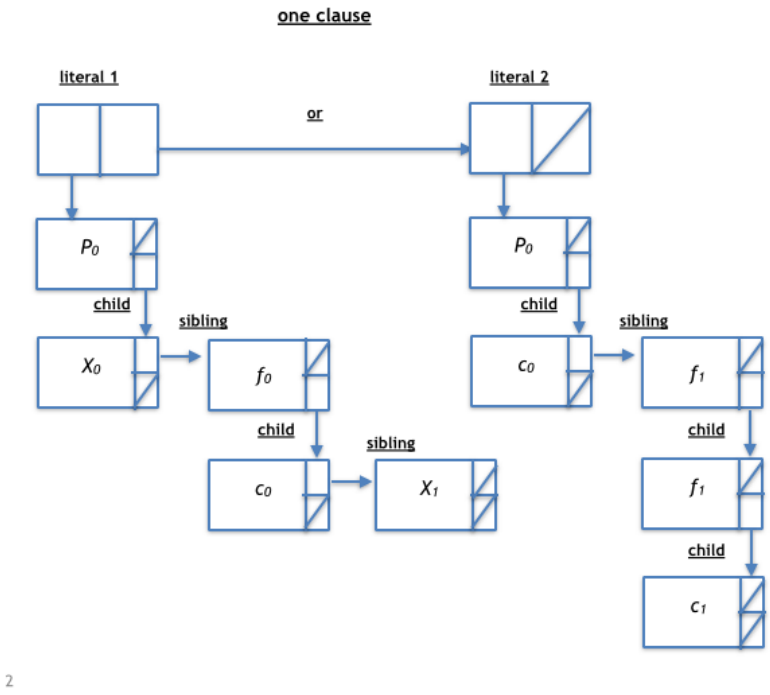

Figure 7.1: Data structure graph for a clause.

7.1.2 SSTTP phases

This subsection describes the construction of the Smart Semantic tree Theorem Prover (SSTTP). This program will be used to achieve the effective construction of closed semantic trees of unsatisfiable clauses. The SSTTP program is divided into three phases. The first phase is a parsing module that operates on a given theorem. It converts the input set into a simple format that is required by the second phase. In the second phase, a collecting module gathers the *SHB* atoms with the help of the HBG algorithm, as described in Section 5.1. Finally, the last phase is concerned with building the semantic tree by using the atoms that were generated in the second phase. The following points describe the three phases in more detail:

1. Parsing TPTP problems:

   Given a theorem (i.e. an unsatisfiable set S of clauses) selected from the TPTP library, the 'ssttp_parser.y' and 'ssttp_lexer.l' modules convert a conjunctive normal form (CNF) format of the given input from TPTP to a clause set. With the help of some functions from 'clauses.c' module in the SSTTP program, a simple clause set format that represents the given problem is generated, which the rest of the SSTTP program will use. The methods are implemented using the Flex and Bison syntax [26].

2. Collecting Smart Herbrand Base atoms:

   Given a set of clauses from the parser, the 'collect.c' module will generate the so-called *SHB* with the help of functions from the 'rename.c', 'unify.c' and 'meta_var.c' modules. Each one of these modules has a number of functions. An important function is 'collect_SHB()' function from 'collect.c'. This function executes the HBG algorithm, which computes the smart atoms, and collects them in the *SHB* set with the help of other functions in the program.

3. Building closed semantic tree:

   Using the simplified clauses, which were generated by the parsing modules, and the *SHB* atoms, which were generated by the collecting modules, the 'semantic.c' module builds a closed semantic tree.

7.2 Heuristics

The heuristic strategies enhance the elimination of useless nodes in building semantic trees. In this dissertation four heuristic strategies will be introduced and studied. Subsection 7.2.1 presents the unit preference heuristic with an example. The idea of heuristic h1 is that it is beneficial to apply splitting on units first. Subsection 7.2.2 describes the degree order heuristic. The degree order heuristic h2 is a generalization of the units first heuristic h1 in that it applies splitting on simpler formulae first in order to obtain smaller semantic trees. Subsection 7.2.3 shows the impact heuristic h3. Its idea is to select the element in the *SHB* that makes the biggest impact in the sense that the corresponding atom resolves with as many clauses as possible. Finally, Subsection 7.2.4 introduces a strategy h4 that is used to eliminate useless atoms inside the *SHB* whilst building the semantic tree. These heuristic strategies were implemented and tested in the SSTTP prover. After describing the heuristics in detail, an empirical comparison is drawn between all four heuristics, as shown in Section 7.3 using the relevant problems from the TPTP problem collection [47].

7.2.1 SSTTP-h1: Unit heuristic

The unit heuristic strategy used in the SSTTP prover is similar to the unit-preference strategy that Nilsson [33] used in resolution to reduce the size of the literals.

Definition (Unit heuristic) Let $\Phi = \{P_1, \ldots, P_n\}$ be given. A Smart Herbrand Base generated applying the Unit restriction heuristic h1 means that for any two elements hb_i and hb_j in $SHB = (hb_1, \ldots, hb_m)$ holds if $i < j$ and hb_j has been generated with the HBG rule by resolving on a unit then hb_i must also be generated by resolving on a unit, that is, all the elements in *SHB*, generated by resolving on a unit come before all the others in *SHB*.

The unit heuristic strategy collects all unit clauses from the base set of the given input problem, and incorporates them within the *SHB* before generating atoms using the SSTTP calculus. This will help the SSTTP prover to close at least one node immediately after each level of the semantic tree. Hence, the complexity of the semantic tree is reduced. Therefore, this strategy looks promising for the efficiency of the SSTTP prover in terms of the size of the semantic tree and of the processing time. Tables 7.1, 7.2 show the results of using the unit heuristic in testing the SSTTP prover with selected problems from the TPTP library [47]. For these test we used the most powerful computer in the School of Computer Science, GPU (with 24 CPU Intel 2.4 GHz computers with 47 GB of RAM), and limited each run to at most 100 seconds. The selected problems presented in this chapter are a subset of the full tables that can be found in the appendix. Here, we choose only the problems that can be solved using one or both SSTTP with and without the use of the heuristics to make the comparison fair. From the tables, we can observe that 18 problems – highlighted in green – out of 57 (31.58%) were proved by SSTTP-h1 in less time and with a smaller number of nodes in the semantic tree than without the heuristics. Moreover, (52.63%) of the results are the same, and on 9 results highlighted in red (15.79%) h1 did not performed so well as without the heuristics.

For example, observe the problem number 41 (LCL446-2.p) in the Table 7.2. SSTTP without the heuristics get a closed semantic tree with only 15 nodes in zero second. Whereas, SSTTP with the unit heuristic cannot find a proof even until 100 seconds. Because of the atoms chosen inside *SHB*. SSTTP-h1 waste time using useless unit atoms first and postponing the useful atoms that generated by HBG algorithm. Example 7.2.1 shows the *SHB* that give us a closed semantic tree for the problem (LCL446-2.p).

	Theorem	SSTTP		SSTTP-h1	
		time	nodes number	time	nodes number
1	ANA013-2	0.000	7	0.000	9
2	ANA041-2	0.000	3	0.000	3
3	ANA042-2	0.000	3	0.000	5
4	CAT007-3	0.470	1739	0.000	11
5	COL101-2	0.000	3	0.000	3
6	COL102-2	0.190	503	0.200	503
7	COL103-2	0.000	3	0.000	3
8	COL104-2	0.200	503	0.190	503
9	COL105-2	0.000	5	0.000	5
10	COL109-2	2.640	1495	1.620	1101
11	COL111-2	0.000	5	0.000	5
12	COL112-2	0.000	7	0.000	11
13	COL113-2	0.000	3	0.000	3
14	COL114-2	0.000	5	0.000	5
15	COL115-2	0.000	7	0.000	11
16	COL116-2	0.000	3	0.000	3
17	COL117-2	2.580	1493	0.000	19
18	COL119-2	0.000	5	0.000	5
19	COL120-2	0.000	5	0.000	5
20	COL122-2	0.000	5	0.000	5
21	COL124-2	T/O	-	1.120	1735
22	COM001-1	0.090	453	0.010	101
23	COM002-1	2.260	2769	1.760	1993
24	COM002-2	0.490	1809	0.020	143
25	FLD006-3	2.960	1097	0.000	11
26	FLD010-3	M/O	-	15.550	1377
27	GEO079-1	0.000	7	0.000	7

Table 7.1: The test results of SSTTP prover, with and without the unit heuristic, in the TPTP library. (T/O means time out and M/O means memory out)

	Theorem	SSTTP		SSTTP-h1	
		time	nodes number	time	nodes number
28	KRS004-1	0.000	5	0.000	5
29	LAT272-2	0.000	7	0.000	7
30	LAT273-2	M/O	-	0.000	13
31	LCL007-1	0.030	53	0.030	53
32	LCL076-2	0.000	11	0.000	11
33	LCL360-1	0.030	89	0.030	89
34	LCL432-2	0.000	5	0.000	5
35	LCL435-2	0.000	5	0.000	5
36	LCL436-2	0.000	5	0.000	5
37	LCL437-2	0.000	5	0.000	5
38	LCL438-2	0.000	21	0.000	23
39	LCL440-2	0.000	3	0.000	3
40	LCL441-2	0.000	17	0.000	19
41	LCL446-2	0.000	15	M/O	-
42	LCL447-2	0.000	7	0.000	7
43	MGT022-1	0.000	53	0.000	31
44	MGT022-2	0.000	53	0.000	31
45	MGT036-3	0.220	227	0.020	81
46	MGT041-2	0.000	23	0.000	15
47	PUZ008-1	0.440	297	0.290	151
48	PUZ012-1	0.680	2553	0.010	119
49	PUZ018-1	M/O	-	0.900	423
50	PUZ019-1	M/O	-	14.210	2041
51	PUZ035-1	0.030	357	0.030	357
52	PUZ035-2	0.040	429	0.040	429
53	PUZ035-3	1.150	1495	1.170	1495
54	PUZ035-4	1.030	1731	1.050	1731
55	PUZ035-5	0.000	89	0.010	91
56	PUZ035-6	0.000	89	0.010	91
57	PUZ035-7	0.300	477	0.310	477

Table 7.2: Continue Table 7.1, Test results of SSTTP prover with and without the unit heuristic in TPTP library. (T/O means time out and M/O means memory out)

Example 7.2.1 *Let us consider the following problem from the Logic Calculi domain (LCL446-2.p) in TPTP. Let S be the following set of clauses:*

1. $\sim P_0(f_0(c_0, c_0, c_1), f_1(c_2, c_1), f_2(c_1))$
2. $P_0(f_0(X_0, f_0(X_1, X_0, X_2), X_2), f_1(X_3, X_2), f_2(X_2))$
3. $\sim P_0(X_4, f_1(X_5, X_6), f_2(X_6))|$
   $\sim P_0(f_0(X_4, X_7, X_6), f_1(X_5, X_6), f_2(X_6))|P_0(X_7, f_1(X_5, X_6), f_2(X_6))$
4. $P_0(f_0(f_0(X_8, f_0(X_9, X_{10}, X_{11}), X_{11}), f_0(f_0(X_8, X_9, X_{11}), f_0(X_8, X_{10}, X_{11}),$
   $X_{11}), X_{11}), f_1(X_{12}, X_{11}), f_2(X_{11}))$

The following are the first 7 atoms of the SHB that builds a closed semantic tree. And these atoms are generated by SSTTP without using any heuristics:

$SHB = \{P_0(f_0(c_0, c_0, c_1), f_1(c_2, c_1), f_2(c_1)), P_0(f_0(H_0, f_0(H_1, H_0, c_1), c_1), f_1(c_2, c_1), f_2(c_1)),$
$P_0(H_2, f_1(c_2, c_1), f_2(c_1)),$
$P_0(f_0(f_0(H_3, f_0(H_4, H_5, c_1), c_1), f_0(f_0(H_3, H_4, c_1), f_0(H_3, H_5, c_1), c_1), c_1), f_1(c_2, c_1), f_2(c_1)),$
$P_0(f_0(c_0, f_0(c_0, c_0, c_1), c_1), f_1(c_2, c_1), f_2(c_1)), P_0(f_0(H_6, f_0(c_0, c_0, c_1), c_1), f_1(c_2, c_1), f_2(c_1)),$
$P_0(H_7, f_1(c_2, c_1), f_2(c_1)), \dots\}$.

The motivation behind the unit heuristic is to close branches containing unit clauses earlier to reduce the size of the semantic tree. Unfortunately, this heuristic is only helpful when the clause set contains unit clauses and the complementary literals of them. Also, the use of the unit heuristic is useful when the unit clauses are grounded; that is, it is better if we build semantic tree with grounded atoms from *SHB* as in example 7.2.2.

Example 7.2.2 *Let us consider the following problem from the Category Theory domain (CAT007-3.p) in TPTP. Let S be the following set of clauses:*

1. $P_0(X_0, X_0)$

2. $\sim P_0(X_1, X_2)|P_0(X_2, X_1)$

3. $\sim P_0(X_3, X_4)|\sim P_0(X_4, X_5)|P_0(X_3, X_5)$

4. $\sim P_1(f_0(X_6))|P_1(X_6)$

5. $\sim P_1(f_0(X_7))|\sim P_0(f_0(X_7), f_1(X_8))|P_1(f_2(X_7, X_8))$

6. $P_1(f_3(X_9, X_{10}))|P_0(X_9, X_{10})$

7. $P_0(X_{11}, f_3(X_{11}, X_{12}))|P_0(X_{12}, f_3(X_{11}, X_{12}))|P_0(X_{11}, X_{12})$

8. $\sim P_0(X_{13}, f_3(X_{13}, X_{14}))|\sim P_0(X_{14}, f_3(X_{13}, X_{14}))|P_0(X_{13}, X_{14})$

9. $P_1(f_0(c_0))$

10. $P_1(f_0(c_1))$

11. $P_0(f_0(c_0), f_1(c_1))$

12. $\sim P_1(f_2(c_0, c_1))$

The following is the SHB without using any heuristics:

$SHB_a = \{P_0(H_0, H_0), P_0(H_1, H_1), P_0(H_2, H_2), P_0(H_3, H_4), P_0(H_5, H_6),$
$P_0(H_7, f_3(H_7, H_8)), P_0(H_9, f_3(H_{10}, H_9)), P_0(H_{11}, H_{12}), P_0(H_{13}, H_{14}),$
$P_0(f_0(c_0), f_1(c_1)), P_0(H_{15}, H_{16}), P_0(H_{17}, H_{18}), P_0(f_0(H_{19}), f_1(H_{20})),$
$P_0(H_{21}, f_3(H_{21}, H_{22})), P_0(H_{23}, f_3(H_{24}, H_{23})), P_1(f_0(c_0)), P_1(f_0(c_1)),$
$P_1(f_0(H_{25})), P_1(f_2(c_0, c_1)), P_0(H_{26}, f_3(H_{26}, H_{26})), P_0(H_{27}, f_3(H_{27}, H_{27}))\}$.

Using SHB_a the SSTTP prover builds a closed semantic tree consisting of 1739 nodes in 0.470 sec. The following is the SHB with using the unit heuristic h1:

$SHB_u = \{P_0(H_0, H_0), P_1(f_0(c_0)), P_1(f_0(c_1)), P_0(f_0(c_0), f_1(c_1)), P_1(f_2(c_0, c_1)),$
$P_0(H_1, H_1), P_0(H_2, H_2), P_0(H_3, H_3), P_0(H_4, H_5), P_0(H_6, H_7), P_0(H_8, f_3(H_8, H_9)),$
$P_0(H_{10}, f_3(H_{11}, H_{10})), P_0(H_{12}, H_{13}), P_0(H_{14}, H_{15}), P_0(H_{16}, H_{17}), P_0(H_{18}, H_{19}),$

$P_0(f_0(H_{20}), f_1(H_{21})), P_0(H_{22}, f_3(H_{22}, H_{23})), P_0(H_{24}, f_3(H_{25}, H_{24})), P_1(f_0(H_{26})),$
$P_0(H_{27}, f_3(H_{27}, H_{27})), P_0(H_{28}, f_3(H_{28}, H_{28}))\}$.

Using SHB_u the SSTTP prover builds a closed semantic tree consisting of 11 nodes in time close to 0 sec. Figure 7.2 introduce the closed semantic tree of example 7.2.2 using the unit heuristic.

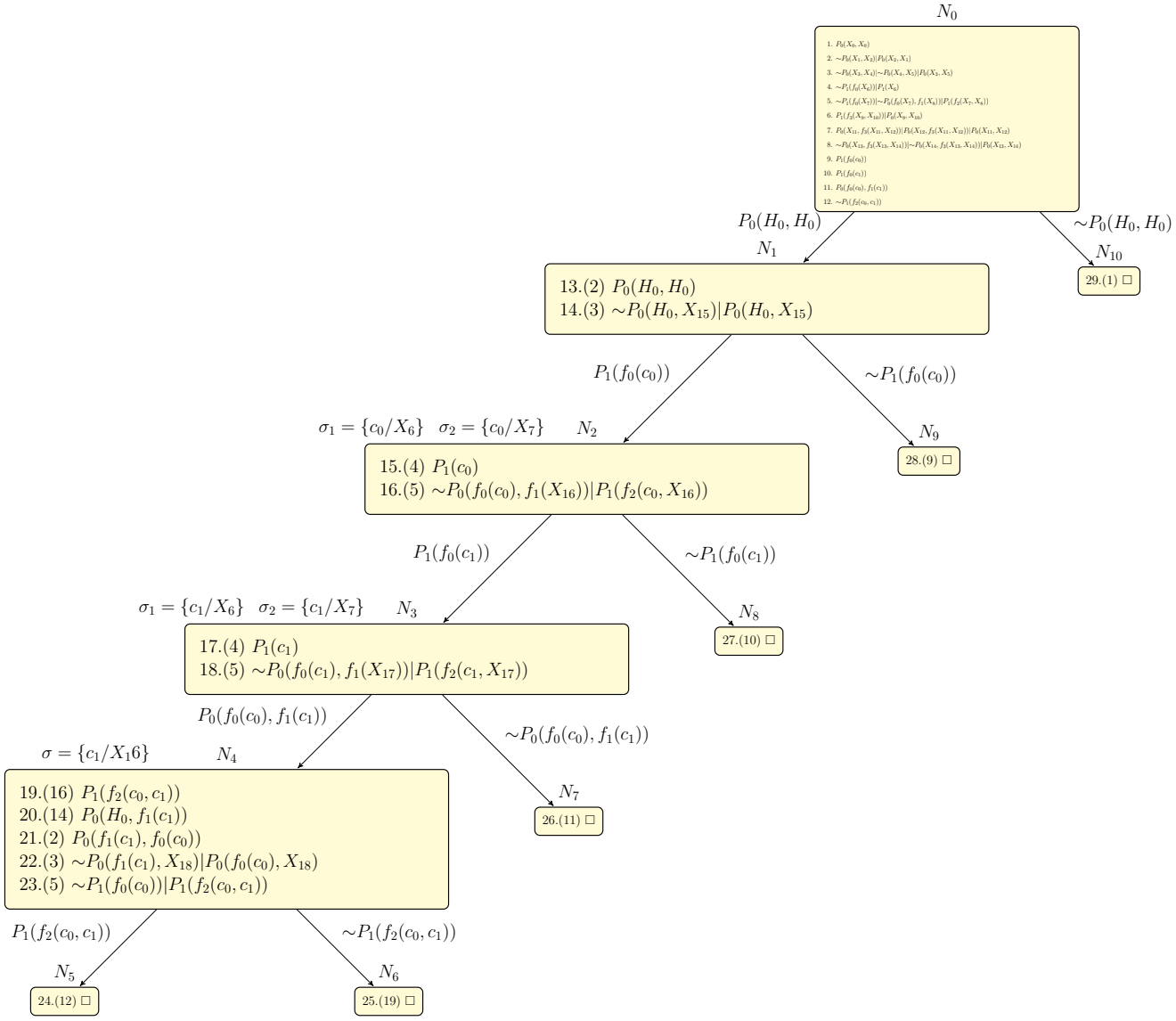

Figure 7.2: Closed semantic tree of the problem from example 7.2.2 applying the unit heuristic h1 which has the advantage that in this example the right subtree can always immediately be closed. Examples in which this is the case have linear complexity.

Observe that, using the unit heuristic allows the SSTTP to build a closed semantic tree in less time and with smaller tree size. Because with the unit heuristic in this example, *SHB* elaborate the grounded atoms first.

7.2.2 SSTTP-h2 : Degree order heuristic

In order to reduce the search spaces in first-order automated deduction, sort strategies are useful with a view to the rearrangement of clauses. It has been reported in the literature (see e.g., [53, 52, 30, 54, 50]) that such strategies can lead enormous gains in efficiency. The degree order heuristic use a selection sort technique in order to sort the clauses inside the semantic tree nodes. In spite of the inefficient time complexity that selection sort has if it is applied on a large number of clauses, it has a simple implementation and it performs well when memory is limited.

Definition (Degree order heuristic) Let $T = N_1, \ldots, N_m$ be a semantic tree. When computing the Smart Herbrand Base with the HBG-rule, the clause sets $K(N_t) = \{P_1, \ldots, P_n\}$ for $t = 1, \ldots, m$ are ordered according to degree, that is, for all $i, j \in \{1, \ldots, n\}$ with $i < j$ holds degree$(P_i) \leq$ degree(P_j). (Note that the result of degree() is a number of literals inside a given clause.)

The idea that we use is to order the clauses inside a semantic tree node by their degree (also called 'arity'). The degree of a clause is defined as a number of literals inside a given clause. SSTTP-h2 applies a sort function that orders the clauses inside each node of the semantic tree by ascending order. That is, it first lists all clauses with degree 1, which are those clauses that have one literal (unit clauses). Then, it lists all clauses with degree 2, which are clauses that have two literals. Then, it lists all clauses with degree 3, and so on. The heuristic is to assist the SSTTP prover to allocate the atoms within the *SHB*, which reduces the size of the clauses first. Accordingly, if the atom that is used to split the tree is generated from a unit clause, then at least one of the branches within the semantic

tree will close in one level as the unit heuristic is carried out. Moreover, if the atom that is used to split the tree is generating from a clause of degree 2, then the new clause that will be produced after the split of the build tree rule is a unit clause, and so on. This technique will reduce the size of the semantic tree in each level, thereby hopefully contributing to the efficiency of the SSTTP prover. Tables 7.3 and 7.4 show the results of using the degree order heuristic h2 in testing the SSTTP prover with selected problems from the TPTP library [47] (again using GPU for a maximum of 100 seconds). These selected problems are a subset of original tables from the appendix. Here, we choose only the problems that can be solved using one or both SSTTP with and without the use of the heuristics to make the comparison fair. From the tables, we can observe that 22 problems – highlighted in green – out of 58 (37.93%) were proved by SSTTP-h2 in less time and with a smaller number of nodes in the semantic tree. Moreover, (43.1%) of the results are the same. For 11 problems – highlighted in red – (18.97%) h2 did not perform as well as using SSTTP without the heuristics. Note that h2 – unlike h1 – can lead to much bigger search spaces. Because, the *SHB* atoms of h2 help reduce the size of the new clauses generating inside each nodes of the semantic tree.

	Theorem	SSTTP		SSTTP-h2	
		time	nodes number	time	nodes number
1	ANA013-2	0.000	7	0.000	7
2	ANA041-2	0.000	3	0.000	3
3	ANA042-2	0.000	3	0.000	3
4	CAT007-3	0.470	1739	0.000	15
5	COL101-2	0.000	3	0.000	3
6	COL102-2	0.190	503	0.820	1367
7	COL103-2	0.000	3	0.000	3
8	COL104-2	0.200	503	0.740	1367
9	COL105-2	0.000	5	0.000	5
10	COL109-2	2.640	1495	0.610	525
11	COL111-2	0.000	5	0.000	5
12	COL112-2	0.000	7	0.000	7
13	COL113-2	0.000	3	0.000	3
14	COL114-2	0.000	5	0.000	5
15	COL115-2	0.000	7	0.000	7
16	COL116-2	0.000	3	0.000	3
17	COL117-2	2.580	1493	0.000	19
18	COL119-2	0.000	5	0.000	5
19	COL120-2	0.000	5	0.000	5
20	COL122-2	0.000	5	0.000	5
21	COL124-2	T/O	-	0.000	47
22	COM001-1	0.090	453	0.000	67
23	COM002-1	2.260	2769	0.150	475
24	COM002-2	0.490	1809	0.010	99
25	FLD006-3	2.960	1097	M/O	-
26	GEO079-1	0.000	7	0.000	7
27	GRP123-7.003	M/O	-	19.040	9677
28	KRS004-1	0.000	5	0.000	5

Table 7.3: Test results of SSTTP prover with and without the degree order heuristic in TPTP library. (T/O means time out and M/O means memory out)

	Theorem	SSTTP		SSTTP-h2	
		time	nodes number	time	nodes number
29	LAT272-2	0.000	7	0.000	7
30	LAT273-2	M/O	-	0.000	13
31	LCL007-1	0.030	53	0.000	7
32	LCL076-2	0.000	11	0.000	15
33	LCL360-1	0.030	89	0.010	55
34	LCL355-1	M/O	-	0.010	39
35	LCL432-2	0.000	5	0.000	5
36	LCL435-2	0.000	5	0.000	5
37	LCL436-2	0.000	5	0.000	5
38	LCL437-2	0.000	5	0.000	5
39	LCL438-2	0.000	21	0.000	21
40	LCL440-2	0.000	3	0.000	3
41	LCL441-2	0.000	17	0.000	13
42	LCL446-2	0.000	15	M/O	-
43	LCL447-2	0.000	7	0.000	7
44	MGT022-1	0.000	53	0.000	31
45	MGT022-2	0.000	53	0.000	31
46	MGT030-1	M/O	-	79.180	2459
47	MGT036-3	0.220	227	0.000	23
48	MGT041-2	0.000	23	0.000	15
49	PUZ008-1	0.440	297	0.180	85
50	PUZ012-1	0.680	2553	0.000	63
51	PUZ018-1	M/O	-	0.760	187
52	PUZ035-1	0.030	357	0.010	201
53	PUZ035-2	0.040	429	0.040	497
54	PUZ035-3	1.150	1495	2.170	3787
55	PUZ035-4	1.030	1731	0.900	2021
56	PUZ035-5	0.000	89	0.080	259
57	PUZ035-6	0.000	89	0.220	433
58	PUZ035-7	0.300	477	0.290	523

Table 7.4: Continue Table 7.3, Test results of SSTTP prover with and without the degree order heuristic in TPTP library. (T/O means time out and M/O means memory out)

The motivation behind the degree order heuristic is to sort the atoms inside *SHB* according to their power to trigger resolution steps. These resolution steps can create the empty clause if the *SHB* atom resolves with a unit clause inside the clause set or they can yield a short new clause. Example 7.2.3 reorders the *SHB* according to their resolution priority that comes from applying the HBG algorithm to a sorted clause set. Observe that, using the degree order heuristic allows the SSTTP to build a close semantic tree in less time and with short tree size than SSTTP without the heuristics. Because with the degree order heuristic in this example, the tree always close one of the branches and generate shorter new clauses first in the other branch.

Example 7.2.3 *Let us consider the following problem from the Combinatory Logic domain (LCL007-1.p) in TPTP. Let S be the following set of clauses:*

1. $\sim P_0(f_0(X_0, X_1))|\sim P_0(X_0)|P_0(X_1)$
2. $P_0(f_0(f_0(X_2, X_3), f_0(X_3, X_2)))$
3. $P_0(f_0(f_0(f_0(X_4, X_5), X_6), f_0(X_4, f_0(X_5, X_6))))$
4. $\sim P_0(f_0(f_0(c_0, f_0(c_1, c_2)), f_0(f_0(c_0, c_1), c_2)))$

The following is the SHB without using any heuristics:

$SHB_a = \{P_0(f_0(f_0(H_0, H_1), f_0(H_1, H_0))), P_0(f_0(f_0(f_0(H_2, H_3), H_4), f_0(H_2, f_0(H_3, H_4)))),$
$P_0(f_0(f_0(c_0, f_0(c_1, c_2)), f_0(f_0(c_0, c_1), c_2)))\}$.

Using SHB_a the SSTTP prover builds a closed semantic tree consisting of 53 nodes in 0.030 sec. The SHB with using the degree order heuristic h2 is the same as without using the heuristics. But the semantic tree is different because of the reorder of the clause set in each node. Therefore, the instantiation of the placeholder variables is different and that constructs a closed semantic tree consisting of 7 nodes in 0 sec. Figure 7.3 introduce the closed semantic tree of example 7.2.3 using the degree order

heuristic which reorders the clause set in each node before applying the Build tree rule (that is, reorder all the clauses that come from the path of each node). During the construction of the tree, the placeholder variables were substituted. The first substitution $\sigma_1 = \{f_0(f_0(H_2, H_3), H_4)/H_0, f_0(H_2, f_0(H_3, H_4))/H_1\}$ *occurs when clause 6 in* N_2 *is produced. The second substitution* $\sigma_2 = \{c_0/H_2, c_1/H_3, c_2/H_4\}$ *occurs when clause 9 in* N_4 *is produced. Note that, to build a closed semantic tree of this example using the unit heuristic, it takes 53 nodes in 0.030 sec as in Table 7.2. The SHB using h1 is the same as SHB without the heuristic and also the same as SHB using h2. However, h2 produces a smaller closed semantic tree because of the reorder strategy.*

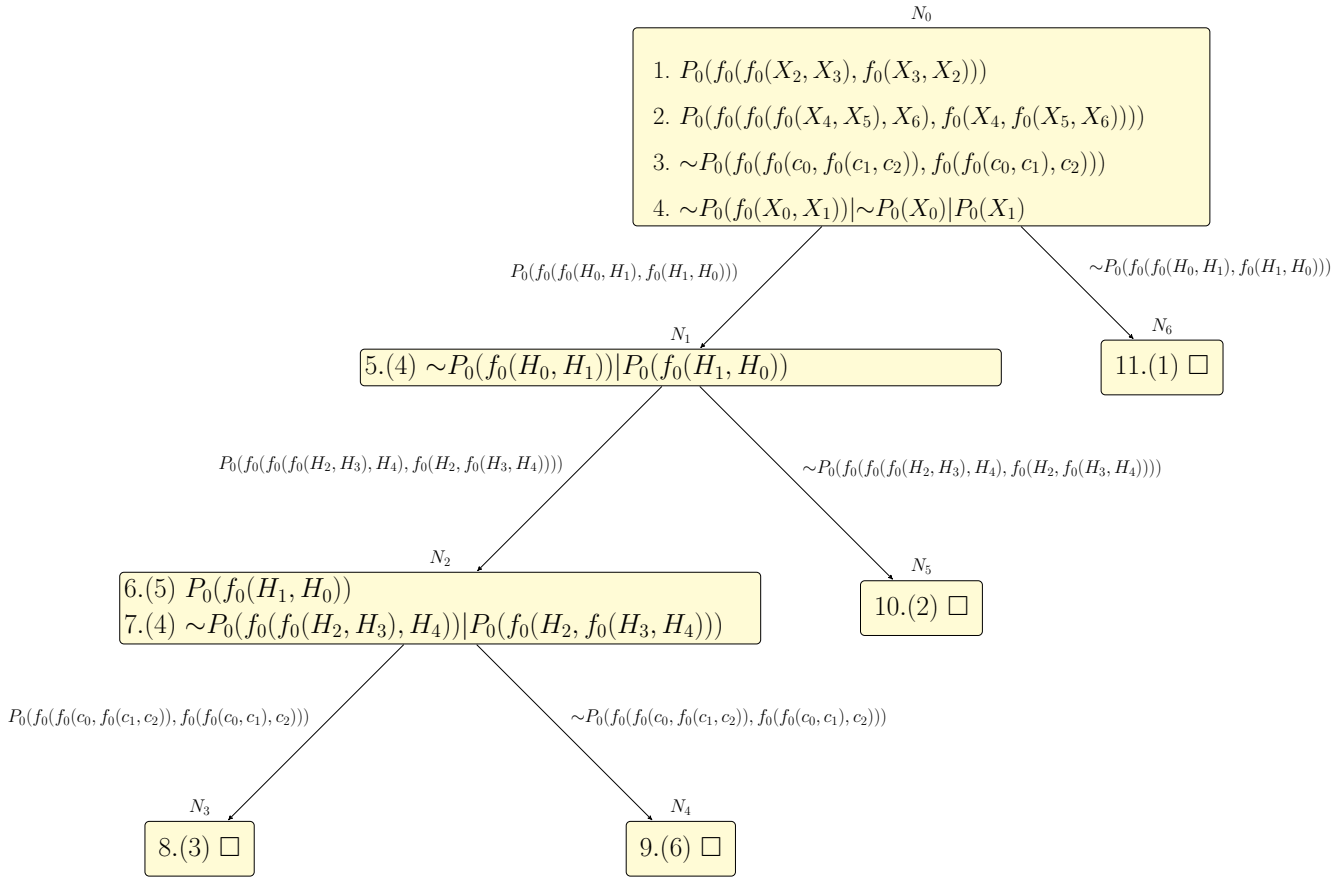

Figure 7.3: Closed semantic tree of the problem from example 7.2.3 applying the degree order heuristic h2 which has the advantage that in this example the right subtree can always immediately closed and that gives a linear complexity. The potential of h2 in this example depends on the reordering of the clauses in each node before applying the Build tree rule. h2 shows in this example a better performance than the other heuristics investigated in the dissertation.

7.2.3 SSTTP-h3 : Impact heuristic

The impact heuristic strategy is a method that should estimate how useful an *SHB* atom is in producing a closed semantic tree by calculating a so-called impact number for each atom inside the *SHB*.

Definition (Impact number) An impact number of an atom hb_i, $imp(hb_i)$, is an integer number assigned to $hb_i \in SHB$ by calculating how many literals inside the clause set match with hb_i (irrespective of variable names).

Definition (Impact heuristic) Let $SHB = (hb_1, \ldots, hb_m)$ be a Smart Herbrand Base of a given problem. If $i < j$ where $i, j \in \{1, \ldots, m\}$ then $imp(hb_i) \geq imp(hb_j)$.

The function 'impact' computes a number that represents how many clauses in each node match with atoms from the *SHB* and attach the number to each atom. Then, it chooses an atom with the maximum number to do the next split of the semantic tree. This is supposed to generate the next node that will have more new clauses that give more choice to close the semantic tree in less time and space. Tables 7.5 and 7.6 show the results of using the impact heuristic in testing the SSTTP prover with selected problems from the TPTP library [47] (again using GPU with a maximum of 100 seconds). These selected problems are a subset of original tables from the appendix. Here, we choose only the problems that both systems can solve in order to make comparison fair. From the tables, we can observe that 19 problems – highlighted in green – out of 55 (34.55%) were proved by SSTTP-h3 in less time and with a smaller number of nodes in the semantic tree. Moreover, (41.82%) of the results are the same. For 13 results – highlighted in red – (20.4%) h3 did not perform as well as when using SSTTP without the heuristics.

	Theorem	SSTTP		SSTTP-h3	
		time	nodes number	time	nodes number
1	ANA013-2	0.000	7	0.000	9
2	ANA041-2	0.000	3	0.000	3
3	ANA042-2	0.000	3	0.000	3
4	CAT007-3	0.470	1739	0.030	117
5	COL101-2	0.000	3	0.000	3
6	COL102-2	0.190	503	3.130	3695
7	COL103-2	0.000	3	0.000	3
8	COL104-2	0.200	503	1.920	1477
9	COL105-2	0.000	5	0.000	5
10	COL109-2	2.640	1495	0.000	19
11	COL111-2	0.000	5	0.000	5
12	COL112-2	0.000	7	0.000	7
13	COL113-2	0.000	3	0.000	3
14	COL114-2	0.000	5	0.000	5
15	COL115-2	0.000	7	0.000	7
16	COL116-2	0.000	3	0.000	3
17	COL117-2	2.580	1493	0.190	117
18	COL119-2	0.000	5	0.000	5
19	COL120-2	0.000	5	0.000	5
20	COL122-2	0.000	5	0.000	5
21	COM001-1	0.090	453	0.040	215
22	COM002-1	2.260	2769	3.330	1301
23	COM002-2	0.490	1809	1.800	787
24	FLD006-3	2.960	1097	M/O	-
25	GEO079-1	0.000	7	0.000	9
26	KRS004-1	0.000	5	0.000	7

Table 7.5: Test results of the SSTTP prover with and without the application of the impact heuristic using suitable problems of the TPTP library.

	Theorem	SSTTP		SSTTP-h3	
		time	nodes number	time	nodes number
27	LAT272-2	0.000	7	0.000	7
28	LCL007-1	0.030	53	0.010	33
29	LCL076-2	0.000	11	0.000	11
30	LCL360-1	0.030	89	2.450	545
31	LCL355-1	M/O	-	3.530	545
32	LCL432-2	0.000	5	0.000	5
33	LCL435-2	0.000	5	0.000	5
34	LCL436-2	0.000	5	0.000	5
35	LCL437-2	0.000	5	0.000	5
36	LCL438-2	0.000	21	0.000	27
37	LCL439-2	M/O	-	13.260	2369
38	LCL440-2	0.000	3	0.000	3
39	LCL441-2	0.000	17	0.000	17
40	LCL445-2	M/O	-	0.210	333
41	LCL446-2	0.000	15	M/O	-
42	LCL447-2	0.000	7	0.000	7
43	MGT022-1	0.000	53	0.000	25
44	MGT022-2	0.000	53	0.000	25
45	MGT036-3	0.220	227	0.140	87
46	MGT041-2	0.000	23	T/O	-
47	PUZ008-1	0.440	297	0.780	277
48	PUZ012-1	0.680	2553	0.040	217
49	PUZ035-1	0.030	357	0.020	161
50	PUZ035-2	0.040	429	0.020	167
51	PUZ035-3	1.150	1495	0.500	611
52	PUZ035-4	1.030	1731	0.430	579
53	PUZ035-5	0.000	89	0.200	445
54	PUZ035-6	0.000	89	0.120	241
55	PUZ035-7	0.300	477	1.550	1591

Table 7.6: Continuation of Table 7.5: Test results of the SSTTP prover with and without the application of the impact heuristic using suitable problems of the TPTP library.

Using the impact heuristic to search for a proof helps to generate an effective *SHB* depending on the input problem. The power of this heuristic is to reorder the *SHB* atoms at each node in the semantic tree before executing Build tree rule. This heuristic works better than SSTTP without the heuristics in the following category: Category Theory, Combinatory Logic, Computing Theory, Management (Organisation Theory), Puzzles as shown in tables 7.5 and 7.6. Unfortunately, the impact heuristic is not performing better than the unit heuristic and the degree order heuristic with some problems as in the following example 7.2.4.

Example 7.2.4 *Let us consider the problem from the previous example 7.2.2. The following is SHB after using the impact heuristic:*

$SHB_i = \{P_1(f_0(H_{25})), P_0(H_1, H_1), P_0(H_2, H_2), P_0(H_3, H_4), P_0(H_5, H_6),$
$P_0(H_7, f_3(H_7, H_8)), P_0(H_9, f_3(H_{10}, H_9)), P_0(H_{11}, H_{12}), P_0(H_{13}, H_{14}),$
$P_0(f_0(c_0), f_1(c_1)), P_0(H_{15}, H_{16}), P_0(H_{17}, H_{18}), P_0(f_0(H_{19}), f_1(H_{20})),$
$P_0(H_{21}, f_3(H_{21}, H_{22})), P_0(H_{23}, f_3(H_{24}, H_{23})), P_1(f_0(c_0)), P_1(f_0(c_1)),$
$P_0(H_0, H_0), P_1(f_2(c_0, c_1)), P_0(H_{26}, f_3(H_{26}, H_{26})), P_0(H_{27}, f_3(H_{27}, H_{27}))\}$.

Using SHB_i builds a close semantic tree consist of 117 nodes in 0.020 sec. This results is better than using SSTTP without the heuristics but it is not better than the unit heuristic in example 7.2.2 and the degree order heuristic in example 7.2.3.

On the other hand, the affect of the impact heuristic appears in solving other problems such as (COL109-2.p) from TPTP library. The examples 7.2.5 will illustrate how the impact heuristic performing better than the unit heuristic and the degree order heuristic in solving (COL109-2.p).

Example 7.2.5 *Let us consider the following problem from the Combinatory Logic domain (COL109-2.p) in TPTP. Let S be the following set of clauses:*

1. $P_0(f_0(c_0, c_1, c_2, c_2), c_3, f_1(c_2, c_2))$

2. $P_0(f_0(c_4, c_5, c_2, c_2), c_3, f_1(c_2, c_2))$

3. $P_0(f_0(c_6, c_7, c_2, c_2), c_3, f_1(c_2, c_2))$

4. $\sim P_0(f_0(f_2(f_2(c_4, c_0), f_2(c_6, c_0)), X_0, c_2, c_2), c_3, f_1(c_2, c_2))|$
   $\sim P_0(f_0(f_2(f_2(f_2(c_8, c_5), c_7), c_1), X_0, c_2, c_2), c_3, f_1(c_2, c_2))$

5. $P_0(f_0(f_2(f_2(f_2(c_8, X_1), X_2), X_3), f_2(f_2(X_1, X_3), f_2(X_2, X_3)), c_2, c_2), c_3, f_1(c_2, c_2))$

6. $\sim P_0(f_0(X_4, X_5, c_2, c_2), c_3, f_1(c_2, c_2))|\sim P_0(f_0(X_6, X_7, c_2, c_2), c_3, f_1(c_2, c_2))|$
   $P_0(f_0(f_2(X_6, X_4), f_2(X_7, X_5), c_2, c_2), c_3, f_1(c_2, c_2))$

The Smart Herbrand Base from first generation consist of 7 atoms:

$SHB = \{P_0(f_0(c_0, c_1, c_2, c_2), c_3, f_1(c_2, c_2)), P_0(f_0(c_4, c_5, c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(c_6, c_7, c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(f_2(f_2(c_4, c_0), f_2(c_6, c_0)), f_2(H_0, H_1), c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(f_2(f_2(f_2(c_8, c_5), c_7), c_1), f_2(f_2(c_5, c_1), f_2(c_7, c_1)), c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(f_2(f_2(f_2(c_8, c_5), c_7), c_1), f_2(H_2, H_3), c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(f_2(f_2(f_2(c_8, H_4), H_5), H_6), f_2(f_2(H_4, H_6), f_2(H_5, H_6)), c_2, c_2), c_3, f_1(c_2, c_2)), \dots\}$.

The Smart Herbrand Base from second generation consist of 24 atoms. Using both generation of SHB, SSTTP without the heuristics builds a close semantic tree consisting of 1495 nodes in 2.64 sec.

After computing the impact number of each atom inside SHB at each node during the build of the semantic tree. The order of the atoms that give us the proof after two generations of SHB are:

$SHB_i = \{P_0(f_0(c_0, c_1, c_2, c_2), c_3, f_1(c_2, c_2)), P_0(f_0(c_4, c_5, c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(c_6, c_7, c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(f_2(f_2(f_2(c_8, H_4), H_5), H_6), f_2(f_2(H_4, H_6), f_2(H_5, H_6)), c_2, c_2), c_3, f_1(c_2, c_2)),$
$P_0(f_0(f_2(f_2(f_2(c_8, c_5), c_7), c_1), f_2(f_2(c_5, c_1), f_2(c_7, c_1)), c_2, c_2), c_3, f_1(c_2, c_2)),$

$P_0(f_0(f_2(f_2(f_2(c_8, c_5), c_7), c_1), f_2(H_2, H_3), c_2, c_2), c_3, f_1(c_2, c_2)),$

$P_0(f_0(f_2(f_2(c_4, c_0), f_2(c_6, c_0)), f_2(H_0, H_1), c_2, c_2), c_3, f_1(c_2, c_2)),$

$P_0(f_0(f_2(c_6, c_0), f_2(c_7, c_1), c_2, c_2), c_3, f_1(c_2, c_2)),$

$P_0(f_0(f_2(c_4, c_0), f_2(c_5, c_1), c_2, c_2), c_3, f_1(c_2, c_2))\}.$

Using SHB_i that generated by SSTTP-h3 builds a close semantic tree consist of 19 nodes in 0 sec. And this results is better not only than SSTTP without the heuristics (that, give us 1495 nodes in 2.640 sec.) but also than SSTTP-h1 (that, give us 1101 nodes in 1.620 sec.) and SSTTP-h2 (that, give us 525 nodes in 0.610 sec.).

7.2.4 SSTTP-h4 : Meta duplicate elimination heuristic

In this section, we try to enhance the SSTTP prover environment by developing a heuristic that deals with the placeholder variables that occur within the *SHB* in order to eliminate redundancy. This heuristic is referred to as the meta duplicate elimination heuristic.

Definition (Meta duplicate elimination heuristic) Let $SHB = (hb_1, \ldots, hb_m)$ be a Smart Herbrand Base of a given problem. If hb_i has an empty placeholder variable (that is, not instantiated with a ground term yet) and hb_i is equivalent to hb_j (i.e., hb_j has an empty placeholder variable as well) and $i < j$ where $i, j \in \{1, \ldots, m\}$ then delete hb_j from *SHB*.

The idea of this heuristic is to minimizes the number of proof steps required by removing the duplicated placeholder atoms from the *SHB*. These atoms are inconvenient when they have an empty placeholder (meta) variables. When the SSTTP algorithm generates the *SHB* atoms that contain variables, it replaces these variables by placeholder variables assigned to an empty cell until they are grounded during the building of the semantic tree (that is, we postpone the instantiation of variables until information is available with what they should be instantiated). Therefore, when in the *SHB* there is a duplication of such a kind of atoms, it is redundant to use them before they are ground. According to

results of using the meta duplicate elimination heuristic h4 from tables 7.7 and 7.8, we can observe that this heuristic yields a minimization of the size of the semantic tree in many cases. Again, these results were obtained by testing the SSTTP prover with selected problems from TPTP library [47] (running the GPU computer for at most 100 seconds). These selected problems form a subset of original tables from appendix. Here, we choose only the problems that can be solved using one or both SSTTP with and without the use of the heuristics to make the comparison fair. From the tables, we can observe that 18 problems – highlighted in green – out of 55 (32.73%) were proved by SSTTP-h4 in less time and with a smaller number of nodes in the semantic tree. Moreover, 61.82% of the results are the same. For three results highlighted with red colour (5.45%) SSTTP did not perform so well using h4 as when not using the heuristics.

The motivation of introducing the meta duplicate elimination heuristic is to reduce the size of the *SHB* as much as possible, if the input problem consist of variables. In general, this method solve more problems from TPTP library than any other heuristics presented in this chapter. Moreover, using SSTTP with this heuristic is much better than using SSTTP without the heuristics. Since SSTTP-h4 failed to beat SSTTP only in three problems from the Puzzle domain in the experiments. The following example 7.2.6 shows the *SHB* and the result test of using SSTTP with the meta duplicate elimination heuristic.

Example 7.2.6 *Let us consider the problem from the previous example 7.2.2. The following is the SHB after using the meta duplicate elimination heuristic:*

$SHB_m = \{P_0(H_0, H_0), P_0(H_7, f_3(H_7, H_8)), P_0(f_0(c_0), f_1(c_1)), P_0(f_0(H_{19}), f_1(H_{20})),$
$P_1(f_0(c_0)), P_1(f_0(c_1)), P_1(f_0(H_{25})), P_1(f_2(c_0, c_1))\}$.

Using SHB_m the prover builds a closed semantic tree consisting of 31 nodes in 0 sec. While using the original SHB used in example 7.2.2 it builds a closed semantic tree consisting of 1739 nodes in 0.470 sec.

	Theorem	SSTTP		SSTTP-h4	
		time	nodes number	time	nodes number
1	ANA013-2	0.000	7	0.000	7
2	ANA041-2	0.000	3	0.000	3
3	ANA042-2	0.000	3	0.000	3
4	CAT007-3	0.470	1739	0.000	31
5	COL101-2	0.000	3	0.000	3
6	COL102-2	0.190	503	0.210	419
7	COL103-2	0.000	3	0.000	3
8	COL104-2	0.200	503	0.080	253
9	COL105-2	0.000	5	0.000	5
10	COL109-2	2.640	1495	2.660	1495
11	COL111-2	0.000	5	0.000	5
12	COL112-2	0.000	7	0.000	7
13	COL113-2	0.000	3	0.000	3
14	COL114-2	0.000	5	0.000	5
15	COL115-2	0.000	7	0.000	7
16	COL116-2	0.000	3	0.000	3
17	COL117-2	2.580	1493	2.690	1493
18	COL119-2	0.000	5	0.000	5
19	COL120-2	0.000	5	0.000	5
20	COL122-2	0.000	5	0.000	5
21	COM001-1	0.090	453	0.000	41
22	COM002-1	2.260	2769	0.140	307
23	COM002-2	0.490	1809	0.250	567
24	FLD006-3	2.960	1097	0.020	45
25	FLD010-3	M/O	-	10.610	3467
26	GEO079-1	0.000	7	0.000	7

Table 7.7: Test results of SSTTP prover with and without the meta duplicate elimination heuristic in TPTP library. (T/O means time out and M/O means memory out)

	Theorem	SSTTP		SSTTP-h4	
		time	nodes number	time	nodes number
27	KRS004-1	0.000	5	0.000	5
28	LAT272-2	0.000	7	0.000	7
29	LCL007-1	0.030	53	0.020	53
30	LCL043-1	M/O	-	5.920	1083
31	LCL076-2	0.000	11	0.000	11
32	LCL360-1	0.030	89	0.020	89
33	LCL432-2	0.000	5	0.000	5
34	LCL435-2	0.000	5	0.000	5
35	LCL436-2	0.000	5	0.000	5
36	LCL437-2	0.000	5	0.000	5
37	LCL438-2	0.000	21	0.000	17
38	LCL440-2	0.000	3	0.000	3
39	LCL441-2	0.000	17	0.000	17
40	LCL446-2	0.000	15	0.000	15
41	LCL447-2	0.000	7	0.000	7
42	MGT022-1	0.000	53	0.000	53
43	MGT022-2	0.000	53	0.000	53
44	MGT036-3	0.220	227	0.080	63
45	MGT041-2	0.000	23	0.000	23
46	NUM015-1	M/O	-	0.450	2771
47	PUZ008-1	0.440	297	T/O	-
48	PUZ012-1	0.680	2553	0.050	183
49	PUZ035-1	0.030	357	0.000	109
50	PUZ035-2	0.040	429	0.010	171
51	PUZ035-3	1.150	1495	1.730	2245
52	PUZ035-4	1.030	1731	1.490	2587
53	PUZ035-5	0.000	89	0.000	29
54	PUZ035-6	0.000	89	0.000	29
55	PUZ035-7	0.300	477	0.240	387

Table 7.8: Continue Table 7.7, Test results of SSTTP prover with and without the meta duplicate elimination heuristic in TPTP library. (T/O means time out and M/O means memory out)

7.3 Heuristics comparison

In order to gain performance, we presented in the previous sections four different heuristics. The first one (SSTTP-h1) deals with the size of the clauses. It chooses to start with the unit clauses. The second (SSTTP-h2), concentrates on the reordering of the clauses in each node before choosing the *SHB* atoms to split with. The third (SSTTP-h3), uses a mathematical technique to compute the most effective atom from the *SHB* to split with. Finally, the fourth heuristic (SSTTP-h4) deals with a different strategy to assist the SSTTP prover to solve more problems from the TPTP library. This heuristic is based on the placeholder grounding method. It helps the prover to eliminate useless atoms that are grounded by placeholder variables from the *SHB*. These atoms can be generated again by the HBG rule, if the prover needs to instantiate them to a different ground term. Each one of the heuristics performs better than the others in different domains from the TPTP library. Table 7.9 and Figure 7.4 shows the performance of the SSTTP prover with and without using the heuristics. Observe that SSTTP-h1 performs better in the Puzzle domain than the others. SSTTP-h2 is best at the problems from the Management (Organisation Theory) domain, while, SSTTP-h3 solves more problems from the Logic Calculi domain. Finally, SSTTP-h4 gains more results in the Group Theory and Number Theory domains. In total, SSTTP-h4 is of use in most of the results from the experiments. Further details for the rest of the theorems can be found in Appendix A. Since the results of the heuristics are close to each other, the experiments of making a combinations between them have not been done. However, it will be interesting to investigate this in future.

	Domain	Input-theorems	SSTTP	SSTTP-h1	SSTTP-h2	SSTTP-h3	SSTTP-h4
1	ALG	1	0	0	0	0	0
2	ANA	20	3	3	3	3	3
3	CAT	1	1	1	1	1	1
4	COL	19	16	17	17	16	16
5	COM	5	3	3	3	3	3
6	FLD	24	1	2	0	0	2
7	GEO	3	1	1	1	1	1
8	GRP	47	0	0	1	0	3
9	KRS	13	1	1	1	1	1
10	LAT	5	1	2	2	2	1
11	LCL	169	12	11	12	14	13
12	MGT	9	4	4	5	3	4
13	MSC	13	0	0	0	0	0
14	NUM	5	0	0	0	0	1
15	PLA	12	0	0	0	0	0
16	PUZ	40	9	11	10	9	8
	Total	386	52	56	56	53	57

Table 7.9: Analysis of SSTTP output in each category of TPTP library in case of the number of problems solved.

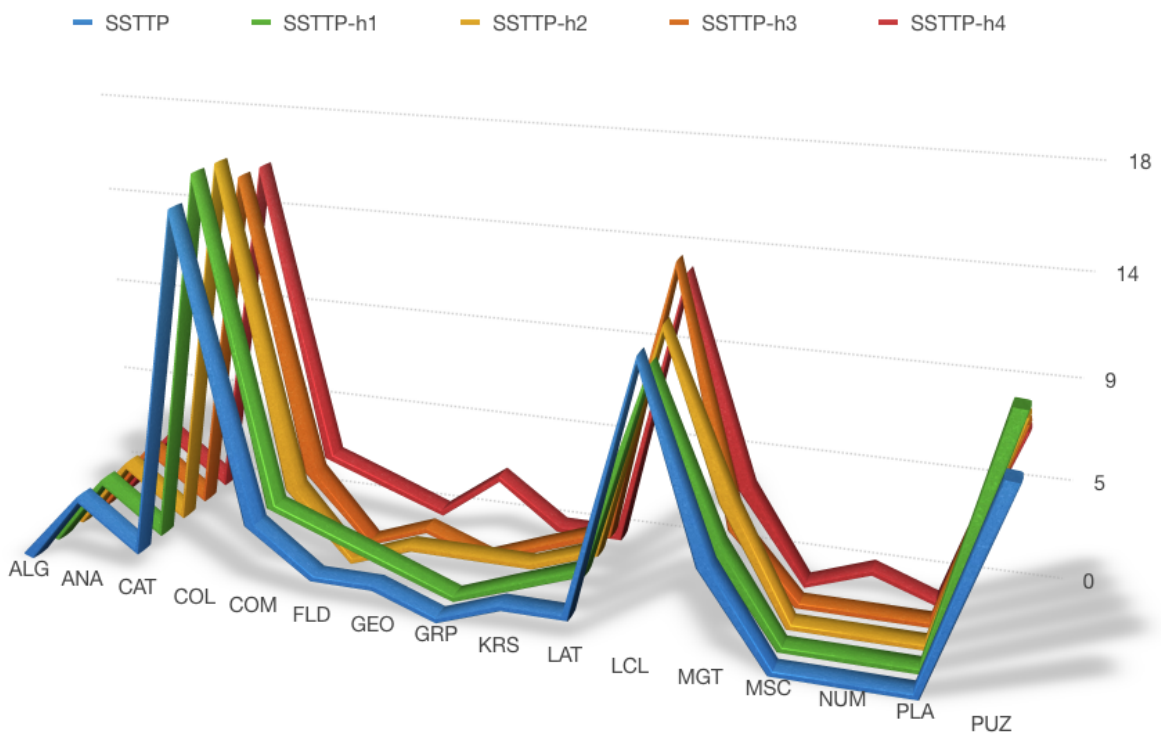

Figure 7.4: Comparison of SSTTP with and without the heuristics in each category of TPTP library. The y-axis represents the number of problems solved.

CHAPTER 8

Related Systems and Experimental Comparison

In this chapter we come back to the related works chapter. The ME calculus and its relationship with the SSTTP calculus are discussed in more detail. Then we present a comparison between the performance of the SSTTP prover and three state-of-the-art systems, which are PROVER9 (32) version 2009-11A, SPASS V 3.7, and VAMPIRE 0.6. All of the experiments were carried out on GPU with 24 CPU Intel 2.4 GHz processors with 47 GB of RAM. A period of 100 seconds was the time limit for the experiments on the CNF problems of the TPTP v5.1.0 [47] without equality.

During the implementation of the SSTTP prover we had to carefully test its correctness to ensure soundness of final system. Tests were carried out using unit testing for the crucial functionality (e.g., unification algorithm, meta-variable substitution). In addition we tested correctness of the system by running it on a set of known non-theorems verifying that it could not derive invalid proofs. For this purpose we constructed a set of 386 non-theorem problems derived from the CNF problems without equality given in the TPTP library [47] by simply negating their theorems and replacing any variables in them by new constants. Table 8.1 presents the performance of the SSTTP prover on these non-theorem problems.

	non-theorems	Model found	Time out	Memory out
Total	386	72	149	165

Table 8.1: Testing the SSTTP prover on non-theorem problems from the TPTP library. Note that "Model found" means that the prover was able to find out that the input non-theorem was not unsatisfiable, that is, for each of the non-theorems, the theorem prover either could establish that it is satisfiable, or it ran out of time, or out of memory.

8.1 Comparison to the ME calculus

Previously in Section 2.2, a summary of the ME calculus was presented. In this section, the idea of the ME calculus is described in more detail. The ME calculus is built on the DPLL procedure. At the present time, most of the SAT solvers are based on the DPLL procedure. The simplicity, polynomial space requirements, and the ability to easily integrate powerful heuristics to reduce the search space make the DPLL procedure popular. The use of such heuristics allow current SAT solvers to tackle large satisfiability problems with hundreds of thousands of variables making them notably very powerful. Therefore, automated reasoning tool developers now are using SAT solvers as back-ends to solve first-order satisfiability problems [6].

DPLL was extended, thus creating the ME calculus, which lends a number of critical ideas from FDPLL. The objective of the ME calculus is to create a Herbrand model of a particular set Φ of clauses, if such a model exists.

The ME calculus rules work on the form $A \vdash \Phi$, where A is a finite set of literals, potentially with variables and parameters referred to as a context, and Φ is the set of clauses potentially comprising variables.

The main rules of the calculus have the objective to identify when a Herbrand interpretation I_A is not a model of Φ, and accordingly complete repairs so that this status is achieved. Such repairs are concerned with the computation of most general unifiers. The progressive repair approach is otherwise recognised as evolution.

The main objective of the pseudo-literal $\sim v$ in a context Λ is to present a default truth-value for those ground literals where the value has not been established by the rest of the context. In actuality, consider a ground literal L where neither L nor $\bar{L}$ is generated by Λ. If L is found to be positive, it is false in I_Λ owing to the fact it is not generated by Λ. Should L be identified as negative, on the other hand, then it is true in I_Λ owing to the fact that it is created by $\sim v$.

It can be seen that, when Φ_0 is unsatisfiable and A_0 is merely $\{\sim v\}$, all potential evolution sequences finitely fail, meaning the calculus is complete. It is further demonstrated that, on the other hand, if all evolution sequences for $I_{\{\sim v\}}$ finitely fail, Φ_0 is then guaranteed to be unsatisfiable, thus meaning the calculus is sound [6].

DARWIN is a first simple application of the ME calculus [5], which is known to comprise three basic derivation rules, namely Split, Assert and Close, along with three optional rules, namely Resolve, Subsume and Compact. Such rules may be recognised in both Baumgartner papers [6, 5].

The proof procedure presented by DARWIN comprises five steps in its main loop: Candidate Selection, Context Unifier Computation, Backtracking and Candidate Generation. Furthermore, a number of heuristics are also involved, with the various steps and heuristics detailed in the paper of Baumgartner (Section 3, pages 7–9, and Section 4.7, pages 17–18) [5]. Prior to entering the main loop, there is the initialisation of the candidate set, with all the literals that could be added to the initial context through the implementation of Assert, which are only the unit clauses from the specified clause set.

The DPLL process underwent complete lifting to create the ME calculus for first-order clausal logic, whereas the SSTTP calculus, on the other hand, is centred on a resolution-refutation principle. Following the process for solving the previous examples in both the ME and SSTTP calculi, it can be recognised that there are a number of both similarities and differences between the approaches: similarities include the fact that both provide a

Herbrand model of a particular problem, whereas the differences include the choice of split, where SSTTP splits in line with the *SHB* , DARWIN splits depending on the candidate set present. Moreover, the split of the SSTTP provides the clause set with a new clause, which is additional information pertinent to the problem, thus meaning solving becomes much simpler, whereas DARWIN does not add any additional information, where the split adds only the literal of the split to the context. In the case of DARWIN, the candidate set is created prior to theorem proving, whereas with SSTTP, the *SHB* is created at any point throughout the theorem proving process. In addition, choosing the literals within the candidate set ultimately depends on the completion of a heuristic search; in SSTTP, the selection of literals within the *SHB* depends on the HBG rule. This relies on the resolution approach, which provides the literals to the *SHB* if such literals are a resolvent of any two clauses within the clause (problem) set.

In the next two subsection, we use two examples to exemplify the way how DARWIN and the SSTTP calculus deal with an unsatisfiable and a satisfiable problem set.

8.1.1 ME vs. SSTTP for an Unsatisfiable Problem

The next example 8.1.1 shows how DARWIN proof an unsatisfiable set. It is the same example that we used in example 5.2.3 to exemplify the SSTTP solver in section 5.2. Two different solutions are presented next, the first shows how DARWIN works with using its heuristic mechanism, the second generates a proof with the ME calculus without using any heuristics.

Example 8.1.1 $\Phi = \{P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))\}$.

- *DARWIN solution (with heuristics):* $\sim v \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

  First Assert unit clauses:

  $\sim v, P(a) \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

  $\sim v, P(a), \sim P(f(f(a))) \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

Assert $P(f(a))$ from clause 2 because it is derived from $P(a)$ and clause 2:

$\sim v, P(a), \sim P(f(f(a))), P(f(a)) \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

Close with clause 2:

$\sim v, P(a), \sim P(f(f(a))), P(f(a)) \vdash \square$

The proof is done.

- *ME-solution (without using heuristics):* $\sim v \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

  Split clause 2:

  $\sim v, \sim P(u) \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

  Close with clause 1:

  $\sim v, \sim P(a) \vdash \square$

  The right part from last Split of clause 2:

  $\sim v, P(a) \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

  Assert clause 3:

  $\sim v, P(a), \sim P(f(f(a))) \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

  Assert P(f(a)) from clause 2:

  $\sim v, P(a), \sim P(f(f(a))), P(f(a)) \vdash P(a), \sim P(X) | P(f(X)), \sim P(f(f(a)))$

  Close with clause 2:

  $\sim v, P(a), \sim P(f(f(a))), P(f(a)) \vdash \square$

  The proof is done.

For example 8.1.1, SSTTP solves the problem with three atoms from the *SHB* and two applications of the HBG rule. However, DARWIN solves the problem with three literals from the clause set and three applications of the Assert rule. This is owing to the fact that DARWIN utilises a heuristic search when selecting a unit clause. However, if DARWIN solves the problem without the use of heuristics but only through blind application of the ME rules, the proof will be longer. So, this mean that in this case SSTTP finds a shorter proof than DARWIN.

8.1.2 ME vs. SSTTP for a Satisfiable Problem

Now, let us see how both DARWIN and SSTTP work if the problem is satisfiable. The next example is from Baumgartner paper (DARWIN: A Theorem Prover for the Model Evolution Calculus) [5].

Example 8.1.2 $\Phi = \{P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|{\sim}P(X,Y),$
$\quad {\sim}P(a,a)|{\sim}Q(X,Y)|{\sim}R(f(a,Y))\}.$

- *Start the proof:*

  ${\sim}v \vdash P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|{\sim}P(X,Y),$
  $\quad {\sim}P(a,a)|{\sim}Q(X,Y)|{\sim}R(f(a,Y))$

- *The candidates for the Split rule are $P(X,a)$ from clause 1 and $Q(u,v)$ from clause 2. Split with $P(X,a)$ from clause 1 (the literal $P(X,a)$ is preferred over the other split literal, $Q(u,v)$, because it is universal, while $Q(u,v)$ is not):*

  ${\sim}v, P(X,a) \vdash P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|{\sim}P(X,Y),$
  $\quad {\sim}P(a,a)|{\sim}Q(X,Y)|{\sim}R(f(a,Y))$

- *Assert $R(f(X,a))$ from clause 3 because it is derived from $P(X,a)$ and clause 3:*

  ${\sim}v, P(X,a), R(f(X,a)) \vdash P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|{\sim}P(X,Y),$
  $\quad {\sim}P(a,a)|{\sim}Q(X,Y)|{\sim}R(f(a,Y))$

- *Subsume $P(X,a)$ from the context with clause 1:*

  ${\sim}v, P(X,a), R(f(X,a)) \vdash Q(X,Y)|Q(Y,X), R(f(X,Y))|{\sim}P(X,Y),$
  $\quad {\sim}P(a,a)|{\sim}Q(X,Y)|{\sim}R(f(a,Y))$

- *Resolve $P(X,a)$ from the context with the first literal from clause 4:*

  ${\sim}v, P(X,a), R(f(X,a)) \vdash Q(X,Y)|Q(Y,X), R(f(X,Y))|{\sim}P(X,Y),$
  $\quad {\sim}Q(X,Y)|{\sim}R(f(a,Y))$

- *Assert* $\sim Q(X,a)$ *from clause 4 because it is derived from* $R(f(X,a))$ *and clause 4:*
  $\sim v, P(X,a), R(f(X,a)), \sim Q(X,a) \vdash Q(X,Y)|Q(Y,X), R(f(X,Y))|\sim P(X,Y),$
  $\sim Q(X,Y)|\sim R(f(a,Y))$

- *Close with clause 2:*
  $\sim v, P(X,a), R(f(X,a)), \sim Q(X,a) \vdash \square$

- *The right part from last Split with* $P(X,a)$ *from clause 1 (use the complement of the Skolemized version of* $P(X,a)$ *, say,* $\sim P(c,a)$*):*
  $\sim v, \sim P(c,a) \vdash P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|\sim P(X,Y),$
  $\sim P(a,a)|\sim Q(X,Y)|\sim R(f(a,Y))$

- *Assert* $S(a)$ *because it is derived from* $\sim P(c,a)$ *and clause 1:*
  $\sim v, \sim P(c,a), S(a) \vdash P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|\sim P(X,Y),$
  $\sim P(a,a)|\sim Q(X,Y)|\sim R(f(a,Y))$

- *Split with* $Q(u,v)$ *from clause 2:*
  $\sim v, \sim P(c,a), S(a), Q(u,v) \vdash P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|\sim P(X,Y),$
  $\sim P(a,a)|\sim Q(X,Y)|\sim R(f(a,Y))$

- *Owing to the fact that no more candidates can be established, the procedure terminates, and subsequently returns the context* $\{\sim v, \sim P(c,a), S(a), Q(u,v)\}$ *to highlight the satisfiability of the clause set. This means the proof is complete.*

In this example, DARWIN shows that the problem is satisfiable. Also, SSTTP shows that the problem is satisfiable as well in the next example 8.1.3.

Example 8.1.3 $\Phi = \{P(X,a)|S(a), Q(X,Y)|Q(Y,X), R(f(X,Y))|\sim P(X,Y),$
$\sim P(a,a)|\sim Q(X,Y)|\sim R(f(a,Y))\}$. *Table 8.2 shows the steps of applying SSTTP calculus and how they are stopped to proof that the problem is satisfiable.*

	Clause set	Candidate set	*SHB* set
Initially the *SHB* is empty:			
	Φ	Φ	$\emptyset$
Start with HBG rule:			
	Φ	$S(a), Q(X,Y) \mid Q(Y,X), R(f(X,Y)),$ $\sim Q(X,Y) \mid \sim R(f(a,Y))$	$P(a,a)$
HBG rule again:			
	Φ	$S(a), Q(Y,X), R(f(X,Y)),$ $\sim R(f(a,Y))$	$P(a,a), Q(X,Y)$
HBG rule again:			
	Φ	$S(a), Q(Y,X)$	$P(a,a), Q(X,Y),$ $R(f(a,Y))$
We stop HBG rule because no more resolvent in the candidate clause set. Now, we start Build tree rule:			
1	Φ	$S(a), Q(Y,X)$	$P(a,a), Q(X,Y),$ $R(f(a,Y))$
1.1 (left)	$\Phi, R(f(X,a)),$ $\sim Q(a,a)$	$\Phi, R(f(X,a)),\ \sim Q(a,a)$	$Q(X,Y), R(f(a,Y))$
1.1.1 (left)	$\Phi, R(f(X,a)),$ $\sim Q(a,a), \square$	$\Phi, R(f(X,a)), \sim Q(a,a), \square$	$R(f(a,Y))$
Apply the Close rule, because there is an empty clause, to close branch 1.1.1 (left)			
1.1.2 (right)	$\Phi, R(f(X,a)),$ $\sim Q(a,a), \square$	$\Phi, R(f(X,a)), \sim Q(a,a), \square$	$R(f(a,Y))$
Apply the Close rule, because there is an empty clause, to close branch 1.1.2 (right)			
1.2 (right)	$\Phi, S(a)$	$\Phi, S(a)$	$Q(X,Y), R(f(a,Y))$
1.2.1 (left)	$\Phi, S(a),$ $\sim P(a,a) \mid R(f(a,a))$	$\Phi, S(a), \sim P(a,a) \mid R(f(a,a))$	$R(f(a,Y))$
1.2.1.1 (left)	$\Phi, S(a),$ $\sim P(a,a) \mid R(f(a,a)),$ $\sim P(a,a) \mid \sim Q(X,Y),$ $\sim P(a,a)$	$\Phi, S(a), \sim P(a,a) \mid R(f(a,a)),$ $\sim P(a,a) \mid \sim Q(X,Y), \sim P(a,a)$	$\emptyset$
Because *SHB* is empty, apply HBG rule to the candidate set. However, it will not generate any new atoms because there is no more resolvent in the candidate set. So, the proof is done and the clause set is satisfiable.			

Table 8.2: Applying the SSTTP calculus to the clause set of example 8.1.3.

DARWIN proves that the clause set is satisfiable and also SSTTP, the HBG will stop generating any atoms in the *SHB* as can be seen in example 8.1.3. The examples show that SSTTP executes only three HBG rules and four Build tree rules before it stops, because no more resolution steps occur to generate *SHB* atoms. This concludes that the problem is satisfiable. DARWIN solves the problem by executing nine rules. So, this mean that in this case SSTTP finds the result in fewer steps than DARWIN.

8.2 Experimental Comparison of SSTTP to State-of-the-Art Provers

The performance of the SSTTP on the applicable TPTP problems has been reported in Chapter 7, with an overview of the results provided in Table 7.9. Moreover, Table 8.3 and Figures 8.1, 8.2 summarise the number of problems solved by each of the systems: SSTTP-h4, PROVER9, SPASS, VAMPIRE (within 100s on GPU). It compares the SSTTP program used with the h4 strategy to three state-of-the-art provers. The results show that no prover has proved all the TPTP theorems. The highest number of proofs were found by the PROVER9 with 281 solved problems, whereas the VAMPIRE proved 255 and the SPASS proved 253, whilst the SSTTP-h4 proved 57 theorems. The decision of choosing the SSTTP-h4 to compare with, is based on the analysis of the outcomes of table 7.9, where better results can be identified when utilising the SSTTP-h4 on the TPTP rather than the other SSTTP heuristics. Moreover, it can also be seen that, through the use of the meta duplicate elimination heuristic, an additional 5 theorems were solved by the SSTTP prover. In addition, various theorems that could not be solved using the SSTTP – specifically in the categories NUM and GRP – were solved by using h4. This provides confirmation of the importance of the meta duplicate elimination heuristic filtering the Smart Herbrand Base when creating a semantic tree provers. Accordingly, the view is tested that the SSTTP prover with h4 heuristic can – although it cannot prove

all TPTP input theorems – be faster than other state-of-the-art provers in searching for the proof of some of the theorems. This could indeed be established as summarised in Tables 8.4 and 8.5. Note that the green cells in the tables show that the SSTTP-h4 can perform better than at least one of the other systems. Specially with the PROVER9, the SSTTP-h4 needs in 61.4% of the results around the same time. Approximately, 73.68% of the results were proved by the SSTTP-h4 in less time than at least one of the other provers. On the other hand, only for 16 results out of 57 (28.07%) – highlighted in red – the SSTTP-h4 performed worse than all the other provers. Further details of the rest of the theorems are in Appendix B. The tables in Appendix B provide details of the names of each TPTP theorem (Column 2), whilst the results of proving the TPTP through the application of the theorem provers can be seen in columns 3-6. These columns indicate whether or not a proof has been established for each of the theorems and the program's execution time (in seconds) to establish a proof. Importantly, the search process is terminated when the size of the proof exceeds the memory size, or otherwise when the 100s time limit is reached.

In conclusion, the SSTTP prover could not solve more than the other three state-of-the-art theorem provers. However, for most of the problems proofs are found faster with SSTTP than with the other systems, specially when compared with SPASS and VAMPIRE. Note that our experiments are done by executing the SSTTP prover with the heuristic h4. The experiments show that SSTTP performs better in the categories CAT, COL, COM, GEO from the TPTP library than in other categories. In order to make the comparison more substantial the results could be enhanced by adding more techniques for variable instantiation and dealing with equality so that the domain of the problem test set is enlarged. This is left to future work.

	Domain	Input-theorems	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	ALG	1	0	1	1	1
2	ANA	20	3	9	10	10
3	CAT	1	1	1	1	1
4	COL	19	16	19	19	19
5	COM	5	3	4	5	5
6	FLD	24	2	12	17	16
7	GEO	3	1	1	2	2
8	GRP	47	3	29	29	29
9	KRS	13	1	8	8	8
10	LAT	5	1	5	5	5
11	LCL	169	13	137	104	105
12	MGT	9	4	9	9	9
13	MSC	13	0	7	8	9
14	NUM	5	1	4	4	5
15	PLA	12	0	7	5	4
16	PUZ	40	8	28	26	27
	Total	386	57	281	253	255

Table 8.3: Analysis of the SSTTP-h4 and the state-of-the-art provers output from the TPTP library.

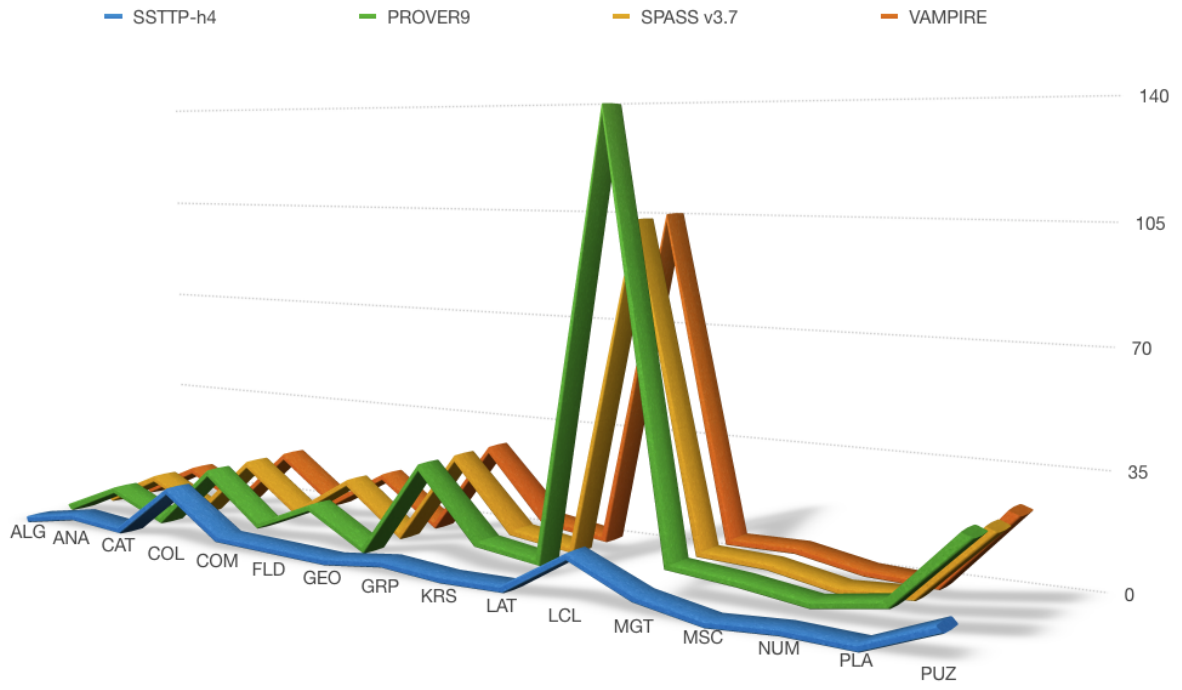

Figure 8.1: Comparison of the SSTTP-h4 and three state-of-the-art theorem provers using examples from the TPTP library. The y-axis represents the number of problems solved.

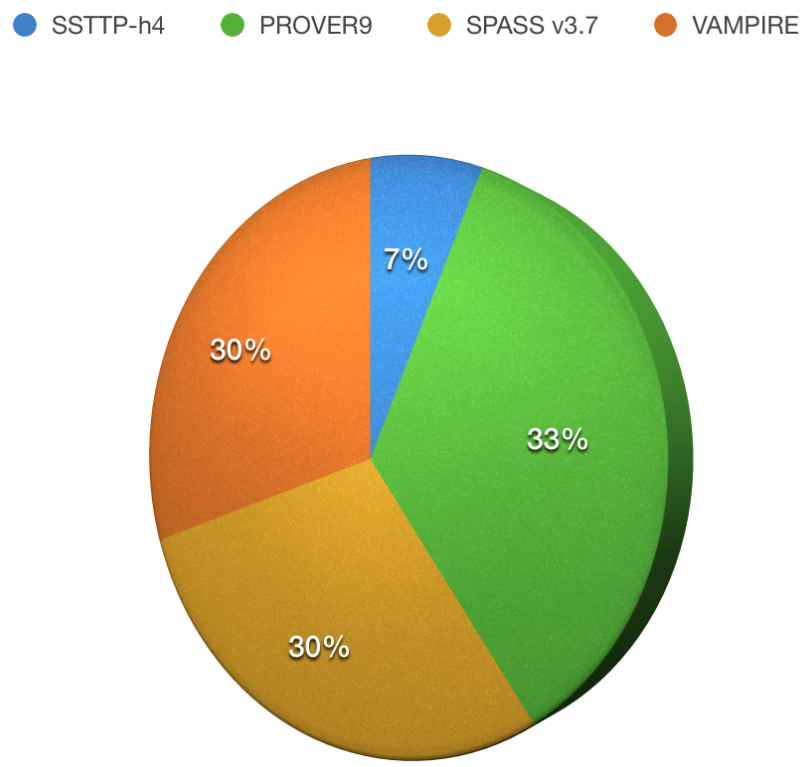

Figure 8.2: Comparing the SSTTP-h4 with three state-of-the-art provers in case of total solved problems from the TPTP library.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	ANA013-2	0.000	0.000	0.020	0.180
2	ANA041-2	0.000	0.000	0.020	0.005
3	ANA042-2	0.000	0.000	0.020	0.004
4	CAT007-3	0.000	0.000	0.010	0.070
5	COL101-2	0.000	0.000	0.020	0.120
6	COL102-2	0.210	0.000	0.020	0.101
7	COL103-2	0.000	0.000	0.010	0.120
8	COL104-2	0.080	0.000	0.020	0.122
9	COL105-2	0.000	0.000	0.020	0.120
10	COL109-2	2.660	0.000	0.010	0.061
11	COL111-2	0.000	0.000	0.010	0.080
12	COL112-2	0.000	0.000	0.020	0.160
13	COL113-2	0.000	0.000	0.020	0.120
14	COL114-2	0.000	0.000	0.010	0.111
15	COL115-2	0.000	0.000	0.010	0.081
16	COL116-2	0.000	0.000	0.010	0.030
17	COL117-2	2.690	0.000	0.010	0.161
18	COL119-2	0.000	0.000	0.010	0.030
19	COL120-2	0.000	0.000	0.010	0.080
20	COL122-2	0.000	0.000	0.010	0.004
21	COM001-1	0.000	0.000	0.010	0.031
22	COM002-1	0.140	0.000	0.010	0.031
23	COM002-2	0.250	0.000	0.010	0.008
24	FLD006-3	0.020	0.000	0.010	0.005
25	FLD010-3	10.610	0.000	0.020	0.062
26	GEO079-1	0.000	T/O	0.020	0.160
27	GRP123-1.003	3.140	0.000	0.030	0.100
28	GRP123-4.003	1.450	0.000	0.030	0.264
29	GRP125-4.003	6.520	0.000	0.010	0.270

Table 8.4: Analysis of the SSTTP and the state-of-the-art provers output in each category of the TPTP library.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
30	KRS004-1	0.000	0.000	0.010	0.090
31	LAT272-2	0.000	0.000	0.030	0.081
32	LCL007-1	0.020	0.000	0.020	0.171
33	LCL043-1	5.920	0.000	0.040	0.042
34	LCL076-2	0.000	0.000	0.090	0.030
35	LCL360-1	0.020	0.000	0.010	0.030
36	LCL432-2	0.000	0.000	0.020	0.110
37	LCL435-2	0.000	0.000	0.020	0.140
38	LCL436-2	0.000	0.000	0.030	0.310
39	LCL437-2	0.000	0.000	0.020	0.140
40	LCL438-2	0.000	0.000	0.010	0.100
41	LCL440-2	0.000	0.000	0.010	0.110
42	LCL441-2	0.000	0.000	0.010	0.240
43	LCL446-2	0.000	0.000	0.020	0.081
44	LCL447-2	0.000	0.000	0.020	0.090
45	MGT022-1	0.000	0.000	0.020	0.181
46	MGT022-2	0.000	0.000	0.030	0.421
47	MGT036-3	0.080	0.000	0.010	0.040
48	MGT041-2	0.000	0.000	0.010	0.240
49	NUM015-1	0.450	0.000	0.020	0.142
50	PUZ012-1	0.050	0.000	0.330	0.186
51	PUZ035-1	0.000	0.000	0.100	0.221
52	PUZ035-2	0.010	0.000	0.090	0.181
53	PUZ035-3	1.730	0.000	0.110	0.006
54	PUZ035-4	1.490	0.000	0.060	0.101
55	PUZ035-5	0.000	0.000	0.010	0.051
56	PUZ035-6	0.000	0.000	0.010	0.190
57	PUZ035-7	0.240	0.000	0.020	0.111

Table 8.5: Continue Table 8.4, Analysis of the SSTTP and the state-of-the-art provers output in each category of the TPTP library.

CHAPTER 9

Conclusion and Future Work

This dissertation has described the SSTTP theorem prover in detail, in particular it has described its algorithm, the data structures used, and a proof procedure. The main motivation behind the development of the SSTTP calculus was to explore the potential to use the Herbrand procedure to create an efficient theorem prover. This objective has been fulfilled to some extent, with the present version of SSTTP applying a semantic tree generator centred on proving theorems in CNF format from the TPTP library without equality. In the evaluation of the performance of the SSTTP prover compared to other modern, highly efficient provers, it is important to consider that the SSTTP calculus is a relatively new development whereas the other systems are long established. A significant degree of knowledge has been developed to combine the Herbrand procedure with binary resolution. More specifically, SSTTP is seen to perform well in regard to clause set problems. In regard to testing, SSTTP tests can be carried out at a dedicated link (`http://www.cs.bham.ac.uk/~nas991/SSTTP.html`). Importantly, subsequent developments can focus on an improved application of low-level implementation structures, such as clauses, literals, substitutions and terms, as well as on improved memory management (making use of term indexing). Furthermore a more sophisticated approach for handing

variables instantiation could improve the performance of SSTTP.

9.1 Concluding Remarks

This dissertation has examined a semantic tree approach to automated theorem proving. Throughout the dissertation, a new calculus has been introduced with the aim of providing atoms for generating semantic trees in a more efficient way than just taking them from the standard Herbrand base. Importantly, it was demonstrated that a set of theorems from the TPTP library can be proved this way. Moreover, this thesis has demonstrated how a combination of binary resolution and semantic trees generation can be used for developing of a Smart Semantic Tree Theorem Prover. The approach focuses on semantic trees, which play only a minor role currently in automated theorem proving. The contributions of this dissertation are:

1. We have described in details in Chapter 5 the HBG algorithm that successfully generates *SHB* atoms. These atoms help in building close semantic trees from a specified unsatisfiable clause set through the application of the SSTTP calculus rules.

2. The soundness and the completeness of the SSTTP calculus that we have created, have been proved in Chapter 6.

3. We introduced different variable instantiation approaches, and have provided key examples for creating semantic trees through each approach in Section 5.3. The first approach is based on the canonical order of the *HU* but with restriction depending on the number of *SHB* generations. The second approach, that we adopted in SSTTP prover, uses placeholder variables. These variables are substituted globally during the execution of the SSTTP calculus that builds the closed semantic tree. This approach generates semantic tree proofs faster than the first approach.

4. We discussed and studied various heuristics centred on enhancing the overall practicality of generating semantic trees for unsatisfiability proofs in Chapter 7. The approaches are the degree order heuristic, the impact heuristic, the meta duplicate elimination heuristic, and the unit heuristic. Such approaches were applied to the semantic tree prover SSTTP and accordingly tested on TPTP library theorems.

5. We have considered in Chapter 8 the creation of semantic trees as an alternative approach to deriving the unsatisfiability of clause sets, contrasting SSTTP with DARWIN. Moreover, this chapter has presented experiments which compare the SSTTP prover against three state-of-the-art theorem provers using all suitable problems from the TPTP library. As a final remark, the SSTTP prover was not as successful as originally hoped, nonetheless it has provided shorter proofs within a shorter time scale for some of the problems under consideration.

9.2 Future Works

In this section, a number of suggestions are made how future work can look like in order to improve the performace of the semantic tree approach.

- Grounding strategies for variables (as introduced in Chapter 5) within atoms from the Herbrand base were introduced in regard to a particular set of clauses. Subsequent examination of the effects of grounding strategies on the practicality of managing variables instantiation are expected to be a valuably contribution to enhancing the generation of semantic trees for unsatisfiability proofs.

- Regardless of the value associated with search methods, the majority of research in this domain has focused on the creation of new inference systems, which are either more restrictive or more powerful than already existing ones implementation. Other

control methods and heuristics centred on eradicating or re-organising atoms in the Herbrand base also need to be taken into account.

- The SSTTP prover accepts CNF problems without TPTP library equality. Regrettably, a large number of problems from the library could not be used owing to these restrictions of the program on parsing only CNF format from the TPTP library. The SSTTP prover can be extended by embedding equality within the system: for example, through the application of paramodulation and e-resolution, the explicit utilisation of equality axioms, and resolution through equality and unification. The presentation of equality within the SSTTP can, without question, help improve the efficiency through permitting it to prove more complex theorems. This can be seen when considering that, when the axioms of reflexivity, symmetry and transitivity for equality are incorporated, any clause comprising an instance of substitution of one such clause is a duplication that can be dismissed. Therefore, dealing with the equality is the most important point in the future works to begin with.

- Finally, the approach could be extended in a way that it makes use of the high number of processors advanced computers have these days. For this it would be necessary to explore a parallel version of the semantic tree prover, with a number of different design alternatives available when parallelising the generation of such trees. Hopefully, by doing the extensions described above, the SSTTP prover will become a powerful system that can compete in the CADE ATP system competition.

APPENDIX A

SSTTP Output Details from all categories of TPTP library

Table A.1 Analysis of ALG theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	ALG002-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

Table A.2 Analysis of ANA theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	ANA001-1	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
2	ANA002-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
3	ANA002-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
4	ANA002-3	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
5	ANA002-4	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
6	ANA003-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
7	ANA003-4	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
8	ANA004-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
9	ANA004-4	M/O	-	M/O	-	M/O	-	T/O	-	M/O	-
10	ANA004-5	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
11	ANA005-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
12	ANA005-4	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
13	ANA005-5	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
14	ANA013-2	0.000	7	0.000	9	0.000	7	0.000	9	0.000	7
15	ANA025-2	M/O	-	M/O	-	T/O	-	M/O	-	T/O	-
16	ANA037-2	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
17	ANA038-2	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
18	ANA039-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
19	ANA041-2	0.000	3	0.000	3	0.000	3	0.000	3	0.000	3
20	ANA042-2	0.000	3	0.000	5	0.000	3	0.000	3	0.000	3

Table A.3 Analysis of CAT theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	CAT007-3	0.470	1739	0.000	11	0.000	15	0.030	117	0.000	31

Table A.4 Analysis of COL theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	COL101-2	0.000	3	0.000	3	0.000	3	0.000	3	0.000	3
2	COL102-2	0.190	503	0.200	503	0.820	1367	3.130	3695	0.210	419
3	COL103-2	0.000	3	0.000	3	0.000	3	0.000	3	0.000	3
4	COL104-2	0.200	503	0.190	503	0.740	1367	1.920	1477	0.080	253
5	COL105-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
6	COL109-2	2.640	1495	1.620	1101	0.610	525	0.000	19	2.660	1495
7	COL110-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
8	COL111-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
9	COL112-2	0.000	7	0.000	11	0.000	7	0.000	7	0.000	7
10	COL113-2	0.000	3	0.000	3	0.000	3	0.000	3	0.000	3
11	COL114-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
12	COL115-2	0.000	7	0.000	11	0.000	7	0.000	7	0.000	7
13	COL116-2	0.000	3	0.000	3	0.000	3	0.000	3	0.000	3
14	COL117-2	2.580	1493	0.000	19	0.000	19	0.190	117	2.690	1493
15	COL118-2	M/O	-	T/O	-	T/O	-	T/O	-	M/O	-
16	COL119-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
17	COL120-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
18	COL122-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
19	COL124-2	T/O	-	1.120	1735	0.000	47	M/O	-	M/O	-

Table A.5 Analysis of COM theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	COM001-1	0.090	453	0.010	101	0.000	67	0.040	215	0.000	41
2	COM002-1	2.260	2769	1.760	1993	0.150	475	3.330	1301	0.140	307
3	COM002-2	0.490	1809	0.020	143	0.010	99	1.800	787	0.250	567
4	COM003-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
5	COM003-2	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-

Table A.6 Analysis of FLD theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	FLD001-3	T/O	-	T/O	-	M/O	-	T/O	-	T/O	-
2	FLD002-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
3	FLD003-1	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
4	FLD004-1	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
5	FLD005-1	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
6	FLD005-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
7	FLD006-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
8	FLD006-3	2.960	1097	0.000	11	M/O	-	M/O	-	0.020	45
9	FLD007-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
10	FLD007-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
11	FLD008-1	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
12	FLD008-2	M/O	-	T/O	-	M/O	-	T/O	-	T/O	-
13	FLD008-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
14	FLD008-4	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
15	FLD009-1	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
16	FLD009-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
17	FLD010-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
18	FLD010-3	M/O	-	15.550	1377	T/O	-	T/O	-	10.610	3467
19	FLD010-5	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
20	FLD011-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
21	FLD011-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
22	FLD012-1	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
23	FLD012-2	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
24	FLD012-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-

Table A.7 Analysis of GEO theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	GEO001-4	T/O	-	T/O	-	T/O	-	M/O	-	T/O	-
2	GEO002-4	M/O	-	T/O	-	T/O	-	T/O	-	M/O	-
3	GEO079-1	0.000	7	0.000	7	0.000	7	0.000	9	0.000	7

Table A.8 Analysis of GRP theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	GRP001-5	T/O	-	M/O	-	M/O	-	M/O	-	M/O	-
2	GRP003-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
3	GRP004-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
4	GRP005-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
5	GRP006-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
6	GRP025-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
7	GRP026-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
8	GRP027-2	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
9	GRP028-1	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
10	GRP028-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
11	GRP028-4	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
12	GRP029-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
13	GRP031-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
14	GRP034-4	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
15	GRP039-6	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
16	GRP123-1.003	T/O	-	T/O	-	T/O	-	T/O	-	3.140	3009
17	GRP123-1.005	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
18	GRP123-2.003	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
19	GRP123-3.003	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
20	GRP123-3.004	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
21	GRP123-4.003	T/O	-	T/O	-	T/O	-	T/O	-	1.450	257
22	GRP123-4.004	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
23	GRP123-6.003	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
24	GRP123-6.005	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
25	GRP123-7.003	M/O	-	M/O	-	19.040	9677	T/O	-	M/O	-
26	GRP123-7.005	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
27	GRP123-8.003	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
28	GRP123-8.004	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
29	GRP123-9.003	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
30	GRP123-9.004	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
31	GRP124-1.004	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
32	GRP124-1.005	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
33	GRP124-2.005	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
34	GRP124-3.004	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
35	GRP124-3.005	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
36	GRP124-4.004	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
37	GRP124-4.005	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
38	GRP124-6.004	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
39	GRP124-6.005	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
40	GRP124-7.004	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
41	GRP124-7.005	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
42	GRP124-8.004	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
43	GRP124-9.004	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
44	GRP124-9.005	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
45	GRP125-1.003	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
46	GRP125-4.003	T/O	-	T/O	-	T/O	-	T/O	-	6.520	3931
47	GRP125-4.004	M/O	-	T/O	-	T/O	-	T/O	-	M/O	-

Table A.9 Analysis of KRS theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	KRS001-1	M/O	-	M/O	-	T/O	-	M/O	-	T/O	-
2	KRS002-1	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
3	KRS003-1	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
4	KRS004-1	0.000	5	0.000	5	0.000	5	0.000	7	0.000	5
5	KRS006-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
6	KRS007-1	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
7	KRS008-1	M/O	-	M/O	-	M/O	-	M/O	-	T/O	-
8	KRS009-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
9	KRS010-1	T/O	-	T/O	-	M/O	-	T/O	-	T/O	-
10	KRS012-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
11	KRS013-1	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
12	KRS015-1	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
13	KRS016-1	M/O	-	M/O	-	M/O	-	T/O	-	M/O	-

Table A.10 Analysis of LAT theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	LAT005-1	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-
2	LAT005-2	M/O	-	M/O	-	T/O	-	T/O	-	T/O	-
3	LAT270-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
4	LAT272-2	0.000	7	0.000	7	0.000	7	0.000	7	0.000	7
5	LAT273-2	M/O	-	0.000	13	0.000	13	0.390	453	M/O	-

Table A.11 Analysis of LCL theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	LCL001-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
2	LCL002-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
3	LCL006-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
4	LCL007-1	0.030	53	0.030	53	0.000	7	0.010	33	0.020	53
5	LCL008-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
6	LCL009-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
7	LCL011-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
8	LCL012-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
9	LCL013-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
10	LCL014-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
11	LCL016-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
12	LCL017-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
13	LCL018-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
14	LCL022-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
15	LCL023-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
16	LCL025-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
17	LCL026-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
18	LCL027-1	M/O	-	M/O	-	M/O	-	T/O	-	M/O	-
19	LCL028-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
20	LCL029-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
21	LCL030-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
22	LCL031-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
23	LCL032-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
24	LCL039-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
25	LCL040-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
26	LCL041-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
27	LCL042-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
28	LCL043-1	M/O	-	M/O	-	M/O	-	M/O	-	5.920	1083
29	LCL044-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
30	LCL045-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
31	LCL046-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
32	LCL047-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
33	LCL048-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
34	LCL049-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
35	LCL050-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
36	LCL051-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
37	LCL052-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
38	LCL053-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
39	LCL054-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
40	LCL055-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
41	LCL056-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
42	LCL057-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
43	LCL058-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
44	LCL059-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
45	LCL060-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
46	LCL061-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
47	LCL062-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
48	LCL063-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
49	LCL064-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
50	LCL064-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
51	LCL065-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
52	LCL066-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
53	LCL067-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
54	LCL068-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
55	LCL069-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
56	LCL070-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
57	LCL072-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
58	LCL073-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
59	LCL075-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
60	LCL076-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
61	LCL076-2	0.000	11	0.000	11	0.000	15	0.000	11	0.000	11
62	LCL076-3	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
63	LCL077-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
64	LCL077-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
65	LCL078-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
66	LCL079-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
67	LCL080-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
68	LCL080-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
69	LCL081-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
70	LCL082-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
71	LCL083-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
72	LCL083-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
73	LCL084-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
74	LCL084-3	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
75	LCL085-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
76	LCL089-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
77	LCL100-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
78	LCL103-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
79	LCL105-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
80	LCL106-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
81	LCL109-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
82	LCL110-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
83	LCL112-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
84	LCL115-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
85	LCL116-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
86	LCL117-1	M/O	-	M/O	-	M/O	-	T/O	-	M/O	-
87	LCL118-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
88	LCL119-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
89	LCL129-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
90	LCL130-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
91	LCL131-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
92	LCL168-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
93	LCL256-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
94	LCL257-1	M/O	-	M/O	-	M/O	-	T/O	-	M/O	-
95	LCL355-1	M/O	-	M/O	-	0.010	39	3.530	545	M/O	-
96	LCL356-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
97	LCL357-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
98	LCL358-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
99	LCL359-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
100	LCL360-1	0.030	89	0.030	89	0.010	55	2.450	545	0.020	89

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
101	LCL361-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
102	LCL362-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
103	LCL363-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
104	LCL364-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
105	LCL365-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
106	LCL366-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
107	LCL367-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
108	LCL368-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
109	LCL369-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
110	LCL370-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
111	LCL371-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
112	LCL372-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
113	LCL373-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
114	LCL374-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
115	LCL375-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
116	LCL376-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
117	LCL377-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
118	LCL378-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
119	LCL379-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
120	LCL380-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
121	LCL381-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
122	LCL382-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
123	LCL383-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
124	LCL384-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
125	LCL385-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
126	LCL386-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
127	LCL387-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
128	LCL388-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
129	LCL389-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
130	LCL390-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
131	LCL391-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
132	LCL392-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
133	LCL393-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
134	LCL394-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
135	LCL395-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
136	LCL396-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
137	LCL397-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
138	LCL398-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
139	LCL399-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
140	LCL400-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
141	LCL401-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
142	LCL402-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
143	LCL403-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
144	LCL404-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
145	LCL405-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
146	LCL414-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
147	LCL415-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
148	LCL420-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
149	LCL421-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
150	LCL425-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
151	LCL426-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
152	LCL427-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
153	LCL428-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
154	LCL429-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
155	LCL430-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
156	LCL432-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
157	LCL433-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
158	LCL435-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
159	LCL436-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
160	LCL437-2	0.000	5	0.000	5	0.000	5	0.000	5	0.000	5
161	LCL438-2	0.000	21	0.000	23	0.000	21	0.000	27	0.000	17
162	LCL439-2	M/O	-	M/O	-	M/O	-	13.260	2369	M/O	-
163	LCL440-2	0.000	3	0.000	3	0.000	3	0.000	3	0.000	3
164	LCL441-2	0.000	17	0.000	19	0.000	13	0.000	17	0.000	17
165	LCL443-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
166	LCL444-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
167	LCL445-2	M/O	-	M/O	-	M/O	-	0.210	333	M/O	-
168	LCL446-2	0.000	15	M/O	-	M/O	-	M/O	-	0.000	15
169	LCL447-2	0.000	7	0.000	7	0.000	7	0.000	7	0.000	7

Table A.12 Analysis of MGT theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	MGT001-1	M/O	-	M/O	-	T/O	-	M/O	-	T/O	-
2	MGT007-1	M/O	-	M/O	-	T/O	-	M/O	-	T/O	-
3	MGT022-1	0.000	53	0.000	31	0.000	31	0.000	25	0.000	53
4	MGT022-2	0.000	53	0.000	31	0.000	31	0.000	25	0.000	53
5	MGT028-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
6	MGT030-1	M/O	-	M/O	-	79.180	2459	M/O	-	M/O	-
7	MGT032-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
8	MGT036-3	0.220	227	0.020	81	0.000	23	0.140	87	0.080	63
9	MGT041-2	0.000	23	0.000	15	0.000	15	T/O	-	0.000	23

Table A.13 Analysis of MSC theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	MSC001-1	M/O	-	M/O	-	M/O	-	T/O	-	M/O	-
2	MSC002-2	M/O	-	T/O	-	M/O	-	T/O	-	T/O	-
3	MSC005-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
4	MSC006-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
5	MSC008-2.002	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
6	MSC015-1.005	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
7	MSC015-1.010	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
8	MSC015-1.015	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
9	MSC015-1.020	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
10	MSC015-1.022	M/O	-	T/O	-	T/O	-	M/O	-	M/O	-
11	MSC015-1.025	M/O	-	T/O	-	T/O	-	M/O	-	M/O	-
12	MSC015-1.027	M/O	-	T/O	-	T/O	-	M/O	-	M/O	-
13	MSC015-1.030	M/O	-	T/O	-	T/O	-	T/O	-	T/O	-

Table A.14 Analysis of NUM theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	NUM015-1	M/O	-	M/O	-	M/O	-	M/O	-	0.450	2771
2	NUM016-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
3	NUM016-2	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
4	NUM017-1	T/O	-	T/O	-	M/O	-	T/O	-	T/O	-
5	NUM283-1.005	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

Table A.15 Analysis of PLA theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	PLA001-1	T/O	-	M/O	-	M/O	-	T/O	-	T/O	-
2	PLA002-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
3	PLA002-2	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
4	PLA003-1	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
5	PLA031-1.001	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
6	PLA031-1.002	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
7	PLA031-1.003	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
8	PLA031-1.004	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
9	PLA031-1.005	T/O	-	T/O	-	T/O	-	T/O	-	M/O	-
10	PLA031-1.006	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
11	PLA031-1.007	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
12	PLA031-1.008	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-

Table A.16 Analysis of PUZ theorems in case of total run time and number of nodes.

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1	PUZ001-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
2	PUZ001-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
3	PUZ002-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
4	PUZ003-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
5	PUZ005-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
6	PUZ008-1	0.440	297	0.290	151	0.180	85	0.780	277	T/O	-
7	PUZ010-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
8	PUZ012-1	0.680	2553	0.010	119	0.000	63	0.040	217	0.050	183
9	PUZ017-1	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
10	PUZ018-1	M/O	-	0.900	423	0.760	187	M/O	-	M/O	-
11	PUZ018-2	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
12	PUZ019-1	M/O	-	14.210	2041	M/O	-	T/O	-	M/O	-
13	PUZ021-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
14	PUZ028-1	M/O	-	M/O	-	T/O	-	T/O	-	T/O	-
15	PUZ028-2	M/O	-	M/O	-	T/O	-	T/O	-	M/O	-
16	PUZ028-5	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
17	PUZ028-6	M/O	-	M/O	-	M/O	-	T/O	-	M/O	-
18	PUZ029-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
19	PUZ030-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
20	PUZ031-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
21	PUZ034-1.003	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
22	PUZ034-1.004	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
23	PUZ035-1	0.030	357	0.030	357	0.010	201	0.020	161	0.000	109
24	PUZ035-2	0.040	429	0.040	429	0.040	497	0.020	167	0.010	171

	Theorem	SSTTP		SSTTP-h1		SSTTP-h2		SSTTP-h3		SSTTP-h4	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
25	PUZ035-3	1.150	1495	1.170	1495	2.170	3787	0.500	611	1.730	2245
26	PUZ035-4	1.030	1731	1.050	1731	0.900	2021	0.430	579	1.490	2587
27	PUZ035-5	0.000	89	0.010	91	0.080	259	0.200	445	0.000	29
28	PUZ035-6	0.000	89	0.010	91	0.220	433	0.120	241	0.000	29
29	PUZ035-7	0.300	477	0.310	477	0.290	523	1.550	1591	0.240	387
30	PUZ036-1.005	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
31	PUZ037-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
32	PUZ037-2	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
33	PUZ037-3	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
34	PUZ047-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
35	PUZ052-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
36	PUZ053-1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
37	PUZ054-1	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
38	PUZ056-2.005	M/O	-	M/O	-	T/O	-	M/O	-	M/O	-
39	PUZ056-2.025	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-
40	PUZ056-2.027	M/O	-	M/O	-	M/O	-	M/O	-	M/O	-

APPENDIX B

Provers Output Details from all categories of TPTP library

Table B.1 Analysis of ALG theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	ALG002-1	M/O	0.000	19.400	0.047

Table B.2 Analysis of ANA theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	ANA001-1	M/O	T/O	T/O	T/O
2	ANA002-1	T/O	T/O	T/O	T/O
3	ANA002-2	T/O	T/O	T/O	T/O
4	ANA002-3	T/O	T/O	T/O	T/O
5	ANA002-4	M/O	T/O	37.410	T/O
6	ANA003-2	T/O	T/O	T/O	26.892
7	ANA003-4	M/O	1.190	0.050	0.143
8	ANA004-2	T/O	T/O	T/O	T/O
9	ANA004-4	M/O	2.130	0.080	0.533
10	ANA004-5	M/O	T/O	T/O	T/O
11	ANA005-2	T/O	T/O	T/O	T/O
12	ANA005-4	M/O	T/O	T/O	T/O
13	ANA005-5	M/O	T/O	T/O	T/O
14	ANA013-2	0.000	0.000	0.020	0.180
15	ANA025-2	T/O	0.000	0.010	0.053
16	ANA037-2	M/O	0.000	0.020	0.131
17	ANA038-2	M/O	0.000	0.010	0.102
18	ANA039-2	M/O	0.000	0.020	0.032
19	ANA041-2	0.000	0.000	0.020	0.005
20	ANA042-2	0.000	0.000	0.020	0.004

Table B.3 Analysis of CAT theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	CAT007-3	0.000	0.000	0.010	0.070

Table B.4 Analysis of COL theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	COL101-2	0.000	0.000	0.020	0.120
2	COL102-2	0.210	0.000	0.020	0.101
3	COL103-2	0.000	0.000	0.010	0.120
4	COL104-2	0.080	0.000	0.020	0.122
5	COL105-2	0.000	0.000	0.020	0.120
6	COL109-2	2.660	0.000	0.010	0.061
7	COL110-2	T/O	0.000	0.010	0.121
8	COL111-2	0.000	0.000	0.010	0.080
9	COL112-2	0.000	0.000	0.020	0.160
10	COL113-2	0.000	0.000	0.020	0.120
11	COL114-2	0.000	0.000	0.010	0.111
12	COL115-2	0.000	0.000	0.010	0.081
13	COL116-2	0.000	0.000	0.010	0.030
14	COL117-2	2.690	0.000	0.010	0.161
15	COL118-2	M/O	0.000	0.010	0.061
16	COL119-2	0.000	0.000	0.010	0.030
17	COL120-2	0.000	0.000	0.010	0.080
18	COL122-2	0.000	0.000	0.010	0.004
19	COL124-2	M/O	0.000	0.010	0.005

Table B.5 Analysis of COM theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	COM001-1	0.000	0.000	0.010	0.031
2	COM002-1	0.140	0.000	0.010	0.031
3	COM002-2	0.250	0.000	0.010	0.008
4	COM003-1	M/O	T/O	0.790	0.306
5	COM003-2	M/O	0.000	0.010	0.007

Table B.6 Analysis of FLD theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	FLD001-3	T/O	0.000	5.370	8.054
2	FLD002-3	T/O	0.000	2.040	8.808
3	FLD003-1	T/O	T/O	0.180	41.278
4	FLD004-1	T/O	T/O	0.160	41.706
5	FLD005-1	T/O	T/O	0.300	4.741
6	FLD005-3	T/O	0.000	0.450	0.320
7	FLD006-1	T/O	0.000	0.020	0.007
8	FLD006-3	0.020	0.000	0.010	0.005
9	FLD007-1	T/O	T/O	T/O	T/O
10	FLD007-3	T/O	0.000	0.080	0.078
11	FLD008-1	T/O	T/O	T/O	T/O
12	FLD008-2	T/O	T/O	T/O	T/O
13	FLD008-3	T/O	T/O	33.470	T/O
14	FLD008-4	T/O	0.000	0.350	0.723
15	FLD009-1	T/O	T/O	0.210	4.854
16	FLD009-3	T/O	0.000	0.570	0.138
17	FLD010-1	T/O	0.000	0.020	0.105
18	FLD010-3	10.610	0.000	0.020	0.062
19	FLD010-5	T/O	0.000	0.150	0.701
20	FLD011-1	T/O	T/O	T/O	T/O
21	FLD011-3	T/O	0.000	0.380	7.066
22	FLD012-1	T/O	T/O	T/O	T/O
23	FLD012-2	T/O	T/O	T/O	T/O
24	FLD012-3	T/O	T/O	T/O	T/O

Table B.7 Analysis of GEO theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	GEO001-4	T/O	T/O	T/O	T/O
2	GEO002-4	M/O	0.000	0.040	0.134
3	GEO079-1	0.000	T/O	0.020	0.160

Table B.8 Analysis of GRP theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	GRP001-5	M/O	0.000	0.020	0.063
2	GRP003-1	T/O	0.000	0.020	0.071
3	GRP004-1	T/O	0.000	0.010	0.006
4	GRP005-1	T/O	0.000	0.010	0.090
5	GRP006-1	T/O	0.000	0.010	0.110
6	GRP025-3	T/O	T/O	T/O	T/O
7	GRP026-3	T/O	T/O	T/O	T/O
8	GRP027-2	T/O	T/O	T/O	T/O
9	GRP028-1	M/O	0.000	0.030	0.151
10	GRP028-3	T/O	0.000	0.020	0.215
11	GRP028-4	T/O	0.000	0.010	0.161
12	GRP029-2	T/O	0.000	1.510	1.030
13	GRP031-2	T/O	0.000	0.030	0.045
14	GRP034-4	T/O	0.000	0.010	0.004
15	GRP039-6	T/O	0.040	0.080	0.418
16	GRP123-1.003	3.140	0.000	0.030	0.100
17	GRP123-1.005	M/O	T/O	T/O	T/O
18	GRP123-2.003	T/O	0.000	0.020	0.172
19	GRP123-3.003	M/O	0.000	0.030	0.149
20	GRP123-3.004	M/O	T/O	T/O	T/O
21	GRP123-4.003	1.450	0.000	0.030	0.264
22	GRP123-4.004	M/O	T/O	T/O	T/O
23	GRP123-6.003	M/O	0.000	0.030	0.157
24	GRP123-6.005	M/O	T/O	T/O	T/O

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
25	GRP123-7.003	M/O	0.000	0.030	0.124
26	GRP123-7.005	M/O	T/O	T/O	T/O
27	GRP123-8.003	M/O	0.000	0.020	0.174
28	GRP123-8.004	M/O	T/O	T/O	T/O
29	GRP123-9.003	M/O	0.000	0.020	0.174
30	GRP123-9.004	M/O	T/O	T/O	T/O
31	GRP124-1.004	M/O	0.000	0.040	0.247
32	GRP124-1.005	M/O	T/O	T/O	T/O
33	GRP124-2.005	M/O	T/O	T/O	T/O
34	GRP124-3.004	M/O	0.000	0.040	1.554
35	GRP124-3.005	M/O	T/O	T/O	T/O
36	GRP124-4.004	M/O	0.000	0.070	1.079
37	GRP124-4.005	M/O	T/O	T/O	T/O
38	GRP124-6.004	M/O	0.000	0.160	0.862
39	GRP124-6.005	M/O	T/O	T/O	T/O
40	GRP124-7.004	M/O	0.000	0.170	0.889
41	GRP124-7.005	M/O	T/O	T/O	T/O
42	GRP124-8.004	M/O	0.000	0.080	0.891
43	GRP124-9.004	M/O	0.000	0.150	0.929
44	GRP124-9.005	M/O	T/O	T/O	T/O
45	GRP125-1.003	M/O	0.000	0.020	0.040
46	GRP125-4.003	6.520	0.000	0.010	0.270
47	GRP125-4.004	M/O	T/O	T/O	T/O

Table B.9 Analysis of KRS theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	KRS001-1	T/O	0.000	0.020	0.006
2	KRS002-1	M/O	0.000	0.020	0.101
3	KRS003-1	M/O	0.000	0.020	0.151
4	KRS004-1	0.000	0.000	0.010	0.090
5	KRS006-1	M/O	T/O	T/O	T/O
6	KRS007-1	M/O	T/O	T/O	T/O
7	KRS008-1	T/O	T/O	T/O	T/O
8	KRS009-1	M/O	T/O	T/O	T/O
9	KRS010-1	T/O	0.000	0.030	0.184
10	KRS012-1	M/O	0.000	0.010	0.210
11	KRS013-1	M/O	0.000	0.010	0.131
12	KRS015-1	M/O	0.000	0.010	0.091
13	KRS016-1	M/O	T/O	T/O	T/O

Table B.10 Analysis of LAT theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	LAT005-1	T/O	0.000	0.460	0.567
2	LAT005-2	T/O	0.000	0.240	0.357
3	LAT270-2	T/O	0.000	0.010	0.120
4	LAT272-2	0.000	0.000	0.030	0.081
5	LAT273-2	M/O	0.000	0.020	0.080

Table B.11 Analysis of LCL theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	LCL001-1	M/O	T/O	T/O	T/O
2	LCL002-1	M/O	0.330	9.420	T/O
3	LCL006-1	M/O	0.000	4.170	25.875
4	LCL007-1	0.020	0.000	0.020	0.171
5	LCL008-1	M/O	0.000	0.010	0.202
6	LCL009-1	M/O	0.000	0.120	0.204
7	LCL011-1	M/O	0.010	2.560	2.709
8	LCL012-1	M/O	0.140	T/O	56.664
9	LCL013-1	M/O	0.000	0.020	0.005
10	LCL014-1	M/O	0.030	2.030	19.930
11	LCL016-1	M/O	0.040	T/O	34.426
12	LCL017-1	M/O	0.110	T/O	26.499
13	LCL018-1	M/O	0.060	T/O	43.960
14	LCL022-1	M/O	0.000	0.020	0.198
15	LCL023-1	M/O	0.000	0.120	0.083
16	LCL025-1	M/O	0.010	18.310	1.686
17	LCL026-1	M/O	0.020	38.010	1.826
18	LCL027-1	M/O	0.000	0.030	0.073
19	LCL028-1	M/O	T/O	T/O	37.638
20	LCL029-1	M/O	0.000	0.620	0.229
21	LCL030-1	M/O	0.180	T/O	T/O
22	LCL031-1	M/O	T/O	T/O	56.854
23	LCL032-1	M/O	T/O	T/O	T/O
24	LCL039-1	M/O	4.780	0.540	0.174
25	LCL040-1	M/O	0.090	T/O	5.680

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
26	LCL041-1	M/O	0.000	0.070	0.038
27	LCL042-1	M/O	1.670	T/O	0.317
28	LCL043-1	5.920	0.000	0.040	0.042
29	LCL044-1	M/O	0.000	0.280	0.100
30	LCL045-1	M/O	0.250	T/O	0.334
31	LCL046-1	M/O	0.000	0.070	0.101
32	LCL047-1	M/O	0.000	0.460	T/O
33	LCL048-1	M/O	0.000	0.280	T/O
34	LCL049-1	M/O	0.030	0.870	T/O
35	LCL050-1	M/O	0.030	0.620	T/O
36	LCL051-1	M/O	0.120	T/O	T/O
37	LCL052-1	M/O	0.020	5.590	7.219
38	LCL053-1	M/O	0.010	5.770	9.485
39	LCL054-1	M/O	T/O	T/O	T/O
40	LCL055-1	M/O	0.010	1.500	7.344
41	LCL056-1	M/O	0.010	1.170	7.172
42	LCL057-1	M/O	0.010	1.150	7.313
43	LCL058-1	M/O	0.220	T/O	T/O
44	LCL059-1	M/O	0.020	10.930	T/O
45	LCL060-1	M/O	0.250	T/O	T/O
46	LCL061-1	M/O	T/O	T/O	T/O
47	LCL062-1	M/O	T/O	T/O	T/O
48	LCL063-1	M/O	T/O	T/O	T/O
49	LCL064-1	M/O	0.030	36.470	19.847
50	LCL064-2	M/O	0.000	0.530	0.893

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
51	LCL065-1	M/O	0.010	0.080	0.229
52	LCL066-1	M/O	0.000	0.050	0.044
53	LCL067-1	M/O	0.070	31.750	9.735
54	LCL068-1	M/O	0.180	T/O	1.564
55	LCL069-1	M/O	0.000	0.380	0.182
56	LCL070-1	M/O	0.070	T/O	28.014
57	LCL072-1	M/O	0.010	6.960	0.271
58	LCL073-1	M/O	T/O	T/O	T/O
59	LCL075-1	M/O	0.010	0.190	0.238
60	LCL076-1	M/O	0.010	0.080	0.024
61	LCL076-2	0.000	0.000	0.090	0.030
62	LCL076-3	M/O	0.000	0.120	0.019
63	LCL077-1	M/O	0.000	0.040	0.048
64	LCL077-2	M/O	0.000	0.080	0.020
65	LCL078-1	M/O	T/O	T/O	T/O
66	LCL079-1	M/O	0.000	0.020	0.111
67	LCL080-1	M/O	0.010	7.060	20.072
68	LCL080-2	M/O	0.010	6.630	20.095
69	LCL081-1	M/O	0.000	0.020	0.187
70	LCL082-1	M/O	0.000	0.020	0.175
71	LCL083-1	M/O	0.000	0.370	0.653
72	LCL083-2	M/O	0.000	0.370	0.179
73	LCL084-2	M/O	2.010	T/O	T/O
74	LCL084-3	M/O	1.670	T/O	T/O
75	LCL085-1	M/O	0.530	T/O	T/O

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
76	LCL089-1	M/O	0.000	0.110	1.181
77	LCL100-1	M/O	T/O	T/O	T/O
78	LCL103-1	M/O	0.040	T/O	46.227
79	LCL105-1	M/O	T/O	T/O	T/O
80	LCL106-1	M/O	0.000	0.020	0.030
81	LCL109-1	M/O	T/O	T/O	T/O
82	LCL110-1	M/O	0.000	0.030	0.138
83	LCL112-1	M/O	0.000	0.030	0.075
84	LCL115-1	M/O	0.000	0.240	0.361
85	LCL116-1	M/O	0.570	T/O	33.238
86	LCL117-1	M/O	0.000	0.080	0.201
87	LCL118-1	M/O	0.000	0.290	2.283
88	LCL119-1	M/O	T/O	19.760	23.708
89	LCL129-1	M/O	1.970	T/O	T/O
90	LCL130-1	M/O	0.000	0.030	0.119
91	LCL131-1	M/O	0.020	7.110	37.713
92	LCL168-1	M/O	T/O	T/O	T/O
93	LCL256-1	M/O	0.000	1.140	7.128
94	LCL257-1	M/O	0.000	0.140	0.164
95	LCL355-1	M/O	0.000	0.030	0.030
96	LCL356-1	M/O	0.000	0.020	0.012
97	LCL357-1	M/O	0.000	0.030	0.006
98	LCL358-1	M/O	0.000	0.320	0.063
99	LCL359-1	M/O	0.000	0.030	0.132
100	LCL360-1	0.020	0.000	0.010	0.030

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
101	LCL361-1	M/O	0.000	0.020	0.141
102	LCL362-1	M/O	0.000	0.020	0.062
103	LCL363-1	M/O	0.000	0.020	0.035
104	LCL364-1	M/O	0.000	0.090	T/O
105	LCL365-1	M/O	T/O	T/O	T/O
106	LCL366-1	M/O	0.000	0.020	0.084
107	LCL367-1	M/O	0.000	0.290	T/O
108	LCL368-1	M/O	0.120	1.560	T/O
109	LCL369-1	M/O	0.110	T/O	T/O
110	LCL370-1	M/O	0.110	T/O	T/O
111	LCL371-1	M/O	0.110	T/O	T/O
112	LCL372-1	M/O	T/O	T/O	T/O
113	LCL373-1	M/O	0.210	T/O	T/O
114	LCL374-1	M/O	T/O	T/O	T/O
115	LCL375-1	M/O	T/O	T/O	T/O
116	LCL376-1	M/O	T/O	T/O	T/O
117	LCL377-1	M/O	T/O	T/O	T/O
118	LCL378-1	M/O	0.010	1.480	7.077
119	LCL379-1	M/O	0.010	1.310	7.188
120	LCL380-1	M/O	0.010	1.130	7.238
121	LCL381-1	M/O	0.010	1.530	7.089
122	LCL382-1	M/O	0.120	T/O	T/O
123	LCL383-1	M/O	T/O	T/O	T/O
124	LCL384-1	M/O	0.020	0.470	T/O
125	LCL385-1	M/O	0.170	T/O	T/O

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
126	LCL386-1	M/O	0.150	T/O	T/O
127	LCL387-1	M/O	0.120	T/O	T/O
128	LCL388-1	M/O	0.230	T/O	T/O
129	LCL389-1	M/O	0.190	T/O	T/O
130	LCL390-1	M/O	0.260	T/O	T/O
131	LCL391-1	M/O	T/O	T/O	T/O
132	LCL392-1	M/O	0.260	T/O	T/O
133	LCL393-1	M/O	T/O	T/O	T/O
134	LCL394-1	M/O	T/O	T/O	T/O
135	LCL395-1	M/O	T/O	T/O	T/O
136	LCL396-1	M/O	0.010	1.670	7.146
137	LCL397-1	M/O	0.000	0.060	0.165
138	LCL398-1	M/O	0.000	0.070	0.191
139	LCL399-1	M/O	0.010	1.120	0.212
140	LCL400-1	M/O	0.020	4.120	9.895
141	LCL401-1	M/O	0.010	4.690	T/O
142	LCL402-1	M/O	0.010	1.280	T/O
143	LCL403-1	M/O	0.200	T/O	T/O
144	LCL404-1	M/O	0.170	T/O	T/O
145	LCL405-1	M/O	0.010	1.470	10.234
146	LCL414-1	M/O	0.000	0.270	0.362
147	LCL415-1	M/O	T/O	T/O	T/O
148	LCL420-1	M/O	T/O	T/O	T/O
149	LCL421-1	M/O	T/O	T/O	T/O
150	LCL425-1	M/O	T/O	T/O	T/O

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
151	LCL426-1	M/O	T/O	T/O	T/O
152	LCL427-1	M/O	T/O	T/O	T/O
153	LCL428-1	M/O	0.000	0.020	0.181
154	LCL429-2	M/O	0.000	0.050	0.172
155	LCL430-2	M/O	0.000	0.040	0.152
156	LCL432-2	0.000	0.000	0.020	0.110
157	LCL433-2	M/O	0.000	0.020	0.314
158	LCL435-2	0.000	0.000	0.020	0.140
159	LCL436-2	0.000	0.000	0.030	0.310
160	LCL437-2	0.000	0.000	0.020	0.140
161	LCL438-2	0.000	0.000	0.010	0.100
162	LCL439-2	M/O	0.000	0.010	0.181
163	LCL440-2	0.000	0.000	0.010	0.110
164	LCL441-2	0.000	0.000	0.010	0.240
165	LCL443-2	M/O	0.000	0.010	2.974
166	LCL444-2	M/O	0.270	T/O	T/O
167	LCL445-2	M/O	0.000	0.050	0.082
168	LCL446-2	0.000	0.000	0.020	0.081
169	LCL447-2	0.000	0.000	0.020	0.090

Table B.12 Analysis of MGT theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	MGT001-1	T/O	0.000	0.060	0.097
2	MGT007-1	T/O	0.000	0.040	0.052
3	MGT022-1	0.000	0.000	0.020	0.181
4	MGT022-2	0.000	0.000	0.030	0.421
5	MGT028-1	M/O	0.000	0.010	0.173
6	MGT030-1	M/O	0.000	0.020	0.195
7	MGT032-2	M/O	0.000	0.020	0.131
8	MGT036-3	0.080	0.000	0.010	0.040
9	MGT041-2	0.000	0.000	0.010	0.240

Table B.13 Analysis of MSC theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	MSC001-1	M/O	T/O	0.020	0.258
2	MSC002-2	T/O	0.000	0.010	0.171
3	MSC005-1	M/O	0.000	0.010	0.191
4	MSC006-1	T/O	0.000	0.010	0.132
5	MSC008-2.002	T/O	0.070	0.310	10.911
6	MSC015-1.005	T/O	0.000	0.030	0.102
7	MSC015-1.010	M/O	0.000	0.030	0.168
8	MSC015-1.015	M/O	0.070	26.530	1.760
9	MSC015-1.020	M/O	T/O	T/O	61.900
10	MSC015-1.022	M/O	T/O	T/O	T/O
11	MSC015-1.025	M/O	T/O	T/O	T/O
12	MSC015-1.027	M/O	T/O	T/O	T/O
13	MSC015-1.030	T/O	T/O	T/O	T/O

Table B.14 Analysis of NUM theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	NUM015-1	0.450	0.000	0.020	0.142
2	NUM016-1	M/O	0.000	0.050	0.161
3	NUM016-2	M/O	0.000	0.030	0.260
4	NUM017-1	T/O	T/O	T/O	61.369
5	NUM283-1.005	M/O	0.000	1.500	4.753

Table B.15 Analysis of PLA theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	PLA001-1	T/O	0.000	T/O	T/O
2	PLA002-1	M/O	0.000	0.030	0.140
3	PLA002-2	M/O	0.000	0.020	0.082
4	PLA003-1	M/O	0.000	0.030	0.201
5	PLA031-1.001	T/O	0.010	25.930	T/O
6	PLA031-1.002	T/O	0.000	21.370	T/O
7	PLA031-1.003	M/O	0.020	T/O	T/O
8	PLA031-1.004	M/O	T/O	T/O	63.329
9	PLA031-1.005	M/O	T/O	T/O	T/O
10	PLA031-1.006	T/O	T/O	T/O	T/O
11	PLA031-1.007	T/O	T/O	T/O	T/O
12	PLA031-1.008	T/O	T/O	T/O	T/O

Table B.16 Analysis of PUZ theorems in case of total run time.

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
1	PUZ001-1	T/O	0.000	0.240	0.241
2	PUZ001-3	T/O	T/O	T/O	T/O
3	PUZ002-1	T/O	0.000	0.030	0.070
4	PUZ003-1	T/O	0.000	0.070	0.120
5	PUZ005-1	M/O	0.000	0.040	0.010
6	PUZ008-1	T/O	0.000	0.120	0.225
7	PUZ010-1	M/O	0.160	14.170	T/O
8	PUZ012-1	0.050	0.000	0.330	0.186
9	PUZ017-1	M/O	0.140	T/O	0.087
10	PUZ018-1	M/O	0.000	T/O	0.226
11	PUZ018-2	M/O	T/O	T/O	T/O
12	PUZ019-1	M/O	0.000	0.080	0.221
13	PUZ021-1	T/O	0.000	0.130	0.336
14	PUZ028-1	T/O	T/O	T/O	T/O
15	PUZ028-2	M/O	T/O	T/O	T/O
16	PUZ028-5	M/O	0.000	0.350	0.184
17	PUZ028-6	M/O	0.020	0.080	0.566
18	PUZ029-1	T/O	0.000	0.080	0.120
19	PUZ030-1	T/O	0.000	0.080	0.118
20	PUZ031-1	T/O	0.000	0.120	0.208
21	PUZ034-1.003	T/O	T/O	T/O	T/O
22	PUZ034-1.004	T/O	T/O	T/O	T/O
23	PUZ035-1	0.000	0.000	0.100	0.221
24	PUZ035-2	0.010	0.000	0.090	0.181

	Theorem	SSTTP-h4	PROVER9	SPASS v3.7	VAMPIRE
25	PUZ035-3	1.730	0.000	0.110	0.006
26	PUZ035-4	1.490	0.000	0.060	0.101
27	PUZ035-5	0.000	0.000	0.010	0.051
28	PUZ035-6	0.000	0.000	0.010	0.190
29	PUZ035-7	0.240	0.000	0.020	0.111
30	PUZ036-1.005	T/O	0.000	0.190	0.150
31	PUZ037-1	T/O	0.000	0.030	0.057
32	PUZ037-2	T/O	0.000	0.100	0.138
33	PUZ037-3	T/O	0.000	T/O	10.193
34	PUZ047-1	M/O	0.000	0.300	0.121
35	PUZ052-1	T/O	T/O	T/O	T/O
36	PUZ053-1	T/O	T/O	T/O	T/O
37	PUZ054-1	M/O	T/O	T/O	T/O
38	PUZ056-2.005	M/O	T/O	0.090	T/O
39	PUZ056-2.025	M/O	T/O	T/O	T/O
40	PUZ056-2.027	M/O	M/O	T/O	T/O

List of References

[1] Mohammed A Almulla. *Analysis of the use of semantic trees in automated theorem proving*. PhD thesis, McGill University, 1996. 2, 2.1, 3.1, 4

[2] Jürgen Avenhaus, Jörg Denzinger, and Matthias Fuchs. COUNT: A system for distributed equational deduction. In Jieh Hsiang, editor, *Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 397–402. Springer, 1995. 2.4.3

[3] Peter Baumgartner. FDPLL: A First-Order Davis-Putnam-Logeman-Loveland Procedure. In David McAllester, editor, *Automated Deduction - CADE-17*, volume 1831 of *Lecture Notes in Computer Science*, pages 200–219. Springer, 2000. 2, 2.2

[4] Peter Baumgartner. A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure. In Gerhard Lakemeyer and Bernhard Nebel, editors, *AI in the new Millennium*, pages 289–329. Morgan Kaufmann, 2002. 2.2

[5] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Darwin: A theorem prover for the model evolution calculus. In *Proceedings of the 1st Workshop on Empirically Successful First Order Reasoning (ESFOR'04)*. Elsevier, 2004. 2, 8.1, 8.1.2

[6] Peter Baumgartner and Cesare Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172:591 – 632, 2008. 2, 2.2, 8.1

[7] Bernhard Beckert and Rajeev Goré. Free-variable tableaux for propositional modal logics. *Studia Logica*, 69(1):59–96, 2001. 2

[8] Bernhard Beckert and Joachim Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995. 2

[9] Magnus Björk. A First Order extension of Stålmarck's method. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2005. 7

[10] Magnus Björk. First Order Stålmarck. *Journal of Automated Reasoning*, 42(1):99–122, 2009. 7

[11] R.S. Boyer, M. Kaufmann, and J.S. Moore. The Boyer-Moore theorem prover and its interactive enhancement. *Computers & Mathematics with Applications*, 29(2):27 – 62, 1995. 1

[12] Alan Bundy. The use of explicit plans to guide inductive proofs. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th CADE*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120. Springer, 1988. 1

[13] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving, 1973*. Academic Press. 2, 3.1, 3.2, 3.2.1, 3.2.1, 4.5

[14] Seungyeob Choi and Manfred Kerber. Semantic Selection for Resolution in Clause Graphs. In Bob McKay and John Slaney, editors, *AI 2002: Advances in Artificial Intelligence*, volume 2557 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2002. 2.3

[15] Jörg Denzinger, Martin Kronenburg, and Stephan Schulz. DISCOUNT - A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997. 2.4.3

[16] Jörg Denzinger and Stephan Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation*, 21:523 – 541, 1996. 2.4.3

[17] Zachary Ernst and Seth Kurtenbach. Automated Reasoning and Mathematics. In Maria Paola Bonacina and Mark E. Stickel, editors, *Toward a Procedure for Data Mining Proofs*, volume 7788 of *Lecture Notes in Computer Science*, pages 229–239. Springer, 2013. 2.4.1

[18] Jacques Herbrand. On the Consistency of Arithmetic. In *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, pages 618–628. 1931. 4

[19] Kahlil Hodgson and John Slaney. TPTP, CASC and the Development of a Semantically Guided Theorem Prover. *AI Commun.*, 15(2,3):135–146, 2002. 2.3

[20] Peter Höfner and Georg Struth. Can Refinement be Automated. *Electronic Notes in Theoretical Computer Science*, 201:197 – 222, 2008. Proceedings of the BCS-FACS Refinement Workshop (REFINE 2007). 2.4.1

[21] Mateja Jamnik, Manfred Kerber, Martin Pollet, and Christoph Benzmüller. Automatic learning of proof methods in proof planning. *Logic Journal of the IGPL*, 11(6):647 – 673, 2003. 1

[22] Choon Kyu Kim. Exploiting parallelism: Highly competitive semantic tree theorem prover. *International Journal of Computer Mathematics*, 81(9):1051–1067, 2004. 2, 2.1

[23] Konstantin Korovin. iProver – An instantiation-based theorem prover for first-order logic (system description). 5195:292–298, 2008. 2

[24] Laura Kovacs and Andrei Voronkov. First-Order Theorem Proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013. 2.4.3

[25] Patrice Lapierre. *Willow: extending Herby's semantic tree theorem-proving heuristics*. McGill University, 1999. 2, 2.1

[26] John Levine and Levine John. *Flex & Bison*. O'Reilly Media, Inc., 1st edition, 2009. 1

[27] William McCune. Prover9 and Mace4. http://www.cs.unm.edu/~mccune/prover9, 2005–2010. 1, 2, 2.4.1

[28] William McCune. Semantic Guidance for Saturation Provers. In *Proceedings of the 8th International Conference on Artificial Intelligence and Symbolic Computation*, AISC'06, pages 18–24. Springer, 2006. 2.4.1

[29] William W. Mccune. OTTER 3.0 Reference Manual and Guide. Technical report, Argonne National Laboratory, Argonne, IL, 1994. 1, 2.3, 2.4.1, 2.4.3

[30] Jose Meseguer, Joseph A. Goguen, and Gert Smolka. Order-sorted unification. *Journal of Symbolic Computation*, 8(4):383 – 413, 1989. 7.2.2

[31] Monty Newborn. *Automated Theorem Proving: Theory and Practice*. Springer, 2001. 1, 2, 2.1, 3.1, 3.1, 4.3, 4.5

[32] Allen Newell and Herbert A Simon. The logic theory machine – A complex information processing system. *Information Theory, IRE Transactions on*, 2(3):61–79, 1956. 1

[33] Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980. 7.2.1

[34] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998. 3.2.1

[35] Jens Otten. ileanTAP: An intuitionistic theorem prover. 1227:307–312, 1997. 2

[36] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning*, volume 2083 of *Lecture Notes in Computer Science*, pages 376–380. Springer, 2001. 1, 2, 2.4.3

[37] Alexandre Riazanov and Andrei Voronkov. Limited Resource Strategy in Resolution Theorem Proving. *J. Symb. Comput.*, 36(1-2):101–115, 2003. 2.4.3

[38] Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier, 2001. 1, 2

[39] J. Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965. 1

[40] Johann M Schumann. *Automated theorem proving in software engineering*. Springer, 2001. 2.4.1

[41] Nourah A. Shenaiber. Smart Semantic Tree Theorem Prover. In *ARW 2011: Proceedings of the Automated Reasoning Workshop 2011*. Glasgow, UK, 2011. 2.2

[42] Nourah A. Shenaiber, Manfred Kerber, and Volker Sorge. Using Herbrand Bases for Building a Smart Semantic Tree Theorem Prover. In *ARW 2015: Proceedings of the Automated Reasoning Workshop 2015*. Birmingham, UK, 2015. 2.2

[43] John Slaney. SCOTT: A Model-Guided Theorem Prover. In *IJCAI-93*, pages 109–114. Morgan Kaufmann, 1993. 2.3

[44] John Slaney. FINDER: Finite domain enumerator system description. In *Automated Deduction CADE-12*, pages 798–801. Springer, 1994. 2.3

[45] John Slaney, Arnold Binas, and David Price. Guiding a Theorem Prover with Soft Constraints. In *16th Europen Conference on Artificial Intelligence*, pages 221–225. IOS press, 2004. 2.3

[46] Geoff Sutcliffe, Matthias Fuchs, and Christian Suttner. Progress in automated theorem proving, 1997-1999. In *Proceedings of the IJCAI*, volume 1, pages 53–60, 2001. 1

[47] Geoff Sutcliffe and Christian Suttner. The TPTP problem library. *Journal of Automated Reasoning*, 21(2):177–203, 1998. 1, 2.4.1, 2.4.2, 7.2, 7.2.1, 7.2.2, 7.2.3, 7.2.4, 8

[48] Geoff. Sutcliffe and Christian Suttner. The state of CASC. *AI Communications*, 19(1):35–48, 2006. 1, 2.4.3

[49] Cesare Tinelli. A DPLL-Based Calculus for Ground Satisfiability Modulo Theories. In Sergio Flesca, Sergio Greco, Giovambattista Ianni, and Nicola Leone, editors, *Logics in Artificial Intelligence*, volume 2424 of *Lecture Notes in Computer Science*, pages 308–319. Springer, 2002. 2, 2.2

[50] Tomás E. Uribe. Sorted Unification Using Set Constraints. In *Automated DeductionÑCADE-11*. Springer, 1992. 7.2.2

[51] Andrei Voronkov. Strategies in rigid-variable methods. In *Proceedings of the 15th International Joint Conference on Artifical Intelligence - Volume 1*, pages 114–121, 1997. 7

[52] Christoph Walther. A Mechanical Solution of Schubert's Steamroller by Many-sorted Resolution. *Artif. Intell.*, 26(2):217–224, 1985. 7.2.2

[53] Hao Wang. Logic of many-sorted theories. *The Journal of Symbolic Logic*, 17:105–116, 6 1952. 7.2.2

[54] Christoph Weidenbach. A sorted logic using dynamic sorts. Max-Planck-Institut für Informatik, 1991. 7.2.2

[55] Christoph Weidenbach, Bijan Afshordel, Uwe Brahm, Christian Cohrs, Thorsten Engel, Enno Keen, Christian Theobalt, and Dalibor Topic. System Description: Spass Version 1.0.0. In *Automated Deduction – CADE-16*, volume 1632 of *Lecture Notes in Computer Science*, pages 378–382. Springer, 1999. 1, 2, 2.4.2, 2.4.3

[56] Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobald, and Dalibor Topic. Spass Version 2.0. In *Automated Deduction CADE-18*, pages 275–279. Springer, 2002. 2.4.2

[57] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In Renate A. Schmidt, editor, *Automated Deduction – CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009. 2.4.2

[58] Christoph Weidenbach, Renate A. Schmidt, Thomas Hillenbrand, Rostislav Rusev, and Dalibor Topic. System Description: Spass Version 3.0. In Frank Pfenning, editor, *Automated Deduction – CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, pages 514–520. Springer, 2007. 2.4.2

[59] Lawrence Wos, George A. Robinson, and Daniel F. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *J. ACM*, 12(4):536–541, 1965. 3.2.1

[60] Qingxun Yu, Mohammed Almulla, and Monroe Newborn. Heuristics used by Herby for semantic tree theorem proving. *Annals of Mathematics and Artificial Intelligence*, 23(3-4):247–266, 1998. 2.1