TABLEAU PROVER GENERATION
CASE STUDY
FOR INTUITIONISTIC
PROPOSITIONAL LOGIC

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2013

By
Fabrizio Principato
School of Computer Science
## Contents

Abstract 7

Declaration 8

Copyright 9

Acknowledgements 10

1 Introduction 11

2 Logic 13
   2.1 Intuitionistic propositional logic 13
      2.1.1 Semantics 15

3 Tableau method 17
   3.1 Introduction to tableau 17
      3.1.1 Tableau method for intuitionistic logic 23

4 Existing provers 26
   4.1 PITP 26
   4.2 FCUBE 33
   4.3 Imogen 35

5 MetTel2 38

6 Implementation of Fitting’s systems 42
   6.1 Basic Fitting’s system 42
      6.1.1 Specification 42
      6.1.2 Tableau rules 43
   6.2 Fitting’s system with derivable rules 46
7 Implementation of PITP systems and its variations

7.1 PITP theorem prover
   7.1.1 Specification
   7.1.2 Tableau rules

7.2 PITP theorem prover with 2-signs variation
   7.2.1 Specification
   7.2.2 Tableau rules

7.3 PITP theorem prover with additional implication rules
   7.3.1 Specification
   7.3.2 Tableau rules

8 Benchmark

8.1 LCL problems set
8.2 SYJ problems set
8.3 SYN problems set

9 Experimental results

10 Conclusion

Bibliography
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Connection between $\alpha$ formulas and their component [DGHP99]</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Connection between $\beta$ formulas and their component [DGHP99]</td>
<td>22</td>
</tr>
<tr>
<td>8.1</td>
<td>ILTP propositional problems summary [ROK07]</td>
<td>63</td>
</tr>
<tr>
<td>8.2</td>
<td>Status for each SYJ problems family</td>
<td>65</td>
</tr>
<tr>
<td>8.3</td>
<td>Status for each SYN problems family</td>
<td>66</td>
</tr>
<tr>
<td>9.1</td>
<td>Unrestricted blocking provers comparison on the LCL and SYJ problems domains</td>
<td>72</td>
</tr>
<tr>
<td>9.2</td>
<td>Unrestricted blocking provers comparison on the SYN problems domain</td>
<td>73</td>
</tr>
<tr>
<td>9.3</td>
<td>Restricted blocking provers comparison on the LCL and SYJ problems domains</td>
<td>75</td>
</tr>
<tr>
<td>9.4</td>
<td>Restricted blocking provers comparison on the SYN problems domain</td>
<td>76</td>
</tr>
<tr>
<td>9.5</td>
<td>PITP, ft-Prolog and ft-C comparison on the LCL and SYJ problems domains ([ILT13])</td>
<td>79</td>
</tr>
<tr>
<td>9.6</td>
<td>PITP, ft-Prolog and ft-C comparison on the SYN problems domain ([ILT13], [AFM08])</td>
<td>80</td>
</tr>
<tr>
<td>9.7</td>
<td>Performance of PITP without Simplification on the SYJ problems domain [AFM08]</td>
<td>80</td>
</tr>
<tr>
<td>9.8</td>
<td>FCUBE performance on some SYJ problems [FFF10]</td>
<td>81</td>
</tr>
<tr>
<td>9.9</td>
<td>Imogen performance on some SYJ problems [MP08]</td>
<td>81</td>
</tr>
<tr>
<td>9.10</td>
<td>Number of ILTP problems solved by the ‘restricted blocking’ version of generated prover</td>
<td>81</td>
</tr>
<tr>
<td>9.11</td>
<td>Number of ILTP problems solved by the ‘unrestricted blocking’ version of generated prover</td>
<td>81</td>
</tr>
<tr>
<td>9.12</td>
<td>Number of ILTP problems solved by other existing provers</td>
<td>81</td>
</tr>
</tbody>
</table>
List of Figures

3.1 Structural rules in the sequent calculus for classical logic . . . . . . . 18
3.2 Connectives rules in the sequent calculus for classical logic [DGHP99]. 19
3.3 Fitting’s tableau system for intuitionistic propositional logic [Fit69]. 23
3.4 Closure rules for Fitting’s tableau system for intuitionistic propositional logic [Fit69]. 24
3.5 Fitting’s alternative tableau system for intuitionistic propositional logic [Fit83]. 25
4.1 Tableau system rules presented in [MMO97]. 28
4.2 Tableau system rules presented in [MMO97]. 28
4.3 Tableau system rules implemented in [AFM04]. 29
4.4 $T \rightarrow$ rules implemented in [AFM04]. 30
4.5 Simplification rule for $T$, $F_C$ and $F$-signed formulas implemented in PITP [AFM06]. 31
4.6 Simplification rule for all the connectives implemented in PITP [AFM06]. 31
4.7 Tableau system rules implemented in FCUBE [FFF10]. 33
4.8 $T \rightarrow$ rules implemented in FCUBE [FFF10]. 34
4.9 Simplification rule for $T$, $F_C$ and $F$-signed formulas implemented in PITP [AFM06]. 34
4.10 Polarities definition [FFF10] . . . . . . . . . . . . . . . . . . . . . . . 34
4.11 Polarities for each connective in intuitionistic propositional logic [MP08] 35
4.12 Translation from polarized to unpolarized formula [MP08] . . . . . . 36
5.1 MetTel2 logical language specification example . . . . . . . . . . . . . 39
5.2 MetTel2 tableau rules specification for classical propositional logic . . 41
6.1 Specification input file for the basic Fitting’s system implementation. 43
6.2 Tableau rule MetTel2 specification for the calculus of Figure 3.3. 47
6.3 Refined version of the calculus of Figure 3.3 . . . . . . . . . . . . . . 48
Abstract

The automated generation of a theorem prover starting from the specification of the logic is a very challenging task. MetTel2 [TSK12] represents a step towards the solution to this task. Starting from the specification of a set of tableau rules, MetTel2 generates a tableau based theorem prover.

Since in automated theorem proving it is very important to generate provers that are fast (this is because the constant growth of the size of the problems the provers have to solve), it is crucial to conduct experiments in order to test how fast a prover is and the range of formulas it can solve.

This project focuses on a case study using MetTel2 to generate provers for intuitionistic propositional logic and to see how these provers perform compared to the existing provers for the same logic fragment. Different rules specification and optimization will be tested to understand how they affect the performance of the provers.
Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
Acknowledgements

I would like to thank my supervisor, Dr Renate A. Schmidt for the advice she gave me and for the time she spent helping me during all the project period.

I would also like to thank my family, for their support and for giving me the opportunity to study at the University of Manchester.
Chapter 1

Introduction

Automated reasoning is fundamental in many computer science areas such as hardware and software verification [EKKV10]. In hardware verification it is important to be able to verify that every single part of a final product behaves correctly. Moreover if bugs are found it is very useful to be able to represent in some way the situation that cause that bug in order to correct it. The same situation happens in software verification. Automated reasoning is not only used for formal verification but also to perform reasoning on databases, in the area of information management.

There exist several techniques to develop an automated theorem prover. Among these, the tableau based reasoning method is one of the most popular. The popularity of tableau methods is probably due to the fact that the rules used are quite easy to understand and it is easy, from a derivation obtained with this method, to output the related model.

Nowadays people from very different areas need automated theorem proving for very different tasks. Hence it would be very useful to have a system that everyone who is interested in having a theorem prover can use to generate his own prover.

The range of activities for which automated theorem prover are used is big. It goes from research purposes to real world problems applications. Thus it is important not only to have provers that behave correctly but also that can scale well on different problem sizes.

MetTel2 [TSK12] is a tool developed by R. Schmidt, D. Tishkovsky and M. Khodadadi in the School of Computer Science at the University of Manchester. Having in input the specification of a set of tableau rules, MetTel2 automatically generates a tableau based theorem prover. MetTel2 can generate provers for different logics, such as description logics, modal logics and hybrid logics [ST12].
In the project the focus will be on automated reasoning for intuitionistic logic. Intuitionistic logic is useful not only for verification but also for program synthesis [ES98]. This is because a proof in intuitionistic logic directly corresponds to a computable program.

Several theorem provers are available for intuitionistic propositional logic. Some of them are based on the tableau method [FFF10] [AFM06] and some others are based on different automated reasoning approaches [MP08].

From my previous experience working with intuitionistic propositional logic [Pri12] (unpublished), there is no clear advantages on using one specific method. It turns out that what influence the performance of a theorem prover on a formula is the structure of the formula to be proved. In the experiments some methods such as Maslov’s inverse method [Mas68] implemented in Imogen [MP08] were better to prove formulas that are not valid while the tableau method implemented in PITP [AFM06] performs better on valid formulas.

The rest of the report is organised as follows. Chapter 2 gives an introduction to intuitionistic propositional logic, its syntax and semantics. Chapter 3 introduces the tableau method in general. Moreover the tableau method for intuitionistic propositional logic is introduced. Chapter 4 presents some existing theorem provers for intuitionistic propositional logic. Chapter 5 presents the MetTel2 tool, explaining how does it works and how to use it. Chapter 6 and Chapter 7 presents the theorem provers implemented using MetTel2. Chapter 8 introduce ILTP, the benchmark suite used to test all the provers. Chapter 9 presents the experimental results, together with an analysis of the performance of the generated provers. Chapter 10 presents the conclusion and some consideration about future work.
Chapter 2

Logic

2.1 Intuitionistic propositional logic

In 1907 Brouwer [Bro80] pointed out that classical logic works well for reasoning about finite situations but it does not present the same behaviour when one wants to reason about infinite structures such as sets.

Classical logic works well when the truth value of a proposition is based on direct observation but when the domain of the problem to be analysed is not finite this direct observation is simply not feasible. Let’s consider a property $P$ and an infinite set of elements. It is possible to be in the situation in which it is not possible to say that all the object satisfy the property $P$ but at the same time it is not possible to find an object in the set that does not satisfy $P$ because this would require an observation of infinite elements.

From this observation it follows that the law of excluded middle it is not always true. From a classical point of view $A \lor \neg A$ it is always true no matter what the truth value of $A$ is, but Brouwer said that there are situation in which the law of excluded middle is not true because there is no evidence about which one between $A$ and $\neg A$ is true.

This of course does not mean that the law of excluded middle is always false. If the truth value of $A$ is known or $A$ is a decidable proposition then the law of excluded middle is true.

Brouwer’s criticism tries to shows how the truth concept cannot be an absolute concept but instead is deeply related to the proof concept.

The first formalization of the concept expressed by Brouwer is due to Heyting [Hey30] who in 1930 developed a formal system for intuitionistic propositional and
predicate logic.

As a result of the radically different meaning of the truth concept, the main difference between classical and intuitionistic logic is the interpretation of the logical connectives. In the propositional fragment of both the logics the connective are: $\lor$, $\land$, $\neg$, $\supset$. The negation is defined as $A \supset \bot$. It is important to notice is that all the connectives in intuitionistic propositional logic are independent while in classical propositional logic this is not true. For example the connective $\lor$ can be defined in term of $\neg$ and $\land$ as $A \lor B \equiv \neg(\neg A \land \neg B)$. The fact that in intuitionistic logic all connectives are independent is related to the truth concept from a intuitionistic point of view and in particular how negation is handled. Below it follows an analysis of each connective from a classical and intuitionistic point of view.

- **Implication:** In classical logic the truth value of $A \supset B$ only depends from the truth value of $A$ and $B$. It is sufficient to see when $A \supset B$ is false (and this happens when $A$ is true and $B$ is false). In all the other cases $A \supset B$ will be true.

  In intuitionistic logic in order to prove (recall that the truth concept is strictly related to the proof concept) $A \supset B$ it is necessary to have a mechanism to transform every proof of $A$ in a proof of $B$.

  The truth value of $A \supset B$ then will not be dependent on the truth value of $A$ and $B$ but depends on a concrete proof of both $A$ and $B$.

- **Negation:** In classical logic $\neg A$ is true only if $A$ is false and vice versa. The truth value of $\neg \neg A$ then will be equivalent to the truth value of $A$.

  In intuitionistic logic the negation is defined as $A \supset \bot$. As a consequence of the different interpretation of the implication in intuitionistic logic, $\neg \neg A$ will not be equivalent to $A$. To say that $\neg A$ is true in intuitionistic logic means that a procedure is available that transforms each proof of $A$ into a contradiction.

- **Disjunction:** In classical logic $A \lor B$ is true if at least one between $A$ and $B$ is true.

  In intuitionistic logic in order to prove $A \lor B$ it is necessary to show a proof of $A$ or a proof of $B$.

  So at first glance it seems hard to recognize the exact meaning of the disjunction in intuitionistic logic: why do we want to say that $A \lor B$ is true only after we know (we have a proof) of either $A$ or $B$? Let’s consider the situation in which $A \lor B$ is part of a more complex proposition such as $(A \lor B) \supset C$. This in intuitionistic logic means that it is possible to go from a proof of $A$ or $B$ to a proof of $C$. In classical logic the same proposition can be written as $\neg(\neg A \land \neg B) \supset C$. 

Intuitionistically this two proposition are very different. From the latter formulation it is not possible to extract a proof of either $A$ or $B$ in order to obtain a proof of $C$ while this was possible with the former formulation.

In intuitionistic logic the information inside a disjunction is useful and actively used to produce new proof and this don’t happen when the disjunction is replace by the classically equivalent conjunction.

- **Conjunction**: Regarding the conjunction the only difference between the classical and intuitionistic points of view is that in order to assert $A \land B$ in intuitionistic logic it is necessary to have a concrete proof of $A$ and $B$.

### 2.1.1 Semantics

There are different possible semantics for intuitionistic propositional logic. One of these is the one introduced by Kripke in 1965 [Kri65]. While the semantics for classical logic is only based on truth tables and can be seen as a “static” semantics, Kripke’s semantics is “dynamic” and is based on how knowledge evolves over time. Instead of calculating the truth value of a formula $F$ by just looking at the truth values of the proposition from which $F$ is made, Kripke proposes to consider the behaviour of $F$ while all its components propositions gradually vary their truth value from false to true according to a specific model.

**Definition 1.** A Kripke model is a triple $\langle K, \leq, t \rangle$ where

- $K$ is a finite set of elements called worlds;
- $\leq$ is a partial ordering relation over the worlds;
- $t$ is a function that associates to each world $k$ an increasing set of atomic formulas $t(k)$ such that for every $k, k' \in K, k \leq k' \Rightarrow t(k) \subseteq t(k')$.

An atomic formula $B$ is true in a node $k$ if and only if $B \in t(k)$. In symbols $k \models B$ and in words “$k$ forces $B$”. In order to pass from atomic formulas to complex formulas it is necessary to define the forcing relation over the logical connectives.

**Definition 2.** Let $k$ be a world of a Kripke models and $B_1$ and $B_2$ two atomic formulas.

- $k \models B_1 \land B_2$ if and only if $k \models B_1$ and $k \models B_2$;
- $k \models B_1 \lor B_2$ if and only if $k \models B_1$ or $k \models B_2$;
- $k \models B_1 \supset B_2$ if and only if for every $k' \geq k$, if $k' \models B_1$ then $k' \models B_2$;
• $k \models \neg B_1$ if there is no $k' \geq k$ such that $k' \models B_1$.

The $t$ function is monotone so if $k \models B$ then for every $k'$ such that $k \leq k'$, $k' \models B$. It is possible to prove that when a formula $F$ becomes true in a world of a Kripke model, then it will remains true in all the future nodes of the same model. From this Kripke has stated that a formula $F$ is intuitionistically valid if and only if it is true in all the worlds of all the Kripke models.
Chapter 3

Tableau method

3.1 Introduction to tableau

Automated reasoning techniques can be summarized in two big categories:

- Those using backward chaining strategies: The proof search proceeds from the formula to be proved towards more simple formulas using a set of rules that allows to split complex formulas in more simple ones.

- Those using forward chaining strategies: The proof starts with a set of rules and a database of known facts (usually clauses). The proof search goes on by searching correspondences between the premises of one rule and the facts inside the database. When a match is found, new facts are generated according to the rule used and this facts are added to the database. This process continues until the goal formula is generated or no new facts can be generated.

The tableau method belongs to the first category. Generally speaking the tableau method is a formal proof procedure.

Historically the origins of the tableau method are the work of Gentzen [Gen35], who in 1935 introduced his sequent calculus and natural deduction for classical logic.

In the natural deduction system, for each connective there is an introduction and an elimination rule. The introduction rule specifies how to construct a formula that has as the main connective the one introduced by the rule. The elimination rule specifies how to decompose a formula in order to eliminate the main connective. Moreover, during a derivation using the natural deduction system, assumptions can be done. These can be then discharged during the derivation as a consequence of the application
A sequent is an expression of the following form

\[ A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \]

Here \( A_1, \ldots, A_n \) and \( B_1, \ldots, B_m \) are formulas. What a sequent says is that the disjunction of the formulas on the right side follows from the conjunction of the formulas on the left side.

The sequent calculus has rules and axioms. The axioms are called initial sequents and are sequents of the form \( A \rightarrow A \) where \( A \) is a formula. Regarding the rule to construct derivations, in the sequent calculus of Gentzen there are two types of rules: structural rules and rules for the connectives.

In all the above rules \( \Delta, \Omega \) are sequences of formulas that can be empty, while \( A \) and \( B \) are formulas. Figure 3.1 shows the structural rules while Figure 3.2 shows the connectives rules for classical logic.

Derivations performed using the rules as they appear represent an application of the forward chaining strategy. The goal formula is built starting from axiom (initial sequent) using the rules.

In contrast if the rules are applied upside down and the proof starts with the goal formula the connection between tableau method and sequent calculus it is clear since both are based on a decomposition of the goal formula.

Gentzen also introduced a sequent system for intuitionistic logic. Since the right hand side of a sequent is intended as a disjunction and given the constructiveness of
intuitionistic logic it should be stated clearly which one of the formulas on the right hand of a sequent is a consequence of the conjunction of the formulas on the left hand of the sequent. This leads Gentzen to modify the sequent system for intuitionistic logic in order to admit only sequent with at most one formula on the right hand side.

After Gentzen another major contribution was due to Beth [Bet55], [Bet56]. He was moved by a semantic analysis of the proofs. What Beth wants to do is to have a systematic approach to find counter examples of a given sequent. He called this approach semantic tableau because he organized the proof in table form.

In the semantic tableau techniques the table is split in two parts, one labelled “valid” and the other one labelled “not valid”.

In order to prove the sequent $A_1, \ldots, A_n \rightarrow B$ at the beginning all the formulas on the left hand side are inserted under the “valid” column while $B$ is inserted under the “not valid” column. The next step is to systematically break down the complex formulas in each column by splitting them into simpler ones.

For example, if the formula $A \land B$ is under the “valid” column then it is possible to add $A$ and $B$ under the same column.

There are cases in which it is not possible to execute this step in an unique way. Considering the formula $A \lor B$ under the “valid” column. In this case it is not possible to say which one between $A$ and $B$ is “valid”. The solution Beth proposed was to
split the “valid” column in two sub-columns in order to have both the possibilities. Of course using this technique requires some kind of annotation to label the couple of “valid” and “not valid” columns that are a consequence of a split. By repeating this step two conditions can be reached:

- The same formula appears in the “valid” column and at the same time also in the “not valid” column. This represents an impossible situation.
- The contradictory scenario described above never happens.

In the former case there are no counter examples to the validity of the sequent. In fact, starting from the assumption that the sequent was not valid, an impossible situation has been reached. This means that the starting sequent is valid.

In the latter case Beth says that from the tableau itself it is possible to extract a counter example that shows that the starting sequent is not a valid sequent.

In the propositional case, the semantic tableau approach as described above is a decision procedure.

Regarding intuitionistic logic, Beth proposed his semantics and a tableau system for it. He proves that his semantics tableau system for intuitionistic logic is sound and complete with respect to his semantics and he also proved the equivalence between his tableau system and the sequent calculus for intuitionistic logic proposed by Gentzen, even if in Beth’s version it is allowed to have more than one formula on the right hand side of a sequent.

In the same years of Beth work, Hintikka introduces the concept of model sets [Hin55]. Like Beth, Hintikka motivations were mainly guided by semantics concerns. For Hintikka a proof of $A$ is a systematic approach to construct a model for $\neg A$. If this attempt fails, $A$ is valid.

The main concept introduced by Hintikka is the concept of model sets. The first assumption he made is that negation only occurs at atomic level. Every other non-atomic negation can be eliminated using the transformation given by the normal form rules. Considering a model $M$ and a set $\mu$ of propositional variables that are true in $M$ and the following set of conditions.

1. (a) If $A \in \mu$, then $\neg A \notin \mu$, $A$ atomic.
2. (a) If $A \land B \in \mu$, then $A \in \mu$ and $B \in \mu$.
3. (a) If $A \lor B \in \mu$, then $A \in \mu$ or $B \in \mu$.
4. (b) If $\neg A \in \mu$, then $A \notin \mu$, $A$ atomic.
5. (b) If \( A \in \mu \) and \( B \in \mu \), then \( A \land B \in \mu \).

6. (b) If \( A \in \mu \) or \( B \in \mu \), then \( A \lor B \in \mu \).

If \( \mu \) is a set that meet all the conditions listed above, then there is a model \( M \) whose true formulas are the ones inside \( \mu \). Hintikka proved that it is sufficient that \( \mu \) satisfy only the conditions marked with (a) in order to obtain \( M \) since the set \( \mu \) can be extended to satisfy also the (b)-conditions. He called the sets meeting (a)-conditions model sets.

From these considerations it follows that every model set is satisfiable. Hintikka idea to proof a formula \( A \) is to start with \( \neg A \) and to prove that this does not belong to any model sets. This can be done by starting to assume that \( \neg A \) belongs to some model sets and, by using the (a)-conditions, to derive a contradiction.

In 1960 Lis [Lis60] proposed his solution to simplify the representation of tableau derivation. Instead of using the two columns “valid” and “not valid” as Beth he proposed to use signs to distinguish between “valid” and “not valid” formulas without the need of splitting them into two columns. Lis proposed + and − as two symbol to identify a “valid” and “not valid” formula respectively. Then he proposed the following rules:

1. If \( \pm \neg A \) then \( \mp A \)
2. if \(+ (A \land B) \) then \(+ A, + B \)
3. if \(- (A \lor B) \) then \(- A, - B \)
4. if \(- (A \supset B) \) then \(+ A, - B \)
5. if \(- (A \land B) \) then \(- A \mid - B \)
6. if \(+ (A \lor B) \) then \(+ A \mid + B \)
7. if \(+ (A \supset B) \) then \(- A \mid + B \)

Rules from 2 to 4 are called conjunctive rules while rules from 5 to 7 are called disjunctive rules and they lead to two separates branches.

The tableau method became well known in 1968 due to the work of Smullyan [Smu68]. Instead of using + and − as signs he used \( T \) and \( F \). He divided the formulas in two groups: type\( \alpha \) formulas and type\( \beta \) formulas.

Formulas of type\( \alpha \) are the ones that behave conjunctively while type\( \beta \) formulas behave disjunctively. For each type of formulas he defined two components: \( \alpha_1 \) and \( \alpha_2 \) for type\( \alpha \) formulas (also called \( \alpha \) formulas) and \( \beta_1 \) and \( \beta_2 \) for type\( \beta \) formulas (also called \( \beta \) formulas). The connection between a formula and its component for classical
logic is showed in Table 3.1 and Table 3.2. Considering the separation into $\alpha$ and $\beta$ formulas it is possible to represent all the inference rules of the tableau method as introduced by Smullyan (he called it *analytic tableau*) with two general rules:

$$
\begin{array}{c|c|c}
\alpha & \alpha_1 & \alpha_2 \\
\hline
T(A \land B) & TA & TB \\
F(A \lor B) & FA & FB \\
F(A \supset B) & TA & FB \\
T\neg A & FA & FA \\
F\neg A & TA & TA \\
\end{array}
$$

Table 3.1: Connection between $\alpha$ formulas and their component [DGHP99].

$$
\begin{array}{c|c|c}
\beta & \beta_1 & \beta_2 \\
\hline
F(A \land B) & FA & FB \\
T(A \lor B) & TA & TB \\
T(A \supset B) & FA & TB \\
\end{array}
$$

Table 3.2: Connection between $\beta$ formulas and their component [DGHP99].

The signs play an important role in order to make clear the relation between the sequent calculus and the tableau method. Intuitively a proof using the tableau method is a proof using the sequent calculus backwards. The goal in the sequent calculus is to prove the validity of a sequent while the tableau method tries to prove that a set of formulas is unsatisfiable. Considering the sequent $A_1, \ldots, A_n \rightarrow B$ it is easy to convert it in a set of formulas for the tableau method even without signs. The result will be $\{A_1, \ldots, A_n, \neg B\}$.

The usefulness of signs comes clear when one tries to convert a set of formulas into a sequent. Considering the set $\{A_1, A_2, \neg A_3\}$. There are two possible sequent that corresponds to this set. One is $A_1, A_2 \rightarrow A_3$ while the other one is $A_1, A_2, \neg A_3 \rightarrow$. Smullyan proposed to use the signs in order to make clear which sequent a set of formulas refers to. If $S$ is a set of formulas such as $\{TA_1, \ldots, TA_n, FB_1, \ldots, FB_m\}$ then $|S|$ is the correspondent sequent $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$. In this way is easy to understand the relation between sequent proofs and tableau proofs.
3.1.1 Tableau method for intuitionistic logic

Gentzen and Beth both developed sequent calculi for intuitionistic logic, but the first tableau system for intuitionistic logic was just due Fitting in 1969 [Fit69].

He uses ideas from both Beth regarding the rules and Smullyan regarding the use of type $\alpha$ and type $\beta$ formulas and signed formulas instead of unsigned ones. This leads to a system with only one column (in the Beth sense) instead of two.

The complete set of rules for this tableau system is listed in Figure 3.3.

For each rule the premise represents a branch of a tableau and the conclusion either add some formulas to the same branch or split into two branches. Moreover in all the rules, $S$ represents the set of formulas in the current branch.

For example if in a branch there is a set of signed formulas $S$ and one of these formulas is $T(A \lor B)$ then the relative rule will add $TA$ and $TB$ to the branch. All formulas with only one branch in the conclusion are the same that Smullyan called type $\alpha$ formulas, while rules with more than one branch in the conclusion are the formulas called type $\beta$ by Smullyan.

Recall that in the Beth tableau system for intuitionistic logic a contradiction is reached when the same formula appear in the “valid” and “not valid” columns. In this
Figure 3.4: Closure rules for Fitting’s tableau system for intuitionistic propositional logic [Fit69].

Here instead of talking about two columns there are different branches. A branch is closed if it contains $TA$ (corresponding to a formula under the “valid” column) and $FA$ (corresponding to a formula under the “not valid” column) for any formula $A$. Since there are rules that split a branch into two branches, we say that a tableau is closed if every branch is closed. Figure 3.4 shows the rules used to close a branch.

It is possible to split the rules in which the conclusion contains the set $S_T$ in two rules. First apply the deletion of every $F$-signed formulas and then apply the correspondent rule of Figure 3.3. Fitting in [Fit83] calls the deletion rule “branch modification rule”. It is important to notice that the “branch modification rule” only plays a role in the rule for which the semantics involves information about any future Kripke’s world. So the branch modification rule in a sense is an answer to the question “What informations should be propagated from the current world to the next word?”. The answer to this question is that every formula that is $T$-signed should be propagated since this means that a proof of these formula has been found, while it is possible to forget about $F$-signed formula since at this stage of the proof, no information about the validity of these formulas is available.

Fitting in [Fit83] also presents another tableau system for intuitionistic propositional logic. It is based on the previous one but with some differences regarding the use of the “branch modification rule”. Instead of dividing the rule according to the Smullyan notation of $\alpha$ and $\beta$ formula, in this system Fitting proposes to split the formulas only according to their signs. This does not change the behaviour of $type\alpha$ and $type\beta$ formulas from the point of view of the number of branches in the conclusion but this time the “branch modification rule” is applied to all the negative formulas (not only the ones involving $\supset$ and $\neg$). Figure 3.5 shows the resulting rules.

Fitting calls this system “Gentzen system” because it is strictly related to the sequents calculus for intuitionistic propositional logic proposed by Gentzen.

Recall that in Gentzen’s sequents calculus only one formula can appear on the right hand of a sequent and the translation of a sequent into a set of formulas for the tableau method convert each formulas in the right hand of the sequent in an $F$-signed formula.
This means that in every branch of a tableau, at most one $F$-signed formula can appear.

The system of Figure 3.5 allows more than one $F$-signed formula to appear in a branch but when one of these formulas is used in the application of a rule, all the other $F$-signed formula will be deleted by the “branch modification rule”.

Fitting shows that both the system presented are complete and correct with respect to Kripke semantics for intuitionistic propositional logic presented in Section 2.1.1.
Chapter 4

Existing provers

4.1 PITP

PITP is a theorem prover developed at the Universita’ di Milano-Bicocca by Avellone et. all [AFM06]. It is based on an improved version of the tableau calculus for intuitionistic propositional logic proposed by Fitting [FIT69]. The final tableau system has been obtained by improving the calculus of [AFM04] and [MMO97] respectively.

One of the main issues regarding automatic theorem proving in intuitionistic propositional logic is the number of duplications required to prove a formula. More duplications mean that the same formula needs to be used more times during the proof and hence the search space can be big. For this reason, reducing the number of duplications that occur during a derivation is a key feature to implement a prover that is performant.

The tableau calculus presented in [MMO97] is a duplication free calculus with respect to intuitionistic propositional logic and it is shown in Figure 4.1 and Figure 4.2.

In addition to the sign $T$ and $F$ a new sign $F_C$ is introduced. This sign represents the forcing of intuitionistic negation and can be represented as $T\neg$. In fact the only rule that introduce the sign $F_C$ is the rule $T\neg$. If during a derivation the signed formula $T(\neg A)$ is derived, then $F_CA$ will be introduced. This represents the fact that no other Kripke’s world will force $A$. In contrast, the rule for $T\neg$ in Fitting first tableau system for intuitionistic logic (see Figure 3.3) will introduce the formula $FA$. Recall that an $F$-signed formula is a formula for which no constructive proof have been found so far, while $F_CA$ express the fact that a proof of $\neg A$ has been found.

It is possible to distinguish signed formulas in PITP in two categories: the certain part, containing $T$-signed and $F_C$-signed formulas and the uncertain part, containing
signed formulas. During the derivation in general both the certain and uncertain formulas will be present in a branch. The situation is different every time a new Kripke’s world is created. At this point, the only information that will be carried into the new world is constituted by the certain part, represented as $S_C$.

The main improvements of this calculus compared to the one proposed by Fitting is represented by the rules of Figure 4.2. Since the rule for $T \rightarrow$ is the only one that requires duplication, it has been substituted by four different rules, according to the structure of the antecedent of the implication to which the rule is applied. The resulting system is composed by two kinds of rules, depending on how they act on the formula in the premise. On the one hand there are rules that decompose the formula in the premise into subformulas. On the other hand there are rules that replace the formula in the premise with one intuitionistically equivalent formula. Among these “replacement” rules, if there are two formulas in the conclusion, the conjunction of these formulas is intuitionistically equivalent to the formula in the premise. The replacement rules are the ones presented in Figure 4.2 with the only exception of the rule $T \rightarrow \rightarrow$ that is a “decomposition” rule.

**Definition 3.** Let $A$ be a formula. The degree of $A$, denoted by $\text{deg}(A)$ is defined as follow.

- If $A$ is an atom then $\text{deg}(A) = 0$.
- If $A = B \rightarrow C$ is an atom then $\text{deg}(A) = \text{deg}(B) + \text{deg}(C) + 1$.
- If $A = B \land C$ is an atom then $\text{deg}(A) = \text{deg}(B) + \text{deg}(C) + 2$.
- If $A = B \lor C$ is an atom then $\text{deg}(A) = \text{deg}(B) + \text{deg}(C) + 3$.
- If $A = \neg B$ is an atom then $\text{deg}(A) = \text{deg}(B) + 1$.

Duplications are eliminated from the calculus in the following way.

- The rules $T \rightarrow \land, T \rightarrow \lor$ replace the antecedent of the implication in the formula in the premise with a formula with lower degree. For example in the rule $T \rightarrow \land A \land B$ is replaced by $A$ which has lower degree than $A \land B$. The consequent of the implication is $B$ that has lower degree than $A \land B$. Considering $A$, $B$ and $C$ atomic, the degree of the formula in the premise is 3, while the degree of the formula in the conclusion is 2.

- The rule $T \rightarrow \rightarrow$ in the left branch replaces the formula $T((A \rightarrow B) \rightarrow C)$ with $T(B \rightarrow C)$ that has lower degree.
### Tableau system rules presented in [MMO97].

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S, T(A \land B) \quad T \land )</td>
<td>( S, TA, TB )</td>
</tr>
<tr>
<td>( S, F(A \land B) \quad F \land )</td>
<td>( S, FA</td>
</tr>
<tr>
<td>( S, F_C(A \land B) \quad F_C \land )</td>
<td>( S, F_CA</td>
</tr>
<tr>
<td>( S, T(A \lor B) \quad T \lor )</td>
<td>( S, TA</td>
</tr>
<tr>
<td>( S, F(A \lor B) \quad F \lor )</td>
<td>( S, FA )</td>
</tr>
<tr>
<td>( S, F(A \lor B) \quad F \lor )</td>
<td>( S, FB )</td>
</tr>
<tr>
<td>( S, F_C(A \lor B) \quad F_C \lor )</td>
<td>( S, F_CA, F_CB )</td>
</tr>
<tr>
<td>( S, F_C(A \to B) \quad F_C \to )</td>
<td>( S, F_C(A \to B) \quad F \to )</td>
</tr>
<tr>
<td>( S, T(\neg A) \quad T \neg )</td>
<td>( S, F(\neg A) \quad F \neg )</td>
</tr>
<tr>
<td>( S, T(\neg A) \quad T \neg )</td>
<td>( S, F_C(\neg A) \quad F_C \neg )</td>
</tr>
</tbody>
</table>

where \( S_C = \{ TA \mid TA \in S \} \cup \{ FC_A \mid FC_A \in S \} \).

Figure 4.1: Tableau system rules presented in [MMO97].

- Instead of having the following rule

\[
\frac{S, F(A \lor B)}{S, FA, FB}
\]

the rule is split into two rules. This is because during the derivation one between

\( FA \) and \( FB \) plays no role, generating a sort of hidden duplication.

A significant improvement of the calculus of Figure 4.1 and Figure 4.2 has been pro-
posed in [AFM04]. The calculus is presented in Figure 4.3 and Figure 4.4. The main

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S, T(A \to B) \quad T \to AN )</td>
<td>( S, T(A \land B \to C) \quad T \to \land )</td>
</tr>
<tr>
<td>( S, FA</td>
<td>S, TB )</td>
</tr>
<tr>
<td>( S, T(A \to (B \to C)) \quad T \to \land )</td>
<td>( S, T((A \lor B) \to C) \quad T \to \lor )</td>
</tr>
<tr>
<td>( S, T((A \lor B) \to C) \quad T \to \lor )</td>
<td>( S, T(A \to C), T(B \to C) )</td>
</tr>
<tr>
<td>( S, T((A \to B) \to C) \quad T \to \to )</td>
<td>( S, T((A \to B) \to C) \quad T \to \to )</td>
</tr>
<tr>
<td>( S, F(A \to B), T(B \to C) \mid S, TC )</td>
<td>( T \to \to )</td>
</tr>
</tbody>
</table>

Figure 4.2: Tableau system rules presented in [MMO97].
Figure 4.3: Tableau system rules implemented in [AFM04].

A minor change with respect to the calculus of [MMO97] is represented by the rule $T \rightarrow \land$. Instead of having just one rule for $A$ atomic or negated formula, two different rules are introduced ($T \rightarrow \text{Atom}$ and $T \rightarrow \neg$). In particular the rule for $T \rightarrow \text{Atom}$ is invertible. Recall that a rule is invertible if the derivability of the formulas in the premise implies the derivability of the formulas in the conclusion. Applying non-invertible rule lead to non-deterministic strategy, hence it is very important to limit the application of such rules as much as possible.

The calculus of Figure 4.3 and Figure 4.4 is the one at the basis of PITP. The basic proof-search procedure of PITP divides the rules into six groups according to the number of branches in the conclusion and the invertibility of each rule. In order to...
reduce the search space it is better to apply invertible rule before non-invertible ones. Among the same kind of rule, applying the rule with only one branch in the conclusion helps to reduce the number of tableau nodes to be created. The resulting group are the following:

- **Group 1** = \{\(T \land, F \lor, F_{C} \lor, T \neg, T \rightarrow Atom, T \rightarrow \land, T \rightarrow \lor\} \)

- **Group 2** = \{\(T \lor, F \land\} \)

- **Group 3** = \{\(F \neg, F \rightarrow\} \)

- **Group 4** = \{\(T \rightarrow \rightarrow, T \rightarrow \neg\} \)

- **Group 5** = \{\(F_{C} \rightarrow, F_{C} \neg\} \)

- **Group 6** = \{\(F_{C} \land\} \)

PITP will attempt to apply the rule in ascending group id order. This is because the rules in **Group 1** are invertible and with only one branch in the conclusion, rules in **Group 2** are invertible with exactly two branches in the conclusion, rules in **Group 3** are non-invertible with only one branch in the conclusion and rules in **Group 4** are non-invertible with exactly two branches in the conclusion. The rules in **Group 5** and **Group 6** are special because if applied only when no other rules is applicable, then backtracking is not required and completeness is not lost.

Starting from this basic procedure, PITP implements some optimization to reduce backtracking and branching.
The first optimization implemented is Simplification. Simplification is a technique introduced in [Mas98] for classical and modal logics to reduce the proof search space. PITP adapts simplification to intuitionistic propositional logic.

Simplification consists of a set of new rules that have to be added to the tableau rules.

Figure 4.5 shows three simplification rules implemented in PITP. The first rule replaces every occurrence of a T-signed formula with \( \top \). This is possible because of the semantics of intuitionistic propositional logic. If a formula \( A \) is forced in a world \( v \) then for every subsequent world \( v' \), \( v' \) will force \( A \). So the forcing of \( A \) starting from \( v \) coincides with the forcing of \( \top \). The same considerations apply to the second rule.

The third rule is different. Here the replacement of the occurrences of \( A \) proceed until \( \neg \) or \( \rightarrow \) is found and then it stops. This is also a consequence of how the Kripke semantics defines the intuitionistic negation and implication. The rules of Figure 4.5 are invertible, hence no backtracking is required. In addition to the rules of Figure 4.5, there are also simplification rules for each connective, as Figure 4.6 shows.

Simplification can drastically reduce the number of formulas to be considered for the application of the tableau rules and hence speedup the performance of the prover.

The second optimization implemented in PITP exploits the symmetry of different formulas to avoid backtracking. Let \( A \) and \( B \) be the following formulas:

\[
A = (W \land Q) \lor (T \rightarrow (W \lor Q)) \quad B = (C \land D) \lor (E \rightarrow (C \lor D))
\]
If it is possible to compute a permutation $\tau$ on propositional variables to transform $A$ into $B$. Then from the information about the satisfiability of $A$ it will be possible to obtain information about the satisfiability of $B$ without performing any additional derivation step. PITP build the permutation $\tau$ using the function $\text{BUILDPERM}$ that take as input two formulas $H$ and $H'$ and return a permutation $\tau$ or $\text{FAIL}$ if $\tau$ does not exists. $\text{BUILDPERM}$ works as it follows.

1. If $H$ and $H'$ are propositional variables and $\tau(H)$ and $\tau(H')$ are not defined, then set $\tau(H) = H'$ and $\tau(H') = H$ and return $\tau$. Otherwise return $\text{FAIL}$.

2. If $H$ and $H'$ are of the kind $\neg A$ and $\neg A'$ respectively, then return $\text{BUILDPERM}(A,A',\tau)$.

3. if $H$ and $H'$ are of the kind $A \text{ SI } B$ and $A' \text{ SI } B'$ respectively, with $\text{SI} \in \{\lor, \land, \rightarrow\}$ then $\tau = \text{BUILDPERM}(A,B,\tau)$. If $\tau$ is not $\text{FAIL}$ return $\text{BUILDPERM}(B,B',\tau)$. Otherwise return $\text{FAIL}$.

If a permutation is found, PITP will use it as follows to avoid backtracking. Let $S$ be a set of formulas and $H$ and $H'$ two formulas with the structure of Group 3 inside $S$. Suppose $U' = (S \setminus \{H'\})_C \cup R_1(H')$ is realized by a Kripke model $K'$. Here $R_1(A)$ represent the left branch of the conclusion of the rule that applies to $A$, and $(S \setminus \{H'\})_C$ represent the certain part of the set $(S \setminus \{H'\})$. At this point if there is a permutation $\tau$ such that for every $V \in U$, $\tau(V) \in U'$ where $U = (S \setminus \{H\})_C \cup R_1(H)$ then $U$ is realized by a Kripke model $K$ such that for every propositional variable $p$ and every world $\alpha$, $\alpha \models_K p$ in $K'$ if and only if $\alpha \models_K p$ in $K$.

The last two optimizations implemented in PITP exploit the fact that on formulas containing only conjunctions and disjunctions the classical and the intuitionistic interpretation coincide. Let $\delta$ be the set of all signed atom of a set of formulas $S$ and let $\theta$ be the classical interpretation defined as follows. If $T p \in \delta$ then $\theta(p) = \text{true}$, otherwise $\theta(p) = \text{false}$. After applying every possible simplification to a sets of formulas $S$ all the $F$-signed in which the only connectives are disjunction and conjunction are satisfied by $\theta$. This means that PITP can ignore these formulas, reducing the number of formulas eligible for rule application.

The last optimization works as it follows. If in $S$ there are no formulas with the structure of Group 1 and Group 2 rules, PITP tries to check if $\theta$ realizes $S$. In this case, there is no need to proceed with the proof since $S$ is satisfiable.

PITP has been mainly used for research purposes and has been extensively tested on ILTP library ([ILT13], [ROK07]).
Figure 4.7: Tableau system rules implemented in FCUBE [FFF10].

4.2 FCUBE

FCUBE is a theorem prover developed at the Universita’ of Milano-Bicocca by Fiorino et all ([FFF10]). FCUBE is a Prolog theorem prover that implements the same calculus of PITP (Figure 4.3 and Figure 4.3 ) with the only difference that the sign $F_C$ has been removed. The formula $F_C A$ has been replaced by the equivalent formula $T \neg A$. Figure 4.7 and Figure 4.8 show the complete tableau system implemented in FCUBE.

FCUBE implements simplification in the same way as PITP does. In addition to Boolean simplification, FCUBE implements a set of rules that allow the substitution of propositional variables with constant polarity with $\top$ or $\bot$. These rules are presented in Figure 4.9, where $p$ is a propositional variable. Figure 4.10 shows the definition of $\preceq^+, \preceq^-$ and $\preceq_w$ respectively.

**Definition 4.** Let $S$ be a set of signed formulas. $p \preceq S$, with $\preceq \in \{\preceq^-, \preceq^+, \preceq_w\}$ iff for every $A \in S$, $p \preceq A$.

The proof search procedure of FCUBE applies both the type of simplification at the beginning in order to reduce the search space. Then, in the same way as PITP does, invertible rules have precedence over non-invertible ones. To output the result, FCUBE use two function: REAL and CM. REAL takes as input a Kripke model $K$ and a set of
\[
S,T,A,T(A \rightarrow B) \quad\Rightarrow\quad S,T(A \rightarrow (B \rightarrow C)) \quad T \rightarrow \land
\]

\[
S,T(\neg A \rightarrow B) \quad\Rightarrow\quad S_T,T(A \rightarrow B) \quad T \rightarrow \neg
\]

\[
S,T((A \lor B) \rightarrow C) \quad\Rightarrow\quad S_T,T(A \rightarrow p),T(B \rightarrow p),T(p \rightarrow C) \quad T \rightarrow \lor \text{ with } p \text{ new atom}
\]

\[
S,T((A \rightarrow B) \rightarrow C) \quad\Rightarrow\quad S_T,T,A,Fp,T(p \rightarrow C),T(B \rightarrow p) \quad| S,T C \quad T \rightarrow \rightarrow \text{ with } p \text{ new atom}
\]

where \( S_T = \{ TA \mid TA \in S \} \).

**Figure 4.8:** \( T \rightarrow \) rules implemented in FCUBE [FFF10].

| S | S[p/\top] | REP if \( p \preceq^+ S \) | S | S[p/\bot] | REP if \( p \preceq^- S \) | S | S[p/\bot] | REP if \( p \preceq_w S \) |
|---|---|---|---|---|---|---|---|

**Figure 4.9:** Simplification rule for \( T, F_C \) and \( F \)-signed formulas implemented in PITP [AFM06].

- \( p \preceq^- Fp \) and \( p \preceq^+ Tp \)
- \( p \preceq^+/\neg S \top \) and \( \preceq^-/\neg S \bot \)
- \( p \preceq^+/\neg Sq \) if \( q \) is a propositional variable different from \( p \)
- \( p \preceq^+/\neg (S(A \land B)) \) iff \( p \preceq^+/\neg SA \) and \( p \preceq^+/\neg SB, SI \in \{\land, \lor\} \)
- \( p \preceq^+/\neg F(A \rightarrow B) \) iff \( p \preceq^+/\neg TA \) and \( p \preceq^+/\neg FB \)
- \( p \preceq^+/\neg T(A \rightarrow B) \) iff \( p \preceq^+/\neg FA \) and \( p \preceq^+/\neg TB \)
- \( p \preceq^+/\neg F\neg A \) iff \( p \preceq^+/\neg TA \)
- \( p \preceq^+/\neg T\neg A \) iff \( p \preceq^+/\neg FA \)
- \( p \preceq_w S \top \) and \( p \preceq_w S \bot \)
- \( p \preceq_w FA \) and \( p \preceq_w T\neg A \) for every \( A \)
- \( p \preceq_w Tq \) if \( q \) is a propositional variable different from \( p \)
- \( p \preceq_w (S(A \land B)) \) iff \( p \preceq_w TA \) and \( p \preceq_w TB, SI \in \{\land, \lor\} \)
- \( p \preceq_w T(A \rightarrow B) \) iff \( p \preceq_w TB \).

where \( S \in \{T,F\} \)

**Figure 4.10:** Polarities definition [FFF10]
signed formulas $S$ and returns true if and only if the smallest world in $K$ realizes $S$. $CM$ takes as input a set of signed formula $S$ and a possibly empty set of Kripke models $M$ and return a counter-model $K$ for $S$ combining the models in $M$.

Like PITP, FCUBE is also a theorem prover focused on research purposes, and it has been tested on the ILTP library ([ILT13], [ROK07]).

### 4.3 Imogen

Imogen is an example of a theorem prover that uses a forward chaining strategy. It has been developed by McLaughlin and Pfenning ([MP08]) at the Carnegie Mellon University in Pittsburgh. The main features of Imogen are focusing, polarization and the use of the inverse method.

Focusing is a technique that allows to reduce the proof search space. It consists in the chain-application of a series of non-invertible rules to a selected formula. Focusing is based on the fact that during a backward derivation it is possible to decompose certain connectives eagerly without affecting completeness. The implication is an example of such a connectives: to prove $A \rightarrow B$ it is sufficient to prove $B$ under to assumption $A$. This kind of connective is called a negative connective. In contrast to negative connectives there are the positive connectives such as $\lor$. In fact while proving $A \lor B$ a decision on which one between $A$ and $B$ must be proved has to be taken.

In intuitionistic logic the polarity of each connectives is not uniquely determined. In fact the connective $\land$ and the truth $\top$ can be either positive or negative. Figure 4.11 shows the polarities for each connective in intuitionistic propositional logic. Due to the fact that truth and conjunction may be both positive or negative, the translation of an unpolarized formula $A$ into a polarized formula is non-deterministic. The only constraint is $|A^-| = A$ where the operator $|A|$ is defined as shown in Figure 4.12.

Imogen works with sequents. In particular, due to focusing, it works with a special kind of sequents named stable sequents.

**Definition 5.** A sequent $\Sigma \rightarrow \gamma$ is called **stable** iff

- $\Sigma$ is either empty, contains only negative formulas and/or positive atoms,
\[ \begin{array}{ll}
| A^+ \lor B^+ | &= | A^+ | \lor | B^+ | \\
| A^+ \land B^+ | &= | A^+ | \land | B^+ | \\
| A^- \land B^- | &= | A^- | \land | B^- | \\
| A^+ \rightarrow B^- | &= | A^+ | \rightarrow | B^- | \\
| \bot | &= \bot \\
| \top | &= \top \\
| \uparrow A^+ | &= | A^+ | \\
| p^+ | &= p \\
| \downarrow A^- | &= | A^- | \\
| p^- | &= p \\
\end{array} \]

Figure 4.12: Translation from polarized to unpolarized formula [MP08]

- \( \gamma \) is either empty, contains only a positive formula or negative atoms.

Using focusing and polarized formula, Imogen derives new rules that have stables sequent as premises and as conclusions. Since focusing is complete these derived rules are sufficient to prove any stable sequents that are valid. Starting from a formula to be proven, it is possible to convert it into a set of stable sequents. Then, for each of these stable sequents, focusing on each constituent formula is applied, decomposing in until all stable sequents are reached. At this point a newly derived inference rule has been computed. This process lead to the generation of a set of new rules. These rules will constitute the calculus used as the basis of the inverse method.

The inverse method generalizes resolution to non-classical logics. Starting from a set of facts, represented by the sequent with empty premises, it generates all possible facts until either a proof is found or the set is saturated. Sometimes during the derivation of new sequents, a stronger version of the goal sequent is derived. To be able to recognize this situation Imogen checks if any new derived sequent subsumes the goal.

**Definition 6.** A sequent \( \Sigma \rightarrow \gamma \) subsumes \( \Sigma' \rightarrow \gamma' \) if \( \Sigma \subseteq \Sigma' \) and \( \gamma \subseteq \gamma' \).

The set of derived sequents is saturated if any sequent that can be derived is subsumed by a sequent that has already been derived. If the process saturates without deriving the goal sequent or a sequent that subsumes the goal, then the goal sequent cannot be proved and the set of the derived sequents can be considered as a sort of counter-model for the goal sequent.

The proof search procedure of Imogen works as follows. Starting from the goal formula \( A \), the translation into \( A^- \) is computed. Then the formulas is decomposed into stables sequents using invertible rules. For each stable sequent, a set of derived
rules will be generated using focusing. The derived rules will be used to generate new sequent until a proof of each stable sequent will be found or the process saturates. Due to the application of focusing, Imogen can be seen as a combination of backward and forward chaining strategy. Imogen has been used to prove that the inverse method is at a competitive level compared to the tableau method regarding intuitionistic propositional logic. It has been tested against some tableau based theorem prover such as PITP ([AFM06]) and FCUBE ([FFF10]) on the ILTP library ([ILT13], [ROK07]).
Chapter 5

MetTel2

In [ST09] Schmidt and Tishkovsky present a framework in which the generation of a tableau calculi for a set of logics can be done automatically starting from a semantic specification of the logic. This set of logics includes intuitionistic logic among others (modal logic, description logic and hybrid logic).

MetTel2 [TSK12] is a tool developed by R. Schmidt, D. Tishkovsky and M. Khodadadi in the School of Computer Science at the University of Manchester. Having in input the specification of a set of tableau rules, MetTel2 automatically generates a tableau based theorem prover. MetTel2 can generate provers for different logics, such as description logics, modal logics and hybrid logics [ST12].

MetTel2 can be seen as a step towards the implementation of a tool that automatically generates sound and complete tableau calculi for any logic, having in input just the specification of the formal semantics of the logic.

MetTel2 is written in Java and produces Java code for the generated provers. It is then possible for the expert user to modify the generated code in order to add additional features or modify the existing one. An example of such features can be the implementation of a different search strategy.

One of the main aims of MetTel2 is to provide an easy-to-use tool suitable not only for expert users but also for non-expert users. MetTel2 is the main tool used in the project.

In order to specify the tableau rules the user has to define the logical language first. The specification of the logical language in MetTel2 reflects the one discussed in [ST09]. It is based on a many-sorted object specification. The user has to specify how expression for each sort can be formed.

Figure 5.1 shows an example of a MetTel2 logical language specification file. The
specification example
syntax example {
  sort formula, element;
  formula true = 'true'
  formula false = 'false'
  formula negation = '∼' formula;
  formula conjunction = formula ' & ' formula;
  formula disjunction = formula ' | ' formula;
  formula implication = formula ' → ' formula;
}

Figure 5.1: MetTel2 logical language specification example

The first word “specification” is a keyword and it is used to specify the name of the language defined (“example” in this case). The block

```
"$syntax$ $example$ { ... }
```

consists of the definition of all the sorts declared. In the example only two sorts are defined. Since for the sort “element” there are no operators defined, any expression of the type “element” is considered atomic. Regarding the sort “formula”, several operators are defined: negation, conjunction, disjunction and implication. Moreover also true and false formulas are defined. These definitions specify how expressions of the sort “formula” can be built.

The specification of the tableau rules in MetTel2 extend the previous one implemented in MetTel ([TSK11]).

Figure 5.2 shows the tableau rules specification for classical propositional logic in MetTel2 syntax. Premises and conclusions are separated by the special character ‘/’. If the rule is a branching rule then the two branches in the conclusion are separated by ‘||’. In order to give the user more flexibility in implementing the prover, it is possible to specify the priority of each rule. By doing this, the search strategy will be affected. An higher value for the priority means that the rule has a lower precedence compared to other rules with a lower priority value. The last rule is a closure rule, since it contains empty conclusion.

In addition to the language specification file and the tableau rule file, the user can also specifies some additional properties that include the character to use to separate premises and conclusions of a rule, the character to separate different branches for a branching rule and the keyword to specify how to use rewriting (see below).

The generation of the prover work as it follows. MetTel2 uses the ANTLR ([ANT13])
A parser generator to parse the logical language specification file. The result is an abstract syntax tree, which is passed to the generator class that produces several files:

- Java classes representing the logical language defined by the user.
- An object factory class which creates the classes for the logical language.
- Java classes used to implement substitution and replacement.
- A ANTLR file for generating the parser for the logical language and the tableau rules.
- Java main class for parsing the command line command and starting the derivation process.
- JUnit classes for testing purposes.

Each expression of one sort is represented as a separated Java class. Each of these classes implements, among others, two methods. The first one takes an expression object as a parameter and it returns the substitution that unifies the parameter expression with the current expression. The second method returns an instance of the current expression after applying the substitution given as a parameter.

During the derivation process, each created expression is represented as a single object. Each tableau node is represented as an object that includes information about the formulas associated with the node in addition to methods for manipulating the formulas and to apply the tableau rules.

The application of tableau rules works as follows. A set of formulas associated with a tableau node is selected. These formulas are matched with the premises of every rule. Since matching is a computationally expensive operation, the system keeps track of all the substitutions obtained by matching every selected formula with the premises of a rule. If the selected rule completely matches the premises of a rule, the substitution computed is applied to the conclusion of the rule.

In MetTel2 blocking is realised through the unrestricted blocking mechanism presented in [ST07]. The conditions to apply the unrestricted blocking mechanism are the following.

- The logic admits the subexpression property: every expression in a derivation is a subexpression of the input.
- The logic admits the finite model property: if a formula $A$ is satisfiable, then it has a finite model.
The tableau calculus is sound and constructively complete.

MetTel2 implements the following features:

- Ordered backward and forward rewriting: to perform reasoning with respect to equalities that appear in a branch MetTel2 implements both ordered backward and forward rewriting. Each tableau node keeps track of a rewrite relation.

  When an equality expression is added to a branch, MetTel2 applies ordered backward rewriting. This causes the rewrite relation saved in the tableau node to be rebuilt. Then all the expressions contained in the node are rewritten using the new rewrite relation.

  In addition, from this point onwards, ordered forward rewriting is applied. This means that all the new expressions that will be added to the node, will be rewritten according to the saved rewrite relation.

  Equality expression can be specified in MetTel2 using one of the keyword “equivalence” or “equality”. To be able to perform reasoning with equality using rewriting, the user has to specify an operator in the syntax specification of the logic.

- Dynamic backtracking: To reduce the search space during the derivation dynamic backtracking is used. In order to avoid the application of the same rule in parallel branches, MetTel2 keeps track of the rule application common to the branches.

- Conflict directed backjumping: MetTel2 discard branches that contains the same conflict sets. This also helps to reduce the search space during a derivation.
Chapter 6

Implementation of Fitting’s systems

6.1 Basic Fitting’s system

The first theorem prover implemented reflects the tableau rule presented in 3.1.1 and listed in Figure 3.3. Figure 3.4 shows the closure rules. This calculus is the most basic tableau calculus for intuitionistic propositional logic. It represents the direct translation of the logical syntax into tableau rules.

6.1.1 Specification

Figure 6.1 shows the specification input file used to generate the prover. Two sorts have been defined. The sort “individual” is used to represent Kripke worlds. The two operators defined specify how to construct formulas of this sort. In particular they specify how to construct the successor worlds for implication and negation using two Skolem functions (f and g).

The main sort is the sort “formula”. For this sort a series of operators have been defined. Each of them specify how to construct formulas using the logical connectives. In particular

- The operator false specify how to construct a false constant, represented by false,
- The operator trueValue specify how to construct a \( T \)-signed formula, by concatenating \( T \) and the formula,
- The operator falseValue specify how to construct a \( F \)-signed formula, by concatenating \( F \) and the formula,
specification Int;
syntax Int {
  sort formula, individual;
  formula false = 'false';
  formula trueValue = 'T' formula;
  formula falseValue = 'F' formula;
  formula at = '@' individual formula;
  formula negation = '˜' formula;
  formula conjunction = formula ' & ' formula;
  formula disjunction = formula ' | ' formula;
  formula implication = formula ' → ' formula;
  individual successorImp = 'f' '(' individual ',' formula ',' formula ')';
  individual successorNot = 'g' '(' individual ',' formula ')';
  formula relation = 'R' '(' individual ',' individual ')';
  formula equality = '[ ' individual '=' individual ' ]';
}

Figure 6.1: Specification input file for the basic Fitting’s system implementation.

- The operator at specify how to identify formulas that holds in a specific Kripke’s worlds by using the character ‘@’ followed by an individual and a formula,
- The operator negation specify how to construct the negation of a formula, by concatenating the character ∼ and the formula,
- The operator conjunction specify how to construct the conjunction of two formulas, by adding the character & between the two formulas,
- The operator disjunction specify how to construct the disjunction of two formulas, by adding the character | between the two formulas,
- The operator implication specify how to construct the implication of two formulas, by adding the character → between the two formulas,
- The accessibility relation $R$ is used to define the semantics of negation and implication. It is specified as an operator relation applied to two individuals returning a formula. More intuitive would be to think of $R$ as a predicate
- the operator equality specify how to compare two Kripke’s world. It is used to implement the blocking mechanism.

6.1.2 Tableau rules

Figure 6.2 shows the encoding of the Fitting’s tableau calculus in MetTel2 syntax. Using the operator defined in the specification file, the rules for each connective are
encoded. It is important to notice how $S_T$ is translated. In [Fit69] Fitting says that before applying the rule relative to $F(\neg P)$ and $F(P \rightarrow Q)$ all the $F-$signed formulas have to be deleted. It is possible to implement the deletion of formulas in MetTel2 using rewriting. The resulting rules would be

\[
\begin{align*}
@l(F(\neg P)) & @l(F Z) / R(l, g(l, P)) @g(l, P)(T P) \\
\quad & (@l (F Z) \leftrightarrow \text{dummy}) \text{ priority 10;} \\
@l(F(P \rightarrow Q)) & @l(F Z) / R(l, f(l, P, Q)) @f(l, P, Q)(T P) \\
\quad & @f(l, P, Q)(F Q) (@l (F Z) \leftrightarrow \text{dummy}) \text{ priority 10;} \\
\end{align*}
\]

Here $\leftrightarrow$ has to be defined as a rewriting operator in the property file. The application of these two rules will cause the rewrite relation in node $l$ to be updated. All the $F-$signed formulas will be rewritten to dummy. By doing this, it will be not possible for the prover to use any $F$-signed formulas to perform any inference step, obtaining the same effect as deleting them. Some experiments have been done using this “deletion rule” to verify the soundness and completeness of the calculus. The results show that by deleting all the $F-$signed formulas unsatisfiable sets of formulas will become satisfiable. As a simple example consider the following set of signed formulas:

$$S = \{F(A \rightarrow A), F(A \rightarrow B)\}$$

The only rule applicable to $S$ is

\[
\begin{align*}
@l(F(P \rightarrow Q)) & @l(F Z) / R(l, f(l, P, Q)) @f(l, P, Q)(T P) \\
\quad & @f(l, P, Q)(F Q) (@l (F Z) \leftrightarrow \text{dummy}) \text{ priority 10;} \\
\end{align*}
\]

Now depending on which formula will be selected for the application of the rule, the result will be different. On the one hand if $F(A \rightarrow A)$ is selected, then the $F(A \rightarrow B)$ will be rewritten as dummy and both $(T A)$ and $(F A)$ will be derived, obtaining a contradiction. On the other hand if $F(A \rightarrow B)$ is selected, then $F(A \rightarrow A)$ will be rewritten to dummy and both $(T A)$ and $(F B)$ will be derived. In this case there is no contradiction and the prover will output “satisfiable”. This example shows that the “deletion” rule leads to incorrect results. The conclusion obtained is that implementing Fitting’s rules literally was a mistake. Rather, a solution is to say that only $T$-signed formulas propagate to the successor world. In order to ensure that, the skolem functions
\( f(l, P, Q) \) and \( g(l, P) \) are used. They create the successor worlds of \( l \). This, combined with the rule

\[
@l(T \ P) \ R(l, l_0) / @l_0(T \ P) \text{ priority 4 ;}
\]

will have the effect of propagating only the \( T \)-signed formulas from \( l \) to any successor of \( l \), included the one created using the two Skolem functions. In effect the \( F \)-signed formulas in the new world are now 'deleted'.

The rules

\[
R(l_0, l_1) \quad R(l_1, l_2) / R(l_0, l_2) \text{ priority 4;}
\]

\[
R(l_0, l_1) \quad R(l_1, l_0) / [l_0 = l_1] \text{ priority 4;}
\]

represent transitivity and reflexivity properties for the accessibility relation between Kripke’s worlds.

As previously said blocking is realised using the unrestricted blocking mechanism [ST07]. This is achieved by adding the following rule to the calculus.

\[
@l \ P \ @l_0 \ Q / [l = l_0] \quad \mid \mid (F([l = l_0])) \text{ priority 9;}
\]

The rule will attempt to set the worlds \( l \) and \( l_0 \) equals every time two formulas appear in the two worlds. If a model is not find on the left branch, then the right branch of the conclusion will be explored. The priority of the blocking rule is set as lower with respect to all the other rules with the exception of the rule that create a successor. This is done just for performance purposes. Before to create a new Kripke’s world the prover will try to find a model with the world’s already created.

Another important feature is how to express the semantic interpretation of negation and implication. On the one hand recall that if \( \neg P \) holds in a world \( l \) then in all the accessible worlds from \( l \) \( P \) must not hold. On the other hand if \( P \rightarrow Q \) holds in \( l \) then either \( P \) must not hold in every accessible world from \( l \) or \( Q \) must hold in all the accessible worlds from \( l \). This is realised in MetTel2 using the expression \( R(l, l_0) \) in the premises of the rule. For example the rule
@l(T(\sim P)) R(l,l0) / @l0(F P) \quad \text{priority 4;}

specifies that if \sim P holds in l and there is a world l0 accessible from l, then P must not hold ((F P)) in l0. The rule

@l(T(\sim P)) R(l,l0) @l0(T P) / \text{priority 2;}

is a closure rule that attempts to improve the performance of the prover by looking at all the successors of a world l in which \sim P holds trying to find a world in which P holds. If this is the case the tableau branch is closed. It is important to notice that the rule above is optional. It can be removed without affecting soundness and completeness. The rules

@l(T \text{false}) / \text{priority 0;}

and

@l(T P) @l(F P) / \text{priority 1;}

are the two closure rules that ensure to close a tableau branch if a contradiction is reached. The former is applied every time the constant formula ‘false’ is T-signed, while the latter is applied every time the same formula appear in a Kripke’s world both T-signed and F-signed.

The priority of each rule has been selected in order to give precedence to the rules that have only one branch in the conclusion. This helps to reduce the search space.

6.2 Fitting’s system with derivable rules

One of the main problems affecting the performance of a tableau based theorem prover is the number of branches created during the derivation. Rules such as
Figure 6.2: Tableau rule MetTel2 specification for the calculus of Figure 3.3.
create two different branches after being applied. As presented in [ST09] it is possible for intuitionistic propositional logic to refine some tableau rules. The refinement reduce the number of branches of a rule by adding more premises to the rule itself. Having a sound and constructively complete tableau calculus $T$ it is possible to refine some of the rule appearing in the calculus and obtain a new calculus $T'$ that is sound and constructively complete. What the rule refinement does is to move formulas from the conclusions to the premises of a rule, in negated form, reducing the number of branches in the conclusion.

The calculus presented in 6.3 has been obtained by adding 3 rules to the calculus of 3.3. These 3 rules ($F \land \text{NEW}$, $T \lor \text{NEW}$ and $T \rightarrow \text{NEW}$) are derivable rule so adding them to the calculus don’t affect soundness and completeness. They can be seen as the refined version of the correspondent rules.
Figure 6.4: Specification input file for the Fitting’s system with derivable rules implementation.

6.2.1 Specification

Figure 6.4 shows the modified specification input file. It is exactly the same as the one used to generate the basic version of Fitting’s system without any derivable rule (see Section 6.1.1).

6.2.2 Tableau rules

Figure 6.5 shows the encoding of the calculus of Figure 6.3 into MetTel2 syntax. This calculus implements the derivable rules regarding disjunction, conjunction and implication. It is important to notice that even in the presence of these rules, the original ones are still present in the calculus with a lower priority.

The higher priority will force the prover to use the derivable rules as soon as possible. It is not possible to remove the original rules from the calculus since this will lead to incorrect answers as showed by the following example.

Let the theorem $S = ((A \rightarrow A) \land (A \rightarrow A))$. Using the calculus of Figure 6.3 it is possible to obtain a closed tableau, starting from $F((A \rightarrow A) \land (A \rightarrow A))$. The only rule applicable at this stage is the not refined rule $F\land$. This will lead to two different branches, each of them containing $F(A \rightarrow A)$. At this point both the branches will be closed by the application of the rule $F \rightarrow$ that will cause the contradiction $TA, FA$. If the original rule $F\land$ is removed the first step of the above proof cannot be performed. Since no other rules are applicable the prover will output “satisfiable”. This simple
example shows that in order to ensure soundness of the calculus, both the derivable and the original rule for conjunction have to be present. The same consideration can be applied to disjunction and implication.

An advantage of having these derivable rules would be that, thanks to the priority setting, each time it is possible to apply this rule instead of the original one it will be applied. This will produce less branching, reducing the search space.

The priority of all other rules with the exception of the derivable rules is the same as the previous version of the generated prover (see Section 6.1).
Chapter 7

Implementation of PITP systems and its variations

7.1 PITP theorem prover

In Section 4.1 the PITP theorem prover has been presented. The calculus implemented in PITP is shown in Figure 4.3 and Figure 4.4. Since this calculus is an enhancement of the one proposed by Fitting, it is interesting to see how the modifications implemented affect the performance of the theorem prover. For this reason a theorem prover implementing the PITP calculus has been generated using MetTel2.

7.1.1 Specification

Figure 7.1 shows the specification input file used to generate the prover. The improvement over the previous specifications are represented by a new set of Skolem functions and the specification of a new sign.

The specification for the new sign $F_C$ will allow the prover to handle formulas of the kind $F_C A$

The new set of Skolem functions has been introduced in order to reflect the behaviour of all the rules that create a new Kripke’s world in PITP. These rules are the ones in which the conclusion only contains $T$ – signed and $F_C$ – signed formulas. Every time a new world is created, a different Skolem function is used, depending on the rule that cause the creation of the world. For example the rule

$$\frac{S, F_C (\neg A)}{S_C, TA}$$
will create a new Kripke’s world. In the MetTel2 language, this will be represented using the Skolem function ‘m’ with the expression $m(l,A)$. Moreover 2 new operators for the sort ‘prop’ have been defined (’y’ and ’u’). These are two Skolem functions that are used to create a new atom as required by the application of the rules $T \rightarrow \rightarrow$ and $T \rightarrow \lor$ (see Figure 4.4).

7.1.2 Tableau rules

Figure 7.2 shows the MetTel2 implementation of the PITP theorem prover calculus. Compared to the previous calculi there are more rules, due to the splitting of the implication rule to avoid duplications (see Section 4.1).

A new closure rule regarding the sign $F_C$ has been specified. This will cause a tableau branch to be closed every time $TA$ and $F_CA$ will appear, for every formula $A$.

Moreover, in order to represent the behaviour of the sign $F_C$ a new rule to propagate $F_C - signed$ formulas has been added. This will have the effect of reproduce the set $S_C$. 

Figure 7.1: Specification input file for PITP theorem prover implementation.

```plaintext
specification Int;
syntax Int {
    sort formula, individual, prop;
    formula false = 'false';
    formula trueValue = 'T' formula;
    formula falseValue = 'F' formula;
    formula certainFalse = 'Fc' formula;
    formula at = '@' individual formula;
    formula atom = 'n' prop;
    prop newatomor = 'y' '(' individual ',' formula ',' formula ',' formula ')';
    prop newatomimp = 'u' '(' individual ',' formula ',' formula ',' formula ')';
    formula negation = '¬' formula;
    formula conjunction = formula '&' formula;
    formula disjunction = formula '|' formula;
    formula implication = formula '→' formula;
    individual successorImp = 'f' '(' individual ',' formula ',' formula ')';
    individual successorNot = 'g' '(' individual ',' formula ')';
    individual successorAnd = 'h' '(' individual ',' formula ',' formula ')';
    individual successorFcimp = 'j' '(' individual ',' formula ',' formula ')' formula 'j';
    individual successorFcNot = 'm' '(' individual ',' formula ')';
    individual successorImpneg = 'n' '(' individual ',' formula ',' formula ')';
    individual successorImpimp = 'z' '(' individual ',' formula ',' formula ', formula ')';
    formula relation = 'R' '(' individual ',' individual ')';
    formula equality = '[ ' individual '=' individual ' ]';
}
```
in the conclusion of the rules in PITP system.

The priority of each rule has been set using the group-schema presented in Section 4.1. Recall that for performance reasons, the priority of the blocking rule has been set to an higher value compared to all rules that creates a new Kripke’s world.

7.2 PITP theorem prover with 2-signs variation

One of the main concept introduced in the tableau system that is the basis of PITP is the sign $F_C$. As previously stated, this sign can be replaced by $T \neg$ (see Section 4.1). In order to see how the introduction of a new sign affects the performance of the theorem prover, a version of the PITP theorem prover with only 2 signs has been implemented.

7.2.1 Specification

The specification file to generate the prover is presented in Figure 7.3. The only difference between this specification and the one used for the PITP theorem prover implementation (see Figure 7.1) is the absence of the declaration of the operator to construct $F_C$-signed formulas.

7.2.2 Tableau rules

Figure 7.4 shows the encoding of the tableau rules for this version of the generated theorem prover. The sign $F_C$ has been substituted with $T \sim$. Moreover, the rule that in Figure 7.2 introduces the sign $F_C$, in this version of the calculus only introduce an $F$-signed formula. This should have an impact on the performance of the prover since there is a big difference in having an $F$-signed formula and and $F_C$-signed formula in term of provability of the formula itself (recall that an the sign $F$ gives no information about the provability of a formula).

7.3 PITP theorem prover with additional implication rules

The last theorem prover implemented has been inspired by [FFF08] where Ferrari et. all presents a tableau system for intuitionistic propositional logic.
Figure 7.2: Tableau rule MetTel2 specification for the calculus of Figure 4.3 and Figure 4.4.
specification Int;
syntax Int {
  sort formula, individual, prop;
  formula false = 'false';
  formula trueValue = 'T' formula;
  formula falseValue = 'F' formula;
  formula at = '@' individual formula;
  formula atom = '#' prop;
  prop newatomor = 'y' '(' individual ',' formula ',' formula ',' formula ')';
  prop newatomimp = 'u' '(' individual ',' formula ',' formula ',' formula ')';
  formula negation = '˜' formula;
  formula conjunction = formula '∧' formula;
  formula disjunction = formula '∨' formula;
  formula implication = formula '→' formula;
  individual successorImp = 'f' '(' individual ',' formula ',' formula ')';
  individual successorNot = 'g' '(' individual ',' formula ')';
  individual successorAnd = 'h' '(' individual ',' formula ',' formula ')';
  individual successorFcimp = 'j' '(' individual ',' formula ',' formula ')';
  individual successorFcNot = 'm' '(' individual ',' formula ')';
  individual successorImpneg = 'n' '(' individual ',' formula ',' formula ')';
  individual successorImpimp = 'z' '(' individual ',' formula ',' formula ','
                              formula ')';
  formula relation = 'R' '(' individual ',' individual ')';
  formula equality = ['( individual '=' individual ')'];
}

Figure 7.3: Specification input file for the PITP theorem prover with 2-sign variation implementation.
<table>
<thead>
<tr>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@l(T false) / priority 0;</td>
</tr>
<tr>
<td>@l(T P) @l(F P) / priority 1;</td>
</tr>
<tr>
<td>@l(T P) @l(Fc P) / priority 1;</td>
</tr>
<tr>
<td>@l(T(P</td>
</tr>
<tr>
<td>@l(F(P</td>
</tr>
<tr>
<td>@l(F(P</td>
</tr>
<tr>
<td>@l(T ~ (P</td>
</tr>
<tr>
<td>@l(T(P &amp; Q)) / @l(T P) @l(T Q) priority 3;</td>
</tr>
<tr>
<td>@l(F(P &amp; Q)) / @l(F P)</td>
</tr>
<tr>
<td>@l(T ~ (P &amp; Q)) / R(l,h(l,P,Q)) @h(l,P,Q)(T ~ (P))</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>@l(T((P)) R(l,10) / @l0(F P) priority 3;</td>
</tr>
<tr>
<td>@l(F((P))) / R(l,g(l,P)) @g(l,P)(T P) priority 9;</td>
</tr>
<tr>
<td>@l(T ((\sim) (P))) / R(l, m(l,P)) @m(l,P)(T P) priority 14;</td>
</tr>
<tr>
<td>@l(T(# P -&gt; Q)) @l(T(# P)) / @l(T(# P)) @l(T Q) priority 3;</td>
</tr>
<tr>
<td>@l(F(P -&gt; Q)) / R(l,f(l,P,Q)) @f(l,P,Q)(T P) @f(l,P,Q)(F Q) priority 9;</td>
</tr>
<tr>
<td>@l(T ~ (P -&gt; Q)) / R(l, j(l,P,Q)) @j(l,P,Q)(T P) @j(l,P,Q)(T ~ Q)) priority 14;</td>
</tr>
<tr>
<td>@l(T((P &amp; Q) -&gt; Z)) / @l(T(P -&gt; (Q -&gt; Z))) priority 3;</td>
</tr>
<tr>
<td>@l(T(P</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>@l(T((P -&gt; Q) -&gt; Z)) / R(l,z(l,P,Q,Z)(T P)</td>
</tr>
<tr>
<td>@z(l,P,Q,Z)(T(# u(l,P,Q,Z))) @z(l,P,Q,Z)(T(# u(l,P,Q,Z) -&gt; Z))</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>@l(T((\sim) P -&gt; Q)) / R(l,n(l,P,Q)) @n(l,P,Q)(T P)</td>
</tr>
<tr>
<td>@l P / R(l,1) priority 3;</td>
</tr>
<tr>
<td>R(l,0,1) R(l,1,2) / R(l,0,2) priority 4;</td>
</tr>
<tr>
<td>R(l,1,1) R(l,1,0) / [l=1] priority 4;</td>
</tr>
<tr>
<td>@l(T P) R(l,10) / @l0(T P) priority 4;</td>
</tr>
<tr>
<td>@l P @l0 Q / [l = 10]</td>
</tr>
<tr>
<td>(F([l = 1]) / priority 0;</td>
</tr>
</tbody>
</table>

**Figure 7.4:** Tableau rule MerTel2 specification for the 2 sign version of the calculus of Figure 4.3 and Figure 4.4.
The main feature of this calculus is the presence of a new set of rules for the formulas of the kind $T((A \rightarrow B) \rightarrow C)$ depending on the main connective of $B$. Having a rule for each connective of $B$ has the effect to reduce the non-determinism during the derivation, because, depending on the structure of the formula to be proved, just one rule would be applicable. Instead of having just one rule for $T \rightarrow \rightarrow$ like the previous implemented theorem provers, here there are 5 different rules that are applicable depending on the structure of the formula.

For example suppose that during the derivation the formula $T((A \rightarrow \neg B) \rightarrow C)$ is derived. With the calculus of Figure 7.2 the only applicable rule is $T \rightarrow \rightarrow$ that produces the following inference step.

$T((A \rightarrow \neg B) \rightarrow C)$

$TA, Fq, T(q \rightarrow C), T(\neg B \rightarrow q) || TC$

Having the set of rules of Figure 7.5 it is possible to apply the rule $T \rightarrow \rightarrow \neg$, obtaining

$T((A \rightarrow \neg B) \rightarrow C)$

$TA, TB || TC$

In the left branch of the conclusion, using the latter rule there are only 2 signed formulas, while using the former lead to produce 4 signed formulas.

### 7.3.1 Specification

Figure 7.6 shows the specification input file to generate this version of the theorem prover. Since the rules $T \rightarrow \rightarrow \land$, $T \rightarrow \rightarrow \lor$ and $T \rightarrow \rightarrow \rightarrow$ introduce a new atom
in the conclusions, 3 new operators for the sort ‘prop’ have been specified (u1, u2 and u3). These have the same effect of the operators y and u. They create a new atom using a Skolem function.

Moreover, since all the rule of Figure 7.5 create a new Kripke’s world, 5 new operators for the sort ‘individual’ have been specified (z2, z3, z4, z5 and z6) in order to have 5 new operators that specify how to construct a new Kripke’s world depending on the rule applied.

### 7.3.2 Tableau rules

Figure 7.7 shows the MetTel2 encoding of the tableau calculus implemented in the theorem prover. This system differs from the one presented in [FFF08]. In fact the following rule is not implemented

\[
S, T (A \rightarrow B) \quad \frac{}{S, F_C A | S, TB} T \rightarrow \text{certain}
\]

This rule, as stated in [FFF08] is only applicable if \( S = S_C \) or, in other words, if there are no \( F \)-signed formulas in the current Kripke’s world. At the moment it is not possible to implement this condition in MetTel2. This is because the premises of the rule in MetTel2 are existentially quantified. This means that in order to apply a rule, the prover tries to match all the formulas in a branch with the premise of each rule. If a match is found then the rule can be applied and the formulas in the conclusion are added to the branch (without loss of generality here only the rule with one branch in the conclusion has been considered). Taking this into consideration, in MetTel2 it is not possible to know when all the formulas in one branch are only \( T \)-signed formulas or \( F_C \)-signed formulas. Hence this rule cannot be specified using the MetTel2 language.

Another difference between the calculus of [FFF08] and the one implemented is represented by the following rule.

\[
S, TA, T (A \rightarrow B) \quad \frac{}{S, TA, TB} T \rightarrow MP
\]

Instead of having the Modus Ponens rule, in the implemented system there is an atomic version of it. The application of the rule \( T \rightarrow MP \) is restricted only to formulas \( TA \) and \( T (A \rightarrow B) \), in which \( A \) is an atom. This can be done thanks to the set of rules for \( T (A \rightarrow B) \) implemented in the system that cover all the possible cases depending on the structure of \( A \) (See Figure 7.7). The new set of rules introduced have the same
specification Int;
syntax Int {
    sort formula, individual, prop;
    formula false = 'false';
    formula trueValue = 'T' formula;
    formula falseValue = 'F' formula;
    formula at = '@' individual formula;
    formula atom = '# prop;
    prop newatomor = 'y' '(' individual ',', formula ',', formula ')' formula;
    prop newatomimp = 'u' '(' individual ',', formula ',', formula ')' formula;
    prop newatomimp1 = 'u1' '(' individual ',', formula ',', formula ')' formula;
    prop newatomimp2 = 'u2' '(' individual ',', formula ',', formula ')' formula;
    prop newatomimp3 = 'u3' '(' individual ',', formula ',', formula ')' formula;
    formula negation = '˜' formula;
    formula conjunction = formula ' & ' formula;
    formula disjunction = formula ' | ' formula;
    formula implication = formula ' → ' formula;
    individual successorImp = 'f' '(' individual ',', formula ',', formula ')' formula;
    individual successorNot = 'g' '(' individual ',', formula ')' formula;
    individual successorAnd = 'h' '(' individual ',', formula ',', formula ')' formula;
    individual successorFcImp = 'j' '(' individual ',', formula ',', formula ')' formula;
    individual successorFcNot = 'm' '(' individual ',', formula ')' formula;
    individual successorImpneg = 'n' '(' individual ',', formula ')' formula;
    individual successorImpimp = 'z' '(' individual ',', formula ',', formula ',
                                    formula ')' formula;
    individual successorImpimp2 = 'z2' '(' individual ',', formula ',', formula ',
                                         formula ')' formula;
    individual successorImpimp3 = 'z3' '(' individual ',', formula ',', formula ',
                                         formula ')' formula;
    individual successorImpimp4 = 'z4' '(' individual ',', formula ',', formula ',
                                        formula ')' formula;
    individual successorImpimp5 = 'z5' '(' individual ',', formula ',', formula ',
                                        formula ')' formula;
    individual successorImpimp6 = 'z6' '(' individual ',', formula ',', formula ',
                                        formula ')' formula;
    formula relation = 'R' '(' individual ',', individual ')' formula;
    formula equality = '( individual ^= individual )';
}

Figure 7.6: Specification input file for the PITP theorem prover with additional implication rules implementation.
priority of the rule $T \rightarrow \rightarrow$ in Figure 7.2. This has been done for two reasons. On the one hand, the priority of this rule has to be lower than the blocking rule for performance reasons. On the other hand, this new set of rule substitutes the rule $T \rightarrow \rightarrow$ in the PITP theorem prover implementation. Apart from the new rules for double implications, all other rules are the same as the ones implemented in Section 7.1.
@l(T false) / priority 0;
@l(T P) @l(F P) / priority 1;
@l(T P) @l(Fc P) / priority 1;

@l(T(P | Q)) / @l(T P) | | @l(T Q) priority 5;
@l(F(P | Q)) / @l(F P) priority 3;
@l(F(P | Q)) / @l(F Q) priority 3;
@l(Fc(P | Q)) / @l(Fc P) @l(Fc Q) priority 3;

@l(T(P & Q)) / @l(T P) @l(T Q) priority 3;
@l(F(P & Q)) / @l(F P) | | @l(F Q) priority 5;
@l(Fc(P & Q)) / R(l.h(l.P,Q)) @h(l.P,Q)(Fc P) | | R(l.h(l.P,Q)) @h(l.P,Q)(Fc Q) priority 13;

@l(T( ~ P)) R(l,10) / @l0(Fc P) priority 3;
@l(F( ~ P)) / R(l,g(l.P)) @g(l.P)(T P) priority 9;
@l(Fc( ~ P)) / R(l, m(l.P)) @m(l.P)(T P) priority 14;

@l(T( # P → Q)) @l(T( # P)) / @l(T( # P)) @l(T Q) priority 3;
@l(F(P → Q)) / R(l.f(l.P,Q)) @f(l.P,Q)(T P) @f(l.P,Q)(F Q) priority 9;
@l(Fc(P → Q)) / R(l,j(l.P,Q)) @j(l.P,Q)(T P) @j(l.P,Q)(Fc Q) priority 11;
@l(T(P & Q) → Z) / @l(T(P → (Q → Z))) priority 3;
@l(T(P | Q) → Z) / @l(T(P → # y(l.P,Q,Z))) @l(T(Q → # y(l.P,Q,Z)))
@l(T # y(l.P,Q,Z) → Z) priority 3;
@l(T( ~ P → Q)) / R(l,n(l.P,Q)) @n(l.P,Q)(T P) | | @l(T Q) priority 12;
@l(T(P → # Q) → Z)) / R(l.z2(l.P,Q,Z)) @z2(l.P,Q,Z)(T P)
@z2(l.P,Q,Z)(F( # Q)) @z2(l.P,Q,Z)(T( # Q → Z)) | | @l(T Z) priority 11;
@l(T(P → ~ Q) → Z)) / R(l.z3(l.P,Q,Z)) @z3(l.P,Q,Z)(T P)
@z3(l.P,Q,Z)(T Q) | | @l(T Z) priority 11;
@l(T((P → (Q & Z)) → W)) / R(l.z4(l.P,Q,Z,W)) @z4(l.P,Q,Z,W)(T P)
@z4(l.P,Q,Z,W)(F( # u1(l.P,Q,Z,W)))
@z4(l.P,Q,Z,W)(F( # u1(l.P,Q,Z,W)))
@z4(l.P,Q,Z,W)(T(Q → Z → # u1(l.P,Q,Z,W)))
@z4(l.P,Q,Z,W)(T(Q → Z → # u1(l.P,Q,Z,W)))
@l(T((P → (Q | Z)) → W)) / R(l.z5(l.P,Q,Z,W)) @z5(l.P,Q,Z,W)(T P)
@z5(l.P,Q,Z,W)(F( # u2(l.P,Q,Z,W)))
@z5(l.P,Q,Z,W)(F( # u2(l.P,Q,Z,W)))
@z5(l.P,Q,Z,W)(T(Z → # u2(l.P,Q,Z,W)))
@z5(l.P,Q,Z,W)(T(Z → # u2(l.P,Q,Z,W)))
@z5(l.P,Q,Z,W)(T(Z → # u2(l.P,Q,Z,W)) → W)) | | @l(T W) priority 11;

@l(T((P → (Q → Z)) → W)) / R(l.z6(l.P,Q,Z,W))
@z6(l.P,Q,Z,W)(T P) @z6(l.P,Q,Z,W)(T Q)
@z6(l.P,Q,Z,W)(F( # u3(l.P,Q,Z,W)))
@z6(l.P,Q,Z,W)(F( # u3(l.P,Q,Z,W)))
@z6(l.P,Q,Z,W)(T(Z → # u3(l.P,Q,Z,W)))
@z6(l.P,Q,Z,W)(T(Z → # u3(l.P,Q,Z,W)) → W)) | | @l(T W) priority 11;

@l P / R(l,l) priority 3;
R(0,11) R(1,12) / R(10,12) priority 4;
R(0,11) R(1,10) / [l=11] priority 4;
@l(T P) R(l,10) / @l0(T P) priority 4;
@l P @l0 Q / [l=10] | | (F(l=10)) priority 8;
(F(l=10)) / priority 0;

Figure 7.7: Tableau rule MetTel2 specification for the PITP theorem prover with additional implication rules implementation.
Chapter 8

Benchmark

Problems coming from real world applications such as hardware or software verification problems always involve formulas with thousands of propositions. Hence theorem provers need not only to be sound and complete but also fast. In order to evaluate the performance of the theorem provers generated by MetTel2, a series of benchmark formulas have been collected. For this purpose the ILTP library ([ROK07], citeILTP) has been used.

ILTP provides about 2800 problems from 24 different domains. Each problem has his status (theorem, non-theorem, unsolved) and a difficulty rating.

The ranking is based on a scale between 0.0 and 1.0. A value of 0.0 means that every state-of-the-art theorem prover for intuitionistic propositional logic can solve the problem, while a value of 1.0 indicates that there is no theorem prover that can solve the correspondent problem.

The propositional part of ILTP is made by 274 problems. Among these 128 are theorems, 109 are non-theorems and 37 are unsolved problems.

The domain for the propositional part of ILTP are: logic calculi (LCL), syntactic (SYN) and intuitionistic syntactic (SYJ). Table 8.1 shows a summary of all the propositional problems inside ILTP together with their difficulty rating.

Problems in ILTP are represented in a standard syntax. Figure 8.1 shows an example of the ILTP syntax. Hence, a translation from the standard ILTP format to MetTel2 syntax is needed. In order to achieve this a tool called tptp2X [Tpt13] has been used.

Tptp2x is a utility written in Prolog that converts problem from ILTP format to an user-defined format. It was originally designed to translate the problems contained in TPTP [Sut09] and it has been adapted to ILTP. Tptp2x takes in input the file containing the original problem and the file containing the specification of the target format and
output a file containing the problem translated. Since ILTP is widely used to test almost every theorem prover for intuitionistic logic (see [ILT13]) there are several available file containing the Prolog instruction to translate the formulas. However no one of these format is the same as the MetTel2 one. So it has been necessary to modify one of the available format files to obtain the MetTel2 format.

### 8.1 LCL problems set

The LCL problems only comprises 2 formulas: LCL181 and LCL230. Both the formulas are non-theorem. LCL181 was judged by Siklossy et al. in [SRM73] as the hardest of the first 52 theorems in [WR27]. LCL230 was judged by Siklossy et al. in [SRM73] as the hardest of the first 67 theorems in [WR27]. Both the formula are non-theorem in intuitionistic propositional logic.
8.2 SYJ problems set

The SYJ problems set includes 19 problems family, some of them containing up to 20 instances in increasing size. Some problems inside the SYJ domain have a particular structure, as it follows.

- SYJ202: formulas inside this family represent the encoding of Cook pigeon-hole problem.
- SYJ204: formulas inside this family can be decided very fast, but with some theorem prover they cause space problems.
- SYJ206: formulas inside this family involve a lot of double implication.

Table 8.2 shows for each SYJ problem family the intuitionistic status (theorem or non-theorem). It is important to notice that the non-theorem families are linked to the theorem families in the following way.

- The SYJ207 formulas have the same structure of the SYJ201 formulas.
- The SYJ208 formulas have the same structure of the SYJ202 formulas.
- The SYJ209 formulas have the same structure of the SYJ203 formulas.
- The SYJ210 formulas have the same structure of the SYJ204 formulas.
- The SYJ211 formulas have the same structure of the SYJ205 formulas.
- The SYJ212 formulas have the same structure of the SYJ206 formulas.

8.3 SYN problems set

The SYN problems set include 19 formulas. Table 8.3 shows the intuitionistic status for each formula.
<table>
<thead>
<tr>
<th>Problems</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYJ101</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ102</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ103</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ104</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ105</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ106</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ107</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ201</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ202</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ203</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ204</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ205</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ206</td>
<td>theorem</td>
</tr>
<tr>
<td>SYJ207</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYJ208</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYJ209</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYJ210</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYJ211</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYJ212</td>
<td>non-theorem</td>
</tr>
</tbody>
</table>

Table 8.2: Status for each SYJ problems family
<table>
<thead>
<tr>
<th>Problems</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN001</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN040</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN041</td>
<td>theorem</td>
</tr>
<tr>
<td>SYN044</td>
<td>theorem</td>
</tr>
<tr>
<td>SYN045</td>
<td>theorem</td>
</tr>
<tr>
<td>SYN046</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN047</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN387</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN388</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN389</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN390</td>
<td>theorem</td>
</tr>
<tr>
<td>SYN391</td>
<td>theorem</td>
</tr>
<tr>
<td>SYN392</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN393</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN416</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN915</td>
<td>theorem</td>
</tr>
<tr>
<td>SYN916</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN977</td>
<td>non-theorem</td>
</tr>
<tr>
<td>SYN978</td>
<td>theorem</td>
</tr>
</tbody>
</table>

Table 8.3: Status for each SYN problems family
Chapter 9

Experimental results

All the generated prover have been tested on ILTP library [ILT13]. All the experiments have been run on a XEON E5530 2.4Ghz CPU with 8192 KB of cache and 8 GB RAM. The timeout time for each instance of each problem has been set to 600 seconds.

In addition to the prover described in Chapter 6 and Chapter 7, in order to see how the blocking mechanism affects the performance of a theorem prover, for each generated prover, a version implementing a ‘restricted’ blocking mechanism has been generated and tested. Instead of using the rule

\[
\text{@}l \ P \ @\text{l0} \ Q \ / \ [l = \text{l0}] \ | \ | \ (F([l = \text{l0}])) \quad \text{priority 8;}
\]

the following rule is used.

\[
R(l,\text{l0})/ [l = \text{l0}] \ | \ | \ (F([l = \text{l0}])) \quad \text{priority 8;}
\]

Instead of attempting to set the worlds \(l\) and \(\text{l0}\) equals every time two formulas appear on the two worlds, this latter rule will set the worlds \(l\) and \(\text{l0}\) equals for every worlds accessible from world \(l\). If a model is not find on the left branch, then the right branch of the conclusion is explored.

An advantage of having the unrestricted blocking rule would be that, considering that it is applied for every world, it should find smaller models (if there is one). This should results in better performance on satisfiable formulas for the theorem provers based on the unrestricted blocking mechanism. Since the application of the blocking
rule is restricted only to accessible world, it will be applied fewer times compared to the unrestricted version.

Table 9.1 and Table 9.2 show the result for the provers that implement the unrestricted blocking mechanism while Table 9.3 and Table 9.4 show the result for the prover that implement the restricted blocking mechanism. In particular

- Fitv1 represents the Fitting’s basic system implementation (see Section 6.1).
- Fitv2 represents Fitting’s system with derivable rules (see Section 6.2)
- PITPv1 represents the PITP theorem prover implementation (see Section 7.1).
- PITPv2 represents the PITP theorem prover with 2-signs variation implementation (see Section 7.2).
- PITPv3 represents the PITP theorem prover with additional implication rules implementation (see Section 7.3).

For each problems family (see Section 8) the time to solve the largest instance that the prover can handle together with the instance number is reported. Additional information are provided as explained below.

- A * indicates that on the next instance, a memory error occurs.
- A ** indicates that on the next instance, a timeout error occurs.
- A *** indicates that on the next instance, the result the prover output is incorrect.

Regarding the Fitting’s basic system the experiments show that the version implementing the restricted blocking mechanism perform better on most of the benchmark formulas. For formulas that are theorem this is expected, since using the restricted blocking rule lead to less branching and hence a minor number of branches have to be closed during the proof. In contrast, from a theoretical point of view, the unrestricted blocking mechanism should perform better on non-theorem formulas since a prover that implement this kind of blocking, eagerly try to collapse all the Kripke’s world and this produces small models.

As the results show the version implementing the unrestricted blocking is faster than the version implementing restricted blocking only on 5 benchmark problems of which only 2 are non-theorem (SYJ210 and SYN916). A significant difference between the two versions is visible in the SYJ204 problem.

The version implementing the unrestricted blocking can solve one more instance compared to the version implementing the restricted blocking mechanism (13 against 12).
As expected the ‘restricted blocking’ version outperform the ‘restricted blocking’ one in proving formulas that are theorem. For example, it can solve all the 4 instance of the SYJ105 problems family.

Regarding the SYJ206 family, both the version can solve up to the 3rd instance (the ‘restricted blocking’ version is faster). On the one hand the ‘unrestricted blocking’ version run out of time while trying to prove the 4th instance. On the other hand the ‘restricted blocking’ version gives the wrong answer.

To understand the nature of this bug a new experiment has been done. One of the feature of MetTel2 that can cause incorrect answers is rewriting. Since rewriting is used to perform equality reasoning, by removing the rule for equality reasoning and blocking, rewriting will not be used during the derivation. Since the all the instance of the SYJ206 problems family are theorem this will not affects the result because it will still be possible to close each branch of the tableau. After implementing the changes to avoid the use of rewriting, the prover implementing Fitting’s basic system can solve correctly the 4th instance of SYJ206. This shows that there is some problem caused by the use of rewriting.

An improvement over the Fitting’s basic system is represented by Fitting’s system with derivable rules implementation. As for the Fitting’s basic system implementation, the ‘restricted blocking’ version perform better than the ‘unrestricted blocking’ one on most of the benchmark formulas. The ‘restricted blocking’ version implementation failed to prove correctly the 5th instance of the SYJ202 problems family even without using rewriting.

Regarding the comparison between the two different implementation of Fitting’s system, the results show, adding the derivable rules to the Fitting’s basic system have some very noticeable impacts on the performance of the theorem prover. In particular on the family SYJ204 and SYJ210 (these two family are linked, see Section 8) both the timing and the number of instances that can be solved improved. For the SYJ204 family, instead of solving only 13 instances in almost 10 minutes (using the unrestricted blocking mechanism), the version with the derivable rules can solve all the 20 instance in less than 0.5s.

This shows that the use of derivable rules produces less branching and, especially using the unrestricted blocking rule, this is a great improvement. In fact, since the unrestricted blocking rule is applied very often, having less branching is crucial to reduce the number of applications of the rule.

The performance of the two versions of the PITP theorem prover implementation
reflect the pattern seen for the Fitting’s basic system implementations. The ‘restricted blocking’ version is generally faster than the ‘unrestricted blocking’ one. The SYJ207 problem family is an exception. While the ‘restricted blocking’ version can solve only one instance of this problem, the ‘unrestricted blocking’ version can solve up to the 5th instance. This is the first case in which the unrestricted blocking mechanism outperforms the restricted one. It is important to notice that the difficulty rating of the formula SYJ207.5 is 0.75. This means that 3/4 of the provers that have been tested on ILTP cannot solve this formula.

As expected the ‘restricted blocking’ version performs better on theorem formulas such as the SYJ205 problems family. The ‘restricted blocking’ version can solve two instance of this problem, while the ‘unrestricted blocking’ one can only solve the first instance. The ‘restricted blocking’ version performs better also on the SYJ211 problems family (recall that, as explained in Section 8 these two family are linked). It can solve all the 20 instances, compared to only 8 of the ‘unrestricted blocking’ version.

Regarding the SYN problems set, a very noticeable improvement has been observed on the SYN393 formula. While the ‘unrestricted blocking’ version can prove the formula in 198.67 seconds the ‘restricted blocking’ one only takes 0.42 seconds. Taking into consideration that the SYN393 formula is refutable, this is not expected from a theoretical point of view.

Compared to the Fitting’s system with the derivable rules, the PITP theorem prover implementation performs better in term of number of instances that can be solved. Both Fitting’s systems cannot solve any instance of the SYJ205 problems family while the PITP theorem prover implementation can solve up to the 2nd instance.

The biggest improvement is represented by the SYJ211 problems family. On the one hand, both Fitting’s system cannot solve any instance of this family. On the other hand, the ‘restricted blocking’ version of the PITP theorem prover implementation can solve all the 20 instances in 15.32 seconds. Regarding the SYN problems set, while all the Fitting’s systems cannot solve the SYN393 problem, the PITP theorem prover can solve it in less than 1 second.

Another significant improvement achieved by the PITP theorem prover implementation is represented by the SY212 problems family. In fact both the ‘restricted blocking’ and the ‘unrestricted blocking’ versions can solve up to 14 instance of this family compared to only 3 instance of the ‘restricted blocking’ version of the Fitting’s system implementations.
The only problems family on which the PITP theorem prover implementation performance are really poor compared to the Fitting’s system implementation is SYJ204. Here, after the 9th instance, the PITP theorem prover implementation gives the wrong answer, even avoiding the use of rewriting. It seems that the size of the problem cause some bugs during the proof of a formula that is a theorem. In fact, even for very big instances, if the formula to be proven is not a theorem, the prover works correctly (see for example SYJ210 or SYJ209). At the moment the reason why this happens is still not clear and more investigation is needed.

Apart from this, the performance of the PITP theorem prover implementation shows that avoiding duplication (see Section 4.1) has a strong impact on the performance of the theorem prover, especially on the attempt to find a model.

- SYJ207, SYJ209, SYJ211 and SYJ212 for the ‘unrestricted blocking’ version.
- SYJ205, SYJ209 and SYJ212 for the ‘restricted blocking’ version.

This shows that reducing the non-determinism that occurs during the derivation has a positive effect on the performance of the theorem prover.

The PITP theorem prover with 2-signs variation has been implementated to see how the sign $F_C$ affects the performance of the theorem prover.

On the one hand, as the results show, this version of the prover perform slightly better than the version with 3 sign on provable formulas, in particular there is a big improvement on the SYJ105 and the SYJ203 families (for the latter only regarding the ‘unrestricted blocking’ versions).

On the other hand the PITP theorem prover implementation performs better on refutable formulas, especially regarding the ‘unrestricted blocking’ versions. For example, while the version with the $F_C$ sign can solve up to 8 instance of the SYJ211 problems family, the 2-signs version can only solve 2 instance of this family. Regarding the ‘restricted blocking’ versions of both the theorem provers the number of instance that the provers can solve is similar.

As for the previous theorem prover implementations the ‘restricted blocking’ version generally performs better than the ‘unrestricted blocking’ one with the only significant exceptions of the family SYJ203, SYJ205, SYJ207 and SYJ208. On these families the ‘unrestricted blocking’ version can solve more instances compared to the ‘restricted blocking’ one (SYJ203 and SYJ207) or is significantly faster (SYJ205, SYJ208).
<table>
<thead>
<tr>
<th>Problem</th>
<th>Fitv1</th>
<th>Fitv2</th>
<th>PITPv1</th>
<th>PITPv2</th>
<th>PITPv3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCL181</td>
<td>0.42s</td>
<td>0.49s</td>
<td>0.48s</td>
<td>0.50s</td>
<td>0.59s</td>
</tr>
<tr>
<td>LCL230</td>
<td>0.30s</td>
<td>0.30s</td>
<td>0.45s</td>
<td>0.52s</td>
<td>0.51s</td>
</tr>
<tr>
<td>SYJ101</td>
<td>0.15s</td>
<td>0.15s</td>
<td>0.20s</td>
<td>0.20s</td>
<td>0.22s</td>
</tr>
<tr>
<td>SYJ102</td>
<td>0.20s</td>
<td>0.26s</td>
<td>0.25s</td>
<td>0.26s</td>
<td>0.30s</td>
</tr>
<tr>
<td>SYJ103</td>
<td>0.15s</td>
<td>0.16s</td>
<td>0.21s</td>
<td>0.20s</td>
<td>0.22s</td>
</tr>
<tr>
<td>SYJ104</td>
<td>0.15s</td>
<td>0.16s</td>
<td>0.20s</td>
<td>0.21s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYJ105</td>
<td>1.06s (3)*</td>
<td>1.07s (3)*</td>
<td>0.30s (2)*</td>
<td>66.76s (4)</td>
<td>15.14s (3)**</td>
</tr>
<tr>
<td>SYJ106</td>
<td>memory</td>
<td>memory</td>
<td>memory</td>
<td>memory</td>
<td>memory</td>
</tr>
<tr>
<td>SYJ107</td>
<td>0.21s</td>
<td>0.20s</td>
<td>0.26s</td>
<td>0.26s</td>
<td>0.26s</td>
</tr>
<tr>
<td>SYJ201</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>SYJ202</td>
<td>5.94s (3)*</td>
<td>0.72s (3)*</td>
<td>9.10s (3)**</td>
<td>9.05s (3)**</td>
<td>10.46s (3)**</td>
</tr>
<tr>
<td>SYJ203</td>
<td>0.31s (1)*</td>
<td>0.32s (1)*</td>
<td>6.79s (2)*</td>
<td>106.5s (3)**</td>
<td>6.23s (2)*</td>
</tr>
<tr>
<td>SYJ204</td>
<td>559.15s (13)**</td>
<td>0.47s (20)</td>
<td>0.36s (9)***</td>
<td>0.35s (9)***</td>
<td>0.37s (9)***</td>
</tr>
<tr>
<td>SYJ205</td>
<td>memory</td>
<td>memory</td>
<td>2.82s (1)*</td>
<td>2.37s (2)*</td>
<td>2.84s (1)*</td>
</tr>
<tr>
<td>SYJ206</td>
<td>1.98s (3)*</td>
<td>2.17s (3)*</td>
<td>205.15s (3)*</td>
<td>197.01s (3)**</td>
<td>240.63s (3)*</td>
</tr>
<tr>
<td>SYJ207</td>
<td>0.57s (1)*</td>
<td>0.63s (1)*</td>
<td>21.15s (5)**</td>
<td>20.45s (5)**</td>
<td>16.32s (5)**</td>
</tr>
<tr>
<td>SYJ208</td>
<td>5.25s (4)*</td>
<td>3.67s (4)*</td>
<td>7.41s (4)*</td>
<td>7.77s (4)*</td>
<td>8.08s (4)*</td>
</tr>
<tr>
<td>SYJ209</td>
<td>3.24s (20)</td>
<td>3.59s (20)</td>
<td>327.05s (20)</td>
<td>337.17s (20)</td>
<td>2.38s (20)</td>
</tr>
<tr>
<td>SYJ210</td>
<td>217.95s (12)**</td>
<td>3.86s (20)</td>
<td>1.13s (20)</td>
<td>1.28s (20)</td>
<td>1.24s (20)</td>
</tr>
<tr>
<td>SYJ211</td>
<td>timeout</td>
<td>timeout</td>
<td>584.45s (8)**</td>
<td>2.57s (2)*</td>
<td>0.87s (20)</td>
</tr>
<tr>
<td>SYJ212</td>
<td>0.81s (2)*</td>
<td>0.96s (2)*</td>
<td>146.54s (14)**</td>
<td>160.13s (14)**</td>
<td>571.63s (16)**</td>
</tr>
</tbody>
</table>

Table 9.1: Unrestricted blocking provers comparison on the LCL and SYJ problems domains
<table>
<thead>
<tr>
<th>Problem</th>
<th>Fitv1</th>
<th>Fitv2</th>
<th>PITPv1</th>
<th>PITPv2</th>
<th>PITPv3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN001</td>
<td>0.31s</td>
<td>0.31s</td>
<td>0.37s</td>
<td>0.42s</td>
<td>0.36s</td>
</tr>
<tr>
<td>SYN040</td>
<td>0.61s</td>
<td>0.62s</td>
<td>0.62s</td>
<td>0.62s</td>
<td>0.66s</td>
</tr>
<tr>
<td>SYN041</td>
<td>0.71s</td>
<td>0.87s</td>
<td>0.87s</td>
<td>0.92s</td>
<td>0.96s</td>
</tr>
<tr>
<td>SYN044</td>
<td>1.42s</td>
<td>1.92s</td>
<td>0.77s</td>
<td>0.66s</td>
<td>0.86s</td>
</tr>
<tr>
<td>SYN045</td>
<td>0.25s</td>
<td>0.25s</td>
<td>0.31s</td>
<td>0.26s</td>
<td>0.30s</td>
</tr>
<tr>
<td>SYN046</td>
<td>0.30s</td>
<td>0.30s</td>
<td>0.30s</td>
<td>0.31s</td>
<td>0.35s</td>
</tr>
<tr>
<td>SYN047</td>
<td>0.41s</td>
<td>0.48s</td>
<td>0.62s</td>
<td>0.61s</td>
<td>0.67s</td>
</tr>
<tr>
<td>SYN387</td>
<td>0.20s</td>
<td>0.20s</td>
<td>0.26s</td>
<td>0.25s</td>
<td>0.30s</td>
</tr>
<tr>
<td>SYN388</td>
<td>0.31s</td>
<td>0.31s</td>
<td>0.36s</td>
<td>0.41s</td>
<td>0.37s</td>
</tr>
<tr>
<td>SYN389</td>
<td>0.25s</td>
<td>0.25s</td>
<td>0.32s</td>
<td>0.31s</td>
<td>0.35s</td>
</tr>
<tr>
<td>SYN390</td>
<td>0.15s</td>
<td>0.15s</td>
<td>0.22s</td>
<td>0.22s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYN391</td>
<td>1.24s</td>
<td>1.39s</td>
<td>1.83s</td>
<td>1.64s</td>
<td>1.89s</td>
</tr>
<tr>
<td>SYN392</td>
<td>0.35s</td>
<td>0.38s</td>
<td>0.48s</td>
<td>0.46s</td>
<td>0.37s</td>
</tr>
<tr>
<td>SYN393</td>
<td>memory</td>
<td>memory</td>
<td>198.67s</td>
<td>182.96s</td>
<td>memory</td>
</tr>
<tr>
<td>SYN416</td>
<td>0.31s</td>
<td>0.36s</td>
<td>0.41s</td>
<td>0.41s</td>
<td>0.40s</td>
</tr>
<tr>
<td>SYN915</td>
<td>0.16s</td>
<td>0.15s</td>
<td>0.21s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYN916</td>
<td>0.15s</td>
<td>0.15s</td>
<td>0.20s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYN977</td>
<td>0.26s</td>
<td>0.26s</td>
<td>0.30s</td>
<td>0.30s</td>
<td>0.35s</td>
</tr>
<tr>
<td>SYN978</td>
<td>0.36s</td>
<td>0.40s</td>
<td>0.47s</td>
<td>0.47s</td>
<td>0.69s</td>
</tr>
</tbody>
</table>

Table 9.2: Unrestricted blocking provers comparison on the SYN problems domain
As observed for the PITP theorem prover implementation, the ‘restricted blocking’ version of the theorem prover can solve the SYN393 problem in less than 1 seconds, compared to 182.96 seconds for the ‘unrestricted blocking’ version.

The PITP theorem prover with additional implication rules is the last theorem prover implemented.

As before, the ‘restricted blocking’ version performs better than the ‘unrestricted blocking’ one on most of the benchmark problems. The exception are represented by the families SYJ207, SYJ211 and SYJ212. On these families the ‘restricted blocking’ version can solve significantly more instance (5 against 1 for the SYJ207 family, 16 against 14 for the SYJ212 family) or is significantly faster (less than 1 second to solve all the 20 instances for the SYJ211 family compared to 18.08 seconds). The ‘restricted blocking’ version is also faster on the SYJ203 problems family.

The 16th instance of the SYJ212 family solved by this version of the theorem prover is best result achieved on this problems family and, considering the fact that it has a difficulty rank of 0.5, it can be seen as a good improvement over the previous versions.

Compared to all the other theorem provers generated, the PITP theorem prover with additional implication rules implementation achieve the best performance on several families.

In order to see how the provers generated with MetTel2 perform compared to others existing provers for intuitionistic propositional logic, Table 9.5 and Table 9.6 show the performance of PITP, ft-Prolog and ft-C respectively on the ILTP benchmark library.

While the provers presented in Chapter 4 have been implemented specifically for intuitionistic propositional logic, ft-Prolog and ft-C are two version of a first order theorem prover implementing several optimization techniques ([SFH92]). Table 9.8 and Table 9.9 show the performance of FCUBE and Imogen on some SYJ problems.

As the results show these optimized provers generally perform better than any MetTel2 implemented prover. In particular on the SYJ problems set, PITP is the fastest prover. The fact that these provers are significantly faster than any prover generated in MetTel2 is not unexpected. This is because they are prover highly optimized for intuitionistic propositional logic while MetTel2 can generate provers for any logic specified by the user.

Optimization techniques play a very important role. One of the most effective optimization is Simplification, implemented in both PITP and FCUBE (see Chapter 4). Table 9.7 shows the performance of PITP on the SYJ problems set without the
<table>
<thead>
<tr>
<th>Problem</th>
<th>Fitv1</th>
<th>Fitv2</th>
<th>PITPv1</th>
<th>PITPv2</th>
<th>PITPv3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCL181</td>
<td>0.25s</td>
<td>0.36s</td>
<td>0.48s</td>
<td>0.41s</td>
<td>0.57s</td>
</tr>
<tr>
<td>LCL230</td>
<td>0.18s</td>
<td>0.32s</td>
<td>0.37s</td>
<td>0.35s</td>
<td>0.35s</td>
</tr>
<tr>
<td>SYJ101</td>
<td>0.20s</td>
<td>0.15s</td>
<td>0.20s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYJ102</td>
<td>0.14s</td>
<td>0.25s</td>
<td>0.25s</td>
<td>0.25s</td>
<td>0.25s</td>
</tr>
<tr>
<td>SYJ103</td>
<td>0.13s</td>
<td>0.16s</td>
<td>0.20s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYJ104</td>
<td>0.13s</td>
<td>0.15s</td>
<td>0.20s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYJ105</td>
<td>3.01s (4)</td>
<td>3.94s (4)</td>
<td>0.25s (2)**</td>
<td>9.95s (4)</td>
<td>8.34s (3)**</td>
</tr>
<tr>
<td>SYJ106</td>
<td>memory</td>
<td>memory</td>
<td>memory</td>
<td>memory</td>
<td>memory</td>
</tr>
<tr>
<td>SYJ107</td>
<td>0.36s</td>
<td>0.21s</td>
<td>0.27s</td>
<td>0.26s</td>
<td>0.26s</td>
</tr>
<tr>
<td>SYJ201</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>SYJ202</td>
<td>4.36s (3)**</td>
<td>0.74s (3)***</td>
<td>9.05s (3)**</td>
<td>8.96s (3)**</td>
<td>10.25s (3)**</td>
</tr>
<tr>
<td>SYJ203</td>
<td>2.62s (2)*</td>
<td>2.48s (2)*</td>
<td>7.17s (2)**</td>
<td>6.74s (2)**</td>
<td>8.18s (2)**</td>
</tr>
<tr>
<td>SYJ204</td>
<td>196.70s (12)*</td>
<td>0.47s (20)</td>
<td>0.40s (9)***</td>
<td>0.35s (9)***</td>
<td>0.37s (9)***</td>
</tr>
<tr>
<td>SYJ205</td>
<td>memory</td>
<td>timeout</td>
<td>69.46s (2)**</td>
<td>65.06s (2)**</td>
<td>58.04s (2)**</td>
</tr>
<tr>
<td>SYJ206</td>
<td>0.71s (3)***</td>
<td>1.07s (3)**</td>
<td>3.18s (3)*</td>
<td>3.03s (3)**</td>
<td>3.42s (3)*</td>
</tr>
<tr>
<td>SYJ207</td>
<td>0.40s (1)**</td>
<td>0.80s (1)*</td>
<td>1.39s (1)*</td>
<td>1.32s (1)*</td>
<td>3.84s (1)*</td>
</tr>
<tr>
<td>SYJ208</td>
<td>2.84s (4)*</td>
<td>2.08s (4)*</td>
<td>5.95s (4)*</td>
<td>8.55s (4)*</td>
<td>6.70s (4)*</td>
</tr>
<tr>
<td>SYJ209</td>
<td>2.30s (20)</td>
<td>2.62s (20)</td>
<td>310.42s (20)</td>
<td>306.36s (20)</td>
<td>2.03s (20)</td>
</tr>
<tr>
<td>SYJ210</td>
<td>218.88s (12)**</td>
<td>1.67s (20)</td>
<td>0.83s (20)</td>
<td>0.92s (20)</td>
<td>0.95s (20)</td>
</tr>
<tr>
<td>SYJ211</td>
<td>timeout</td>
<td>memory</td>
<td>15.32s (20)</td>
<td>24.20s (20)</td>
<td>18.08s (20)</td>
</tr>
<tr>
<td>SYJ212</td>
<td>13.85s (3)**</td>
<td>1.77s (3)**</td>
<td>131.06s (14)**</td>
<td>135.85s (14)**</td>
<td>115.30s (14)**</td>
</tr>
</tbody>
</table>

Table 9.3: Restricted blocking provers comparison on the LCL and SYJ problems domains
<table>
<thead>
<tr>
<th>Problem</th>
<th>Fitv1</th>
<th>Fitv2</th>
<th>PITPv1</th>
<th>PITPv2</th>
<th>PITPv3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN001</td>
<td>0.24s</td>
<td>0.31s</td>
<td>0.39s</td>
<td>0.47s</td>
<td>0.36s</td>
</tr>
<tr>
<td>SYN040</td>
<td>0.37s</td>
<td>0.51s</td>
<td>0.31s</td>
<td>0.32s</td>
<td>0.57s</td>
</tr>
<tr>
<td>SYN041</td>
<td>0.57s</td>
<td>0.81s</td>
<td>0.57s</td>
<td>0.87</td>
<td>0.57s</td>
</tr>
<tr>
<td>SYN044</td>
<td>0.41s</td>
<td>0.51s</td>
<td>0.31s</td>
<td>0.31s</td>
<td>0.35s</td>
</tr>
<tr>
<td>SYN045</td>
<td>0.21s</td>
<td>0.26s</td>
<td>0.31s</td>
<td>0.25s</td>
<td>0.30s</td>
</tr>
<tr>
<td>SYN046</td>
<td>0.20s</td>
<td>0.26s</td>
<td>0.31s</td>
<td>0.26s</td>
<td>0.31s</td>
</tr>
<tr>
<td>SYN047</td>
<td>0.31s</td>
<td>0.42s</td>
<td>0.32s</td>
<td>0.31s</td>
<td>0.36s</td>
</tr>
<tr>
<td>SYN387</td>
<td>0.15s</td>
<td>0.20s</td>
<td>0.21s</td>
<td>0.20s</td>
<td>0.25s</td>
</tr>
<tr>
<td>SYN388</td>
<td>0.21s</td>
<td>0.32s</td>
<td>0.37s</td>
<td>0.46s</td>
<td>0.36s</td>
</tr>
<tr>
<td>SYN389</td>
<td>0.20s</td>
<td>0.25s</td>
<td>0.26s</td>
<td>0.25s</td>
<td>0.30s</td>
</tr>
<tr>
<td>SYN390</td>
<td>0.15s</td>
<td>0.15s</td>
<td>0.21s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYN391</td>
<td>0.41s</td>
<td>0.57s</td>
<td>0.62s</td>
<td>0.57s</td>
<td>0.67s</td>
</tr>
<tr>
<td>SYN392</td>
<td>0.21s</td>
<td>0.34s</td>
<td>0.32s</td>
<td>0.31s</td>
<td>0.36s</td>
</tr>
<tr>
<td>SYN393</td>
<td>memory</td>
<td>memory</td>
<td>0.42s</td>
<td>0.46s</td>
<td>50.01</td>
</tr>
<tr>
<td>SYN416</td>
<td>0.23s</td>
<td>0.27s</td>
<td>0.33s</td>
<td>0.30s</td>
<td>0.30s</td>
</tr>
<tr>
<td>SYN915</td>
<td>0.15s</td>
<td>0.15s</td>
<td>0.22s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYN916</td>
<td>0.18s</td>
<td>0.16s</td>
<td>0.21s</td>
<td>0.20s</td>
<td>0.20s</td>
</tr>
<tr>
<td>SYN977</td>
<td>0.16s</td>
<td>0.21s</td>
<td>0.26s</td>
<td>0.25s</td>
<td>0.26s</td>
</tr>
<tr>
<td>SYN978</td>
<td>0.21s</td>
<td>0.31s</td>
<td>0.32s</td>
<td>0.30s</td>
<td>0.35s</td>
</tr>
</tbody>
</table>

Table 9.4: Restricted blocking provers comparison on the SYN problems domain
implementation of Simplification. It is possible to see that the performances of the
prover dramatically decrease on most of the problems families, especially regarding
refutable formulas. Having Simplification helps in reducing the number of formula to
be considered and hence a model can be found in less time.

At the moment in MetTel2 it is not possible to implement Simplification, because
of the rule of Figure 4.5. The only way to implement these rules in MetTel2 is using
rewriting, but there are some problems as the following examples show. The encoding of the
rule

\[
\frac{S, TA}{S[A/\top], TA}
\]

in MetTel2 would be

\[
\emptyset (T \ A) / (\emptyset (T \ A) \leftrightarrow \top)
\]

The problem is that the \( TA \) formula in the conclusion of the rule is lost because rewriting will be applied to any formula of the kind \( TA \). There is no way to replace just all
the occurrence of \( TA \) in \( S \). The encoding of the rule

\[
\frac{S, FA}{S\{A/\bot\}, FA}
\]

in MetTel2 would be

\[
\emptyset (F \ A) / (\emptyset (F \ A) \leftrightarrow \bot)
\]

Here, in addition to the problem mentioned for the previous rule, there is also the issue
related to the different kind of substitution. While every occurrence of \( TA \) in \( S \) can be
replaced by \( \top \), the substitution of \( FA \) must proceed only until \( \neg \) or \( \rightarrow \) is found. Using
rewriting there is no way to specify this condition.

Taking into consideration that Simplification is not implemented in any provers
generated with MetTel2 it is interesting to see how they perform against PITP without Simplification.
All the version implemented perform better than PITP without Simplification on the SYJ209 problems family. Moreover, on this family, the provers generated with MetTel2 are among the fastest theorem provers (together with FCUBE and Imogen, that is the faster on this particular family) and perform significantly better than PITP even with Simplification. The PITP theorem prover implementation and its variations (using the unrestricted blocking mechanism) perform better than PITP without Simplification on the SYJ207 family. The PITP theorem prover with additional implication rules implementation (using the unrestricted blocking mechanism) and all the PITP theorem prover implementation variations (using the restricted blocking mechanism) perform better than PITP (with and without Simplification) on the SYJ211 family.

It’s interesting to notice that all the problems on which the provers generated with MetTel2 are faster than PITP are refutable problems and that Simplification has the greatest impact on these kind of problems. This suggest that the performance of the provers generated with MetTel2 would speedup significantly using Simplification.

In contrast PITP requires significantly less time and resources (memory) to reach a contradiction and this can explain the very strong performance of PITP on provable formulas, even without Simplification.

The fact that PITP and all the optimized existing provers can handle in a better way the memory is also visible by looking at the kind of error generated by these theorem prover when a formula cannot be proved. No one of these errors is memory related, they are all related to the timeout setting.

To summarize and to have a better view of the effect of different tableau rules specifications on the performance of a theorem prover, Table 9.10 and Table 9.11 show the number of ILTP problems solved by each version of the generated theorem provers. It is possible to see the difference with respect to the existing optimized provers (see Table 9.12).
<table>
<thead>
<tr>
<th>Problem</th>
<th>PITP</th>
<th>ft-Prolog</th>
<th>ft-c</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCL181</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>LCL230</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYJ101</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYJ102</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYJ103</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYJ104</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYJ105</td>
<td>&lt;0,01s (4)</td>
<td>&lt;0,01s (4)</td>
<td>&lt;0,01s (4)</td>
</tr>
<tr>
<td>SYJ106</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYJ107</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYJ201</td>
<td>0,01s (20)</td>
<td>0,37s (20)</td>
<td>0,05s (20)</td>
</tr>
<tr>
<td>SYJ202</td>
<td>50,25s (9)**</td>
<td>516,56 (7)**</td>
<td>76,30s (7)**</td>
</tr>
<tr>
<td>SYJ203</td>
<td>&lt;0,01s (20)</td>
<td>&lt;0,01s (20)</td>
<td>&lt;0,01s (20)</td>
</tr>
<tr>
<td>SYJ204</td>
<td>&lt;0,01s (20)</td>
<td>&lt;0,01s (20)</td>
<td>&lt;0,01s (20)</td>
</tr>
<tr>
<td>SYJ205</td>
<td>&lt;0,01s (20)</td>
<td>60,26s (8)*</td>
<td>85,84s (9)**</td>
</tr>
<tr>
<td>SYJ206</td>
<td>3,92s (20)</td>
<td>144,50s (10)**</td>
<td>481,98s (11)**</td>
</tr>
<tr>
<td>SYJ207</td>
<td>75,95s (6)**</td>
<td>358,05s (7)**</td>
<td>51,13s (7)**</td>
</tr>
<tr>
<td>SYJ208</td>
<td>0,19s (20)</td>
<td>65,41s (8)**</td>
<td>81,41s (17)**</td>
</tr>
<tr>
<td>SYJ209</td>
<td>222,45s (10)**</td>
<td>543,09s (10)**</td>
<td>96,99s (10)**</td>
</tr>
<tr>
<td>SYJ210</td>
<td>&lt;0,01s (20)</td>
<td>&lt;0,01s (20)</td>
<td>&lt;0,01s (20)</td>
</tr>
<tr>
<td>SYJ211</td>
<td>223,15s (20)</td>
<td>66,62s (4)**</td>
<td>17,25s (4)**</td>
</tr>
<tr>
<td>SYJ212</td>
<td>&lt;0,01s (20)</td>
<td>0,01s (20)</td>
<td>&lt;0,01s (20)</td>
</tr>
</tbody>
</table>

Table 9.5: PITP, ft-Prolog and ft-C comparison on the LCL and SYJ problems domains ([ILT13])
### Table 9.6: PITP, ft-Prolog and ft-C comparison on the SYN problems domain ([ILT13], [AFM08])

<table>
<thead>
<tr>
<th>Problem</th>
<th>PITP</th>
<th>ft-Prolog</th>
<th>ft-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN001</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN040</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN041</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN044</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN045</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN046</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN047</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN387</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN388</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN389</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN390</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN391</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN392</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN393</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN416</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN915</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN916</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN977</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
<tr>
<td>SYN978</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
<td>&lt;0,01s</td>
</tr>
</tbody>
</table>

### Table 9.7: Performance of PITP without Simplification on the SYJ problems domain [AFM08]

<table>
<thead>
<tr>
<th>Problem</th>
<th>PITP-noSimp</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYJ201</td>
<td>0,01s (20)</td>
</tr>
<tr>
<td>SYJ202</td>
<td>19,90s (6)**</td>
</tr>
<tr>
<td>SYJ203</td>
<td>0,01s (20)</td>
</tr>
<tr>
<td>SYJ204</td>
<td>0,01s (20)</td>
</tr>
<tr>
<td>SYJ205</td>
<td>0,01s (20)</td>
</tr>
<tr>
<td>SYJ207</td>
<td>11,31s (4)**</td>
</tr>
<tr>
<td>SYJ208</td>
<td>428,73s (8)**</td>
</tr>
<tr>
<td>SYJ209</td>
<td>357,65s (10)**</td>
</tr>
<tr>
<td>SYJ210</td>
<td>0,01s (20)</td>
</tr>
<tr>
<td>SYJ211</td>
<td>597,27s (20)</td>
</tr>
<tr>
<td>SYJ212</td>
<td>0,28s (20)</td>
</tr>
</tbody>
</table>
### Table 9.8: FCUBE performance on some SYJ problems [FFF10]

<table>
<thead>
<tr>
<th>Problem</th>
<th>FCUBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYJ201</td>
<td>22.34s (20)</td>
</tr>
<tr>
<td>SYJ202</td>
<td>529.36s (8)**</td>
</tr>
<tr>
<td>SYJ206</td>
<td>&lt;0.01s (20)</td>
</tr>
<tr>
<td>SYJ207</td>
<td>7.44s (20)</td>
</tr>
<tr>
<td>SYJ208</td>
<td>472.07s (17)**</td>
</tr>
<tr>
<td>SYJ209</td>
<td>0.09s (20)</td>
</tr>
</tbody>
</table>

### Table 9.9: Imogen performance on some SYJ problems [MP08]

<table>
<thead>
<tr>
<th>Problem</th>
<th>Imogen</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYJ201</td>
<td>28.35s (20)</td>
</tr>
<tr>
<td>SYJ202</td>
<td>64.06s (7)**</td>
</tr>
<tr>
<td>SYJ205</td>
<td>0.01s (20)</td>
</tr>
<tr>
<td>SYJ206</td>
<td>42.07s (20)</td>
</tr>
<tr>
<td>SYJ207</td>
<td>97.25s (20)</td>
</tr>
<tr>
<td>SYJ208</td>
<td>506.02s (20)</td>
</tr>
<tr>
<td>SYJ209</td>
<td>0.02s (20)</td>
</tr>
<tr>
<td>SYJ211</td>
<td>0.02s (20)</td>
</tr>
<tr>
<td>SYJ212</td>
<td>0.04s (20)</td>
</tr>
</tbody>
</table>

### Table 9.10: Number of ILTP problems solved by the ‘restricted blocking’ version of generated prover

<table>
<thead>
<tr>
<th></th>
<th>Fitv1</th>
<th>Fitv2</th>
<th>PITPv1</th>
<th>PITPv2</th>
<th>PITPv3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems solved</td>
<td>88</td>
<td>104</td>
<td>125</td>
<td>128</td>
<td>127</td>
</tr>
</tbody>
</table>

### Table 9.11: Number of ILTP problems solved by the ‘unrestricted blocking’ version of generated prover

<table>
<thead>
<tr>
<th></th>
<th>Fitv1</th>
<th>Fitv2</th>
<th>PITPv1</th>
<th>PITPv2</th>
<th>PITPv3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems solved</td>
<td>86</td>
<td>101</td>
<td>116</td>
<td>114</td>
<td>130</td>
</tr>
</tbody>
</table>

### Table 9.12: Number of ILTP problems solved by other existing provers

<table>
<thead>
<tr>
<th></th>
<th>PITP</th>
<th>Imogen</th>
<th>Ft-Prolog</th>
<th>Ft-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems solved</td>
<td>238</td>
<td>261</td>
<td>188</td>
<td>199</td>
</tr>
</tbody>
</table>

81
Chapter 10

Conclusion

The aim of this project was to undertake an empirical investigation on how different tableau rules specifications affect the performance of a theorem prover generated with MetTel2.

As the results in Chapter 9 show, the specification of the tableau rules has a very noticeable impact on the performance of the theorem prover. Adding a few more rules in a tableau system can have a big impact, as the difference between the Fitting’s basic system and the Fitting’s system with derivable rules show.

The experiments show also that specify the same set of rules in different ways may decrease or increase the performance of a theorem prover. This cause the difference in performance between the PITP theorem prover and the PITP theorem prover with 2-signs variations where the sign $F_C$ was replaced by the equivalent $T\neg$.

It is also important to notice that the blocking mechanism can affects the performance of a theorem prover in a very remarkable way. Even using the same set of rules, having a different blocking rule lead to very different performance that are often against the expectations.

As expected the provers generated with MetTel2 are generally slower compared with existing theorem provers for the same logic fragment. This show that, apart from the tableau rules specification, there are other factors that influence the performance of a theorem prover. These factor can either be optimization at a logical level (such as Simplification) or at a code level. For example, FCUBE [FFF10] and PITP [AFM06] implements the same tableau rules (the only difference is the presence of the sign $F_C$ in PITP) and both of them implement Simplification but they perform very differently on the SYJ201 problems family. This is because FCUBE use different data structures that penalize the performance of the prover. This shows that the performance of a theorem
prover depends on both the logical aspect and the implementation techniques.

The comparison between different theorem provers generated with MetTel2 shows that there is no one single combination of tableau rules and blocking mechanism that perform better on all kind of formulas. In order to obtain the best results an exhaustive series of experiments have to be done, since a theoretical analysis is not enough and sometimes the practical behaviour of a theorem prover is different from what is expected.

MetTel2 is not the only tool that allows the automatic generation of theorem provers. Lotrec ([FdCFG+01]) and TWB ([AG09]) are two tools that aim to automate the process of generating tableau based theorem provers.

The main difference between these system and MetTel2 is that they do not produce any code for the theorem prover but they act as a virtual machine that performs derivations.

Due to time constraint and some technical difficulties it has not been possible to compare the provers generated using MetTel2 with the provers generated with Lotrec and TWB but it will be of interest in the future to compare them, in order to see how the provers generated with MetTel2 perform against prover generated by the same kind of existing tool. Since both Lotrec and TWB are not optimized for intuitionistic propositional logic, the comparison between these systems can give a better idea on the performance of MetTel2.
Bibliography


86

[Smu68] Raymond M. Smullyan. First-Order Logic. Ergebnisse der Mathematik

[SRM73] Laurent Siklóssy, A. Rich, and Vesko Marinov. Breadth-first search:

[ST07] R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive
description logics with role negation. In K. Aberer, K.-S. Choi, N. Frid-
man Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika,
D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, edi-
tors, The Semantic Web, 6th International Semantic Web Conference,
2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Bus-
san, Korea, November 11–15, 2007, volume 4825 of Lecture Notes in

[ST09] Renate A. Schmidt and Dmitry Tishkovsky. Automated synthesis of
tableau calculi. In Martin Giese and Arild Waaler, editors, Automated
Reasoning with Analytic Tableaux and Related Methods, volume 5607
of Lecture Notes in Computer Science, pages 310–324. Springer Berlin

[ST12] Renate A. Schmidt and Dmitry Tishkovsky. Using tableau to decide de-
scription logics with full role negation and identity. Computer Research

[Sut09] G. Sutcliffe. The TPTP Problem Library and Associated Infrastruc-
ture: The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning,

Archive/TPTP-v2.7.0/TR-v2.7.0/Utilities.shtml, May 2013.

Mettel: A tableau prover with logic-independent inference engine. In K. Brünnler and G. Metcalfe, editors, Automated Reasoning with Ana-
lytic Tableaux and Related Methods (TABLEAUX 2011), volume 6793 of