2014-10-09

The Efficiency of Automated Theorem Proving by Translation to Less Expressive Logics

Negin Arhami
University of Miami, neginarhami@yahoo.com

UNIVERSITY OF MIAMI

THE EFFICIENCY OF AUTOMATED THEOREM PROVING BY TRANSLATION TO LESS EXPRESSIVE LOGICS

By

Negin Arhami

A THESIS

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Master of Science

Coral Gables, Florida

May 2014

UNIVERSITY OF MIAMI

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science

THE EFFICIENCY OF AUTOMATED THEOREM PROVING BY TRANSLATION TO LESS EXPRESSIVE LOGICS

Negin Arhami

Approved:

_____	_____
Geoff Sutcliffe, Ph.D. Associate Professor of Computer Science	Ubbo Visser, Ph.D. Associate Professor of Computer Science
_____	_____
Peter Lewis, Ph.D. Associate Professor of Philosophy	M. Brian Blake, Ph.D. Dean of the Graduate School

ARHAMI, NEGIN (M.S., Computer Science)
The Efficiency of Automated Theorem Proving by Translation to Less Expressive Logics (May 2014)

Abstract of a thesis at the University of Miami.

Thesis supervised by Professor Geoff Suttcliffe.
No. of pages in text. (72)

Many Automated Theorem Prover (ATP) systems for different logical forms, and translators for translating different logical forms from one to another, have been developed and are now available. Some logical forms are more expressive than others, and it is easier to express problems in those forms. On the other hand, the ATP systems for less expressive forms have been tested for many years, and are more powerful and reliable. There is a trade-off between expressivity of a logical form, and power and reliability of the available ATP systems. Different ATP systems and translators can be combined to solve a problem expressed in a logic. In this research, an experiment has been designed and carried out to compare all different possible ways of trying to solve a problem. The possibilities are either to use an ATP system for the original form, or translate the problem to a less expressive form. If the problem is translated to a less expressive form, then again the same two possibilities are available, until no further translation is possible. No translator was available to translate from Conjunctive Normal Form to Description Logic, which sits between Effectively Propositional Logic and Propositional Logic in terms of expressivity. A translation procedure for translating Conjunctive Normal Form to Description Logic, and its implementation as `Saffron` are presented. Additionally, this research includes

a survey of different logical forms. Propositional Logic, Description Logic, First Order Form, Conjunctive Normal Form, Effectively Propositional, Typed First order Form - monomorphic, Typed First order Form - polymorphic, Typed Higher order Form - monomorphic. The properties, syntax, and semantics of each logical form are briefly described. For each form, the most popular ATP systems and translators for translating to a less expressive forms are introduced.

Dedicated To

My Parents, My Sister and My Brothers

Acknowledgements

Above all, I would like to express my greatest gratitude to my adviser who prepared and organized a great research opportunity for me. Dr. Geoff Sutcliffe, I would never forget your appreciative passion, effort and support.

For their valuable assistance and guidance in this research, I am most grateful to Mr. Saminda Abeyruwan, Dr. Christoph Benzmüller, Dr. Jasmin Blanchette, Dr. Andrei Paskevich, Dr. Stephan Schulz, and Dr. Ubbo Visser.

I would also like to thank the other member of the guidance committee, Dr. Peter Lewis, for reviewing my research work and thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sound (valid) reasoning deals with the process of determining whether or not a conclusion follows inevitably from some accepted facts. Example 1 presents a set of fact statements A, and a conclusion statement C. Testing whether the set of facts A leads to the conclusion C can be done by considering one or both of two perspectives. One perspective is the *semantic* perspective, in which the meaning of the sentences is considered. From basic mathematics, it is known that four is divisible by two because four is an even number, and all even numbers are divisible by two. The other perspective is the *syntactic* perspective. In this perspective, without considering the mathematical rules for numbers, only by considering the structure of the statements and applying logical rules, the statement `four is divisible by two` can be concluded from the statements `All even numbers are divisible by two` and `four is an even number`.

Example 1

Sound reasoning deals with the process of determining whether or not a conclusion follows inevitably from some accepted facts.

A = { All even numbers are divisible by two,

Four is an even number}

C = Four is divisible by two

Logic is used to formalize these notions. A logic consists of a syntax and a semantics. The syntax is used to write statements about a domain as a set of formalized statements, *formulae*. A *logic problem* consists of some *axiom* formulae (the fact statement) and a *conjecture* formula (the conclusion).

The semantics of a logic is expressed in terms of *interpretations*. An interpretation of a problem relates the symbols of the problem to entities of the real world, and assigns `TRUE` or `FALSE` to each formula of the problem by applying interpretation rules. An interpretation that evaluates a formula to `TRUE` is a *model* of the formula. An interpretation that evaluates all formulae in a set of formulae to `TRUE` is a model of the set of formulae. A set of formulae is *satisfiable* if and only if it has at least one model. A set of formulae is *unsatisfiable* if and only if it has no models. The conjecture of a logic problem is a *logical consequence* of its axioms if every model of the axioms is a model of the conjecture. The conjecture of a problem is a *theorem* if it is a logical consequence of the axioms of the problem. On the contrary, the conjecture of a problem is a *nontheorem* if it is not a logical consequence of the axioms of the

problem. Sound reasoning processes typically use a syntactic perspective to establish that a conjecture is a theorem.

An alternative way to check if the conjecture of a problem is a theorem is to check if the union of the axioms and the negated conjecture has no model, i.e. is unsatisfiable. A common approach is to apply satisfiability preserving transformations from the set consisting of the axioms and the negated conjecture. If these lead to an obviously unsatisfiable set, e.g. a set containing an empty formula, then it is known that the original set is unsatisfiable, and hence, the conjecture is a theorem of the axioms. From now on in this thesis, logic problems are expressed in this form, i.e. a set consisting of the axioms and the negated conjecture.

A logic is *decidable* if for every problem expressed in that logic it can be determined if it is satisfiable or unsatisfiable in a finite time. A logic is *undecidable* if it is not decidable, i.e., there is at least one problem in that logic for which the satisfiability cannot be determined in a finite time. A logic is *semi-decidable* if for every problem expressed in that logic, it can be determined whether it is unsatisfiable (in a finite time), but the satisfiability of some problems in that logic cannot be determined. Some logics are decidable, but many are semi-decidable or undecidable.

To express a problem in a logical form, different logics with different levels of expressivity are available. Expressing a problem in a more expressive logic is easier because it is more natural and compact, but it is more difficult to reason about a problem in a more expressive form. On the other hand, expressing a problem in a less expressive logic is more difficult, but it is easier to reason about the problem. It is possible to translate a problem in one logic to another logic. A translation from one

logic to another is a *satisfiability preserving translation* if every satisfiable problem in the source logic remains satisfiable in the destination logic after the translation.

1.1 Automated Theorem Proving (ATP) and Thousands of Problems for Theorem Provers (TPTP)

Automated Theorem Proving (ATP) [1] is concerned with developing automatic techniques and computer programs for checking whether the conjecture of a logic problem is a theorem. An *ATP system* is a program that automatically checks whether the conjecture a problem is a theorem. Many logics have ATP systems available. The example of available ATP systems are `iProver`, `Vampire` [2], and `Princess` [3].

Many satisfiability preserving translators for translating a problem in one logic to another one have been implemented [4]. Examples of available translators are `Monotonox` [5] and `ECNF` [6]. For the goals of this research, only translations from more expressive logic to less expressive ones are interesting.

The *Thousands of Problems for Theorem Provers* (TPTP) [7] is a comprehensive library of test problems for ATP systems and translators. It also includes software tools that facilitate using the ATP systems and translators. The TPTP was developed to make evaluating and comparing ATP systems and translators possible by providing general guidelines, outlining the requirements for ATP system evaluation, and standards for input and output for ATP systems. The TPTP provides syntaxes for logics that can be input on standard keyboards without use of special mathematical characters.

The TPTP library of sample problems, the ATP systems available in the `SystemOnTPTP` [8], and the translators in the `SystemB4TPTP` [8], have been the source of test problems, ATP systems, and translators in this research. Also, software tools of the TPTP have been used to execute ATP systems and translators over the sample problems.

1.2 Related Conferences and Competitions

The two major forums for presentation of new research about different aspects of ATP and the development of new ATP systems and translators are the *Conference on Automated Deduction* (CADE) and *International Joint Conference on Automated Reasoning* (IJCAR) [9]. These conferences were started in 1974, and were held every two years in the beginning, and every year since 1996. At each CADE and IJCAR conference, the *CADE ATP System Competition* (CASC) [10] is held to introduce and evaluate contestant ATP systems. CASC evaluates the performance of sound and fully automatic ATP systems. The source of test problems for the competition is the TPTP library. To evaluate an ATP system, the number of problems solved within a time limit, the number of problems solved with a solution output within a time limit, and the average runtime for problems solved are considered.

CASC is the main basis for selecting the ATP systems used in the experiments in this research. Most of the ATP systems are the winners of the latest competitions.

1.3 Research Goals

Propositional Logic, Description Logic (DL), First Order Form (FOF), Conjunctive Normal Form (CNF), Effectively Propositional Form (EPR), Typed First order Form-monomorphic (TFF0), Typed First order Form-polymorphic (TFF1), and Typed Higher order Form-monomorphic (THF0) are some of the logics supported by the TPTP. These logics and their properties are discussed in Chapter 2.

As illustrated in the Figure 1.1, a real-world problem is first expressed in a logic as a logic problem. The logic problem can be either solved using the ATP system of that logic or translated to a less expressive logic. If it is translated to less expressive logic, again the same two options of translating down (if possible), and solving an ATP system are available. This continues until no further translation is possible. In Figure 1.1, the double line arrows represent the encoding of a domain problem in the original logic in which the problem is expressed, the plain arrows are available translations, and the dashed arrows are reasoning.

Going down from the more expressive logics to less expressive logics, the expressivity of the logic decreases. On the other hand, the ATP systems for less expressive logics are more powerful and more trustworthy than the more expressive ones because they have been tested for many years, compared to more expressive logic ATP systems. There are translators that can be used to translate a problem in a more expressive logic down to less expressive ones. A problem can be expressed in more expressive logic to benefit from the expressivity of the logic, then translated down to a less expressive logic to benefit from the power of the ATP systems. In the process

of translating between two logics extra complexity might be added to the problem, which affects the performance of the ATP system. There is is a trade-off between power of the ATP system for a logic and the effect of translating to the destination logic. The choice of reasoning using an ATP system of the original logic, or translating to a less expressive logic, is the major concern of this research.

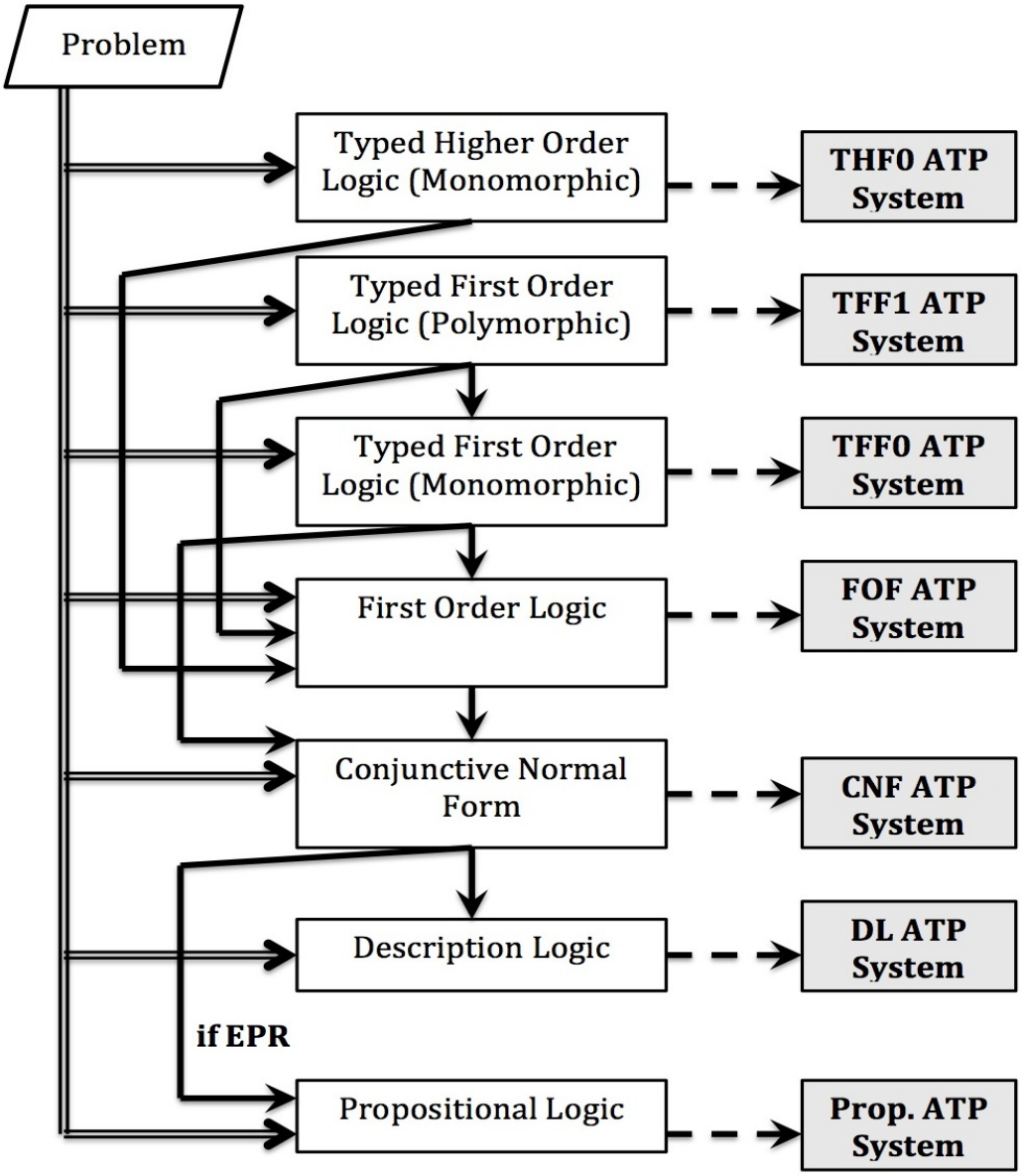

Figure 1.1: Different Combinations of Translators and ATP Systems to Solve a Problem

1.4 Road Map

This thesis is organized as follows. Chapter 1 contains an introduction to the research and related issues, to build up a basis for understanding the thesis. In Chapter 2 a survey of the logics related to the research is presented. Chapter 3 describes a translation procedure from CNF to DL, and its implementation as `Saffron`. Additionally, it includes a discussion of the results of testing `Saffron` on sample problems. In Chapter 4 the experiment to evaluate different ways of reasoning about logic problems in different logical forms is described, and its results are analyzed and discussed. Finally, Chapter 5 contains the summary of the whole thesis and conclusions.

Chapter 2

Survey of Logic

In this chapter the logical forms Propositional logic, Description Logic (DL) First Order Form (FOF), Conjunctive Normal Form (CNF), Typed First order Form-monomorphic (TFF0), Typed First order Form-polymorphic (TFF1), and Typed Higher order Form-monomorphic (THF0) are described.

2.1 Propositional Logic

Propositional logic is a decidable logic. Examples of applications of propositional logic include designing digital circuits [11], encoding constraint satisfaction problems [12], and reasoning about specifications [13].

2.1.1 Syntax

The syntax of propositional logic consists of propositions and logical operators. Some examples of propositions are presented in the Example 2.

Example 2

`coffee_helps_me_stay_awake`

`coffee_is_my_favorite_drink`

`p`

`q`

By combining different propositions using logical operators, propositional formulae can be generated.

Operators in propositional logic.

Negation. ¬ is a unary prefix operator, and negates the proposition.

Example 3

¬`coffee_is_my_favorite_drink`

¬`p`

And. ∧ is a binary infix operator. It is the conjunctive operator, and can be interpreted as English "and".

Example 4

`coffee_helps_me_stay_awake` ∧ `coffee_is_my_favorite_drink`

Or. ∨ is a binary infix operator. It is the disjunctive operator, and can be interpreted as English "or".

Example 5

`coffee_helps_me_stay_awake` $\vee$ `coffee_is_my_favorite_drink`

Implication. $\Rightarrow$ is a binary infix operation, and can be interpreted as English "if-then".

Example 6

`coffee_helps_me_stay_awake` $\Rightarrow$ `coffee_is_my_favorite_drink`

Equivalence . $\Leftrightarrow$ is a binary infix operation, and can be interpreted as English "if-and-only-if".

Example 7

`coffee_helps_me_stay_awake` $\Leftrightarrow$ `coffee_is_my_favorite_drink`

In TPTP, different symbols are used for some logical operators. The symbols `&`, `|`, `=>`, and `<=>` in TPTP are respectively equivalent to the $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ mathematical symbols.

2.1.2 Semantics

A propositional logic interpretation assigns either `TRUE` or `FALSE` to each proposition. The truth value of a formula is determined by a considering the values of its propositions. The truth table of each operator holds the resulting value of any possible combination of the values of the propositions. Table 2.1 shows the truth tables of all the binary operators.

Table 2.1: Example of a Truth Table

p	q	p∧q	p∨q	p⇒q	p⇔q
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

2.1.3 ATP Systems

Two ATP systems for propositional logic are `MiniSat` [14] and `ZChaff` [15]. In this research, `MiniSat` is used for reasoning about problems expressed in propositional logic. Different versions of `MiniSat` have been implemented for different applications, and they are implemented in different languages, such as C# and C++. Version 2.2.0, used in this research, is in C++. Examples of the reasoning techniques applied in `MiniSat` are conflict-clause recording and conflict-driven back jumping. `MiniSat` is a powerful solver, and was the winner of SAT-Race in 2008 [16].

2.2 Description Logic

Description Logic (DL) is a logical form for knowledge representation [17]. It is a decidable logical form, and is more expressive than propositional logic. DL is applied mostly in semantic web technologies [17]. A major application of DL is expressing ontologies. An ontology includes a categorization of elements of a domain of interest, definitions of the relationships between these categories, and knowledge about the domain.

2.2.1 Syntax

Manchester OWL Syntax [18] and exchange syntaxes like XML or RDF/XML are popular syntaxes for Description Logic. In this research, RDF syntax is used. Example 8 can be expressed in DL.

Example 8

8.a. Coffee is a drink.

8.b. Negin is a person.

8.c. Coffee is Negin's favorite drink.

The building blocks of DL are *individuals*, *classes*, and *roles*.

Individuals are equivalent to constants in First Order Form (FOF) (see Section 2.3) or Conjunctive Normal Form (CNF) (see Section 2.3.2). For example, in the sentences "Coffee is a drink", and "Negin is a person", `coffee` and `Negin` are individuals. In DL, the two individuals would be expressed as Example 9.

Example 9

```
<owl:NamedIndividual rdf:about="coffee"/>

<owl:NamedIndividual rdf:about="negin"/>
```

Classes are sets of individuals. Every individual belongs to one or more classes. A class can be empty, finite, or infinite. Classes are same as unary predicates in FOF or CNF. The default class is `Thing`. If an individual does not belong to a class, then it belongs to only the default class `Thing`. A class can be a subclass of, equivalent

to, disjoint to, or the complement of another class. It can also be a conjunction or disjunction of other classes. All these relationships between classes are the same as the corresponding relationships between two sets. By default, every class is a subclass of `Thing` so that every member of each class is also a member of the default class `Thing`. The empty class in DL is class `Nothing`, which has the properties of the empty set. In Example 8 , `Coffee` belongs to the class `Drink`, and `Negin` belongs to the class `Person`. Examples 10 and 11 are the DL syntax for Examples 8.a and 8.b.

Example 10

```
<owl:Class rdf:about="Drink"/>

<owl:NamedIndividual rdf:about ="coffee">
    <rdf:type rdf:resource="#Drink"/>
</owl:NamedIndividual>
```

Example 11

```
<owl:Class rdf:about="Person"/>

<owl:NamedIndividual rdf:about="negin">
    <rdf:type rdf:resource="#Person"/>
</owl:NamedIndividual>
```

Individuals can be related one to another by *roles*. In RDF syntax, roles are declared using object property tags. In Example 8.c, the role of `coffee` is `Negins favorite drink`. Example 12 is the DL syntax for Example 8.c.

Example 12

```
<owl:ObjectProperty rdf:about="favoriteDrink">
</owl:ObjectProperty>

<owl:NamedIndividual rdf:about="coffee">
     <favoriteDrink rdf:resource="#negin"/>
</owl:NamedIndividual>
```

There are different variations of DL. *ALC* is the basic DL, which includes the language constructs, such as classes, roles, individuals, class membership and role instances, conjunction, disjunction and negation of classes. One of the more expressive variations of DL is *SROIQ*. Each letter of the word *SROIQ* represents a feature in this variation of DL.

S All the *ALC* features.

R Limited complex role inclusion axioms such as reflexivity, irreflexivity, and role disjointness.

O Nominals (closed classes with a finite number of elements).

I Inverse properties.

Q Qualified cardinality restrictions.

SROIQ is the target description logic in this research. Most of its features are covered by `Saffron`, the CNF to DL translator presented in the Chapter 4.

2.2.2 Semantics

An interpretation of a DL problem consists of a nonempty set, a domain, and an interpretation function. The interpretation function maps every individual in the problem to an element in the domain. Additionally, the interpretation function maps

every class in the problem to a subset of the domain, maps the default class `Thing` to the domain, and the empty class `Nothing` to the empty set. The interpretation function also maps every role to a subset of the set of all pairs of elements of the domain.

2.2.3 ATP Systems

`FaCT++` [19] and `HermiT` [20] are examples of ATP systems for DL. `FaCT++` is an open-source ATP system for SROIQ, implemented in C++. The `FaCT++` decision procedure is tableau-based [19]. `HermiT` is also an open-source ATP system for SROIQ, implemented in Java. The `HermiT` decision procedure is hypertableau-based [20].

2.3 First Order Form, Conjunctive Normal Form and Effectively Propositional Form

2.3.1 First Order Form (FOF)

To express some statements in a logical form, a more expressive form than propositional logic or description logic is required. Examples 13.a and 13.b cannot be expressed in propositional logic or description logic, but they can be expressed in FOF.

Example 13

13.a. Coffee is some people's favorite drink.

13.b. Coffee helps everybody stay awake.

FOF Syntax

Examples 14.a and 14.b are the FOF syntaxes for sentences 13.a and 13.b in Example 13.

Example 14

14.a. ∃ `X is(coffee,favorite_drink_of(X))`

14.b. ∀ `Y helps_stay_awake(coffee,Y)`

There are three sets of symbols in FOF, variables, functions, and predicates. In Example 14.a, `X` and `Y` are variables. The symbols `coffee` and `favorite_drink_of` are functions with zero and one argument respectively. The number of arguments of a function is called the arity. Functions can have any arity greater than or equal to zero. Functions with zero arguments are also called constants. The symbols `is` and `helps_stay_awake` are predicates, both with two arguments. Predicates can have any arity greater than or equal to zero. Predicates with zero arity are the same as propositions in propositional logic.

Terms are built of functions and variables, and atoms are built of predicates applied to terms. *Ground terms* and *ground atoms* are terms and atoms without any variables. By combining atoms using logical operators, more complex formulae can

be constructed. The logical operators of FOF are the same as the logical operators of propositional logic, with two extra quantifiers, the Existential Quantifier ∃ and the Universal Quantifier ∀. For example, ∃ is the existential quantifier in Example 14.a, which quantifies over the variable `X`, and ∀ is the universal quantifier in Example 14.b, which quantifies over the variable `Y`. The TPTP symbols for ∃ and ∀ are `?` and `!` respectively.

FOF Semantics

An interpretation of a FOF language includes a *domain*, a *function map*, and a *predicate map*. The domain of an interpretation is a nonempty set. The function map defines a function for each function symbol, from n-tuple elements of the domain to an element of the domain, where n is the arity of the function symbol. (Thus, every constant of the problem is mapped to an element of the domain.) The predicate map defines a function for each predicate symbol from m-tuple elements of the domain to {`TRUE, FALSE`}, where m is the arity of the predicate. Atoms can only get the values of `TRUE` and `FALSE`. An interpretation is used to map ground terms to domain elements and ground atoms to {`TRUE, FALSE`}. The interpretation of logical operators of FOF, other than quantifiers, are the same as for propositional logic. The interpretations of the existential quantifier and the universal quantifier are as follows.

- *Existential Quantifier* ∃. If an existential quantifier quantifies over a variable in a formula, then there should be at least one element in the domain of the interpretation that makes the formula `TRUE`.

- *Universal Quantifier* $\forall$. If an existential quantifier quantifies over a variable in a formula, then all the elements in the domain the of interpretation should make the formula `TRUE`.

Herbrand Interpretation. The *Herbrand Universe* of a FOF problem is the set of *ground terms*. The *Herbrand Base* is the set of all ground atoms. A *Herbrand interpretation* is an interpretation of a FOF problem if the domain of the interpretation is the Herbrand Universe. The function map of the Herbrand interpretation is the identity function. The predicate map of the Herbrand interpretation is a subset of the Herbrand Base that maps to `TRUE`.

FOF ATP Systems

Many FOF ATP systems are available. Examples of FOF ATP systems are `Vampire` [2], `SPASS` [21], `Z3` [22], `E`—`Darwin`, and `Darwin` [23].

In this research, `Vampire` is used. The current version of `Vampire` was implemented at the University of Manchester, England. `Vampire` was the winner of CASC-J6 FOF and LTB divisions in 2013. This ATP system is written in C++ and consists of two layers, the shell and the kernel. The shell accepts problems in FOF or CNF. It then transforms the input to CNF if it is not already CNF. The CNF is passed to the kernel for reasoning. To increase efficiency, the search space is pruned by omitting redundancies, applying operations such as subsumption and removing tautologies. The kernel of `Vampire` supports equality, and handles equality by implementing the calculi of ordered binary resolution and superposition.

FOF Translators

FOF problems can be translated to less expressive logical forms. There is a satisfiability preserving procedure to translate FOF problems to CNF. There are some translators available, such as `ECNF` [6] and `VCNF` [2]. In this research, `ECNF` is used. `ECNF` is a subsystem of the `E` [24] ATP system, which was first started in the Technische Universitat Munchen, Germany. ECNF is used to clausify FOF without using advanced Skolemization techniques. This technique is explained in [25].

Some FOF problems can be translated to a DL, but there is no translator available for this purpose yet.

2.3.2 Conjunctive Normal Form

CNF Syntax

Each formula in FOF can be translated to an equisatisfiable CNF formula. Each problem in CNF is a conjunction of *clauses*, and each clause is a disjunction of some *literals*. A literal is an atom either with or without the negation sign. An empty clause has no literals. In a CNF problem, all variables are universally quantified. Example 15 is an example of a CNF formula with three clauses.

Example 15

`(a(X,Y)∨b∨c(T,S))∧(¬a(X,V)∨b)∧(d(R)∨¬b∨c(V,S)∨e(X,V,T))`

CNF Semantics

CNF semantics are the same as FOF semantics. Every set of clauses has a model if and only if it has a Herbrand model. This is the basis for building sound inference rules for a set of clauses, e.g., resolution, as used in `Vampire`.

CNF ATP Systems

Most FOF ATP systems can also be used for CNF. All the FOF ATP systems, mentioned in Subsetion 2.3.1, except for `Z3` are also CNF ATP systems.

CNF Translators

There are a few translation tools to translate CNF problems to less expressive logics. Some CNF problems can be translated to DL. In this research a translator for this purpose is developed, and is described in Chapter 4.

2.3.3 Effectively Propositional Form (EPR)

CNF problems with a finite Herbrand Universe are called Effectively Propositional (EPR) problems. EPR is a decidable fragment of CNF. It is also decidable to check if a CNF problem is EPR or not. A problem with no functions of arity more than zero has a finite Herbrand Universe and is EPR. In addition, if a problem has a function with arity more than zero, but does not have any variables or equality, then it is an EPR problem. It can be checked if in a problem, either there is no function, or all the functions are constants. Additionally, in a problem with functions with arity more than or equal to one, it can be checked if there are any variables or equality.

EPR ATP Systems

All CNF ATP systems can also be used for EPR problems. In this research `iProver` [26] is used for reasoning about EPR problems. `iProver` is the winner of the EPR division at all CASC from 2008 to 2012.

EPR Translators

All EPR problems can be translated to propositional logic. In this research, `EGround` [27] is used to translate EPR problems to propositional logic. EPR problems and the set of all of their ground instances are equisatisfiable. To translate an EPR problem, `EGround` generates the ground instances of the problem. Since the number of the ground instances is exponential to the number of variables in a clause, i.e. with n constant symbols a clause with m variables has n^m ground instances, `EGround` applies clause splitting, structural constraints, and propositional simplification techniques to reduce the number of ground instances.

2.4 Typed First-order Form - monomorphic (TFF0)

TTF0 [24] is the monomorphic typed first-order logic with predefined and user-defined types. It supports ad-hoc polymorphism for only equality and some defined symbols.

2.4.1 Syntax

The syntax described here is the syntax used in the TPTP. The defined types are `$i` for individuals and `$o` for booleans. Any user-defined type is of type `$tType`. Example 16 can be expressed in TFF0.

Example 16

16.a. Negin is a person.

16.b. Coffee is a beverage.

16.c. Vanilla syrup is a syrup.

16.d. The mixture of a syrup and a beverage is a beverage.

16.e. The mixture of coffee and any syrup helps some people stay awake.

In Example 16, `person`, `beverage`, and `syrup` are types. Statements 16.a, 16.b, and 16.c are expressed in TFF0 as Example 17.

Example 17

```
tff(person_type, type, person:  $tType ).

tff(beverage_type, type, beverage:  $tType ).

tff(syrup_type, type, syrup:  $tType ).
```

The types of a function's arguments and its return type are specified at the time of declaration. In Example 16, `Negin`, `coffee`, `vanilla syrup` and `mixture` are functions, and are expressed in TFF0 as Example 18.

Example 18

```
tff(negin_type, type, negin:  person).

tff(coffee_type, type, coffee:  beverage).

tff(vanilla_syrup_type, type, vanilla_syrup:  syrup).

tff(mixture_type, type, mixture:  beverage * syrup > beverage).
```

If a function is not declared, the default type of its arguments and return type is `$i`.

A predicate is a mapping from the types of its arguments to type `$o`. The types of a predicate's arguments are specified at the time of declaration. In Example 16, "helps stay awak" is a predicate, and is expressed in TFF0 as Example 19.

Example 19

```
tff(help_stay_awake_type, type,
    help_stay_awake:  person * beverage > $o).
```

If a predicate is not declared, the default types of all of it's arguments are `$i`.

Statements 16.d and 16.e are expressed in TFF0 as Example 20. `S` and `P` are variables of type `syrup` and `person`. The types of variables are specified at the time of quantification. If the type of a variable is not specified, its default type is `$i`.

Example 20

```
tff(mixture_of_coffee_helps_some_one_stay_awake, axiom,
    !  [S: syrup]:  ?  [P: person] :
     help_stay_awake(P,mixture(coffee,S)) ).
```

2.4.2 Semantics

All rules for logical operators, and quantifiers in FOF are also applied in TFF0. Types are interpreted by non-empty, pairwise disjoint domains. All uninterpreted functions and predicates are monomorphic.

2.4.3 ATP Systems

Available ATP systems for TFF0 include `Princess` [3] and `SNARK` [28]. `Princess` is the ATP system used in this research. `Princess` was developed at Chalmers University, Sweden. `Princess` was the winner of the CASC-J6 TFA division in 2013. This ATP is written in Scala, and employs Scala standard libraries, Java standard libraries, and Cup parser libraries. It can be executed on any recent Java virtual machine.

2.4.4 Translators

TFF0 problems can be translated to FOF and CNF. There is a satisfiability preserving procedure to translate a TFF0 problems to FOF, and similarly to CNF. `Monotonox2FOF` and `Monotonox2CNF` are variations of `Monotonox` [5], which is a translator of many typed first order logic to first order logic. `Monotonox` is based on three basic approaches; generating type predicates, generating type functions, and type erasure. Generating type predicates and type functions preserves the satisfiability of a problem, but results in very complicated formulae that affect most ATP systems negatively. Type erasure reduces the complexity, but it does not always preserve

satisfiability. Only certain types can be erased without changing the satisfiability of the original problem. Therefore, `Monotonox` applies the combination of these three approaches to reduce the complexity of the generated formulae, while preserving the satisfiablity.

2.5 Typed First-order Form - polymorphic (TFF1)

Typed First-order Form - polymorphic (TFF1) is distinguished from TFF0 by allowing polymorphism over the types of the arguments and return value of functions and predicates.

2.5.1 Syntax

The syntax described here is the syntax used in the TPTP. Similar to TFF0, the defined types are `$i` for individuals and `$o` for booleans. Any user-defined type is of type `$tType`. Monomorphic function types, and predicates are expressed as in TFF0, applications of monomorphic functions and predicates are the same as TFF0. Polymorphic function types are expressed using *type signatures*. In Example 21, the type definition `a_polymorphic_function_type` defines the polymorphic function `polymorphic_function` with $M + N$ arguments, where `t01`, `t02`, ..., and `tN` are user defined types. The `!>[AT1:$tType, AT2:$tType, ..., ATM:$tType, RT:$tType]` is the type signature and expresses that the first M arguments and the return value of this function are polymorphic. In the axiom `a_polymorphic_function_application`, `t11`, `t12`, ..., `t1M1`, `t21`, `t22`, ..., `t2M2`, and `rt` are user-defined types ($M1 + M2 = M$);

c1, c2,..., cM2 are constants of types t21, t22, ..., and t2M2; A11, A12, ..., A1M1 are variables; arg01, arg02,..., argN are of types t01, t02, ..., and tN; and retValue is a constant of type rt.

Example 21

```
tff(a_polymorphic_function_type,type,(
     polymorphic_function:
      !>[AT1:  $tType, AT2:  $tType, ..., ATM: $tType, RT: $tType] :
       ( ( AT1 * AT * ...  * ATM * t01 * t02 * ...  *tN ) > RT ) )).

tff(a_polymorphic_function_application,axiom,(
     ?[A1:  t11, A2:  t12, ..., A1M1:  t1M1] :
       retValue = polymorphic_function(t11,t12,...,t1M1,t21,t22,...,t2M2,rt,
        A11,A12,...,A1M1,
        c1,c2,...,cM2,
        arg01,arg02,...,argN) )).
```

Example 22 can be expressed in TFF1 as Example 23. In Example 23, beverage and syrup are user-defined types, and coffee, vanilla_syrup, caramel_syrup, and help_people_stay_awake are monomorphic function types. In this example mixture is a polymorphic function. The first argument type, and the return type are polymorphic, and can be either of the user defined types of beverage or syrup. In the type definition mixture_type, the !>[BeverageOrSyrup:$tType] is the type signature, and in the axiom mixture_of_coffee_help_people_stay_awake, the polymorphic function mixture is used. In mixture(beverage,coffee,S), the first argument, beverage, expresses the type of the polymorphic type of this function.

Example 22

Axioms = { Coffee is a beverage,

Vanilla syrup is a syrup,

Caramel syrup is a syrup,

The mixture of a syrup and a beverage is a beverage,

The mixture of a syrup and a syrup is a syrup,

The mixture of coffee and any syrup helps people stay awake }

Conjecture = Caramel vanilla coffee help people stay awake

Example 23

```
tff(beverage_type,type,(
    beverage:  $tType )).

tff(syrup_type,type,(
    syrup:  $tType )).

tff(coffee_type,type,(
    coffee:  beverage )).

tff(vanilla_syrup_type,type,(
    vanilla_syrup:  syrup )).

tff(caramel_syrup_type,type,(
    caramel_syrup:  syrup )).

tff(mixture_type,type,(
    mixture:
      !>[BeverageOrSyrup:  $tType] :
      ( ( BeverageOrSyrup * syrup ) > BeverageOrSyrup ) )).

tff(help_people_stay_awake_type,type,(
    help_people_stay_awake:  beverage > $o )).

tff(mixture_of_coffee_help_people_stay_awake,axiom,(
    !  [S: syrup] :
      help_people_stay_awake(mixture(beverage,coffee,S)) )).

tff(caramel_vanilla_coffee_help_people_stay_awake,conjecture,(
```

```
help_people_stay_awake(
 mixture(beverage,coffee,
  mixture(syrup,caramel_syrup,vanilla_syrup))) )).
```

2.5.2 Semantics

An interpretation of a TFF1 problem consists of a nonempty collection D of nonempty sets, the *domains*. The union of all domains is called the universe, U. Interpretation of the monomorphic terms and atoms are similar to TFF0 using each domain as a domain of a specific type. Only interpretation for a polymorphic functions and predicates needs to be added. Every possible domain of a polymorphic variables will be considered. For example, the function map of a polymorphic function $f(A_1, A_2, ..., A_n)$ with n arguments, such that m arguments have free types ($m < n$), and the return type is monomorphic, maps all possible n-tuples to an element of the domain that corresponds to the return type of the function. $n - m$ elements of the n-tuple, corresponding to the monomorphic arguments of the function, are selected from their corresponding domain, and m elements of the n-tuple, corresponding to polymorphic arguments of the function are selected from all possible domains.

2.5.3 ATP Systems

Available ATP systems for TFF1 include `Alt-Ergo` [29], which is the ATP system used in this research.

2.5.4 Translators

TFF1 problems can be translated to TFF0 and FOF. There is a satisfiability preserving procedure to translate a TFF1 problems to TFF0, and similarly to FOF. `Why3_TFF` and `Why3_FOF` are the only available translators for translating TFF1 to TFF0 and FOF, and are used in this research.

2.6 Typed Higher-order Form - monomorphic (THF0)

The Typed Higher-order Form - monomorphic (THF0) is distinguished from TFF0 by addition of a λ-calculus and quantification over functions.

2.6.1 Syntax

The syntax described here is the syntax used in the TPTP. In addition to the FOF TPTP symbols, THF0 includes types and two symbols `@` and `^` (which is the subsitution for mathematic λ). The `@` symbol is used for function application.

Types consists of defined, user-defined, and function types. The defined types are `$i` for individuals and `$o` for booleans. Any user-defined type is of type `$tType`. Function types are `a  ` or `b > c` where `a` is of type any of defined or user-defined types, and `b` and `c` are any of defined, user-defined or function types of the form `b > c`. Predicates in THF0 are functions that their return types are type `$o`. The types of variables are function types. The axioms and the conjecture in Example 24 can be

expressed in THF0 as in Example 25. The user-define types are `beverage`, and `syrup`, and the function types are `coffee`, `hot`, `heat`, `mix`, `hot_mixture`, and `cold_mixture`.

In Example 25 in the axiom `hot_mixture_definition`, `mix @ B @ S` states that the `mix` function is applied on the variable `B` (of type `beverage`) and `S` (of type `syrup`).

The λ-calculus is used for function definition (function abstraction is also a term used in many papers). In a function definition, the symbol `^` quantifies over local variable which are used in the body of the function definition. In Example 25, the axiom `cold_mixture_definition`, variables `B` and `S` of types `beverage` and `syrup` are local variables which are used in the definition of the function `cold_mixture`.

Example 24

Axioms = { Coffee is a beverage,

Heating a beverage makes the beverage hot,

The mixture of a beverage and a syrup is a beverage,

The hot mixture of a beverage and a syrup is a mixture of the beverage and the syrup, which is heated,

The cold mixture of a beverage and a syrup is a mixture of the beverage and the syrup}

Conjecture = There is some mixture of coffee and any syrup which is hot

Example 25

```
thf(syrup_type,type,(
    syrup:  $tType )).

thf(beverage_type,type,(
    beverage:  $tType )).

thf(coffee_type,type,(
    coffee:  beverage )).

thf(heat_type,type,(
    heat:  beverage > beverage )).

thf(hot_type,type,(
    hot:  beverage > $o )).

thf(mix_type,type,(
    mix:  beverage > syrup > beverage )).

thf(hot_mixture_type,type,(
    hot_mixture:  beverage > syrup > beverage )).

thf(cold_mixture_type,type,(
    cold_mixture:  beverage > syrup > beverage )).

thf(hot_mixture_definition,definition,
    ( hot_mixture
     = ( ^ [B: beverage,S: syrup] :
      ( heat @ ( mix @ B @ S ) ) ) )).

thf(cold_mixture_definition,definition,
    ( cold_mixture
     = ( ^ [B: beverage,S: syrup] :
      ( mix @ B @ S ) ) )).

thf(its_hot,axiom,(
    !  [B: beverage] :
     ( hot @ ( heat @ B ) ) )).

thf(hot_coffee,conjecture,(
    ?  [Mixture:  beverage > syrup > beverage] :
    !  [S: syrup] :
    ?  [B: beverage] :
```

```
( ( ( Mixture @ coffee @ S )
= B )
& ( hot @ B ) ) )).
```

2.6.2 Semantics

Interpretations for THF0 problems are similar to interpretations for TFF0 problems. The difference is that quantification over functions is allowed, so a variable of a function type with arity n corresponds to any n-tuple of a domain of an interpretation mapped to a domain of interpretation. There are two kinds of interpretations commonly employed for THF0. *Full semantics* requires that once the domain of discourse is satisfied, the variables of function types range over all possible elements of the correct type (all subsets of the domain, all functions from the domain to itself, etc.). Thus the specification of a full interpretation is the same as the specification of a TFF0 interpretation. *Henkin semantics*, which is essentially multi-sorted first-order semantics, are used in this research. Henkin semantics requires the interpretation to specify a separate domain for each type of higher-order variable to range over. Thus, an interpretation in Henkin semantics includes a domain D, a collection of subsets of D, a collection of functions from D to D, etc. [30]

2.6.3 ATP Systems

Available ATP systems for THF0 include `Isabelle-HOT` [31], `LEO` [32], and `Satallax` [33]. `Isabelle-HOT` is the ATP system used in this research.

2.6.4 Translators

THF0 problems can be translated to TFF0 and FOF. There is a satisfiability preserving procedure to translate a THF0 problems to TFF0, and similarly to FOF. `Isabelle-2TF0` and `Isabelle-2FOF` are the only available stand-alone translators for translating THF0 to TFF0 and FOF, and are used in this research. `Isabelle-2FOF` can also be used for translating TFF0 to FOF.

Chapter 3

`Saffron`, a Translator for CNF to DL

As explained in Chapter 1, the goal of this research is to investigate the possible ways of increasing the efficiency of solving problems using available ATPs and translators. A problem expressed in a logical form can be either solved using an ATP for that logical form, or if a translator is available, it can be translated to a less expressive logical form. If it is translated to a less expressive logical form, again the same two options are available. The experiment described in Chapter 4 was designed and carried out to compare different possible combinations of available ATPs and translators to solve a problem. No CNF to DL translator was available when this research was started. In this chapter, a CNF to DL translation procedure, as well as its implementation as `Saffron`, is described. `Saffron` is implemented in Prolog. The input problem is in CNF in the TPTP syntax, and the output of the translation is DL in RDF syntax.

3.1 Preparing a CNF Problem for Translation

Problems with only constants, and predicates with only one or two arguments, can be translated directly to DL. Constants, predicates with one argument, and predicates with two arguments, in CNF correspond to individuals, classes, and roles that are the building blocks of DL. Since these problems have no functions with arity greater than zero, they are EPR (their Herbrand Universe is finite). Predicates with arity greater than two, and functions with arity greater than zero cannot be translated directly to DL because there is no DL syntax for such predicates and functions. In this research, an effort was made to find a way to transform problems with predicates with arity greater than two or functions with arity greater than zero to problems with only predicates with arity one or two. However, these efforts failed for the following reasons. There is no known approach for replacing a predicate with arity greater than two with predicates with arity less than or equal to two. However, a function replacement approach for CNF problems is presented in [34]. This approach was considered for EPR problems with functions, but no variables or equality. In this approach a function with arity n ($n > 0$) is replaced with a predicate with arity $n + 1$. This approach was considered for replacing the functions with arity one with predicates with arity two. After replacing the functions with predicates, *axioms of totality* must be added to make the theorem proving techniques such as resolution complete. According to the axiom of totality for a function with arity n, the function should map every n-tuple of elements of the domain of an interpretation, to some element of the domain of the interpretation. Expressing the axiom of totality forces

the use of an existential quantifier, so the axioms of totality are first expressed in FOF, and then transformed to CNF. Transforming an existential quantifier to CNF introduces Skolem functions. Since the existential quantifier quantifies over the return value of a function with arity n, the arity of the introduced Skolem function is also n. As the goal of this process is to remove all the functions with arity greater than zero, this defeats the goal. However, if the domain of an interpretation has d elements the axioms of d-totality are sufficient. Example 26 shows the axioms of d-totality where `p` is a replacement for function `f` with arity one, and 1,2,...,d are the domain elements. Since the EPR problems are interesting for translation to DL, and the Herbrand Universe of an EPR problem is finite, the axioms of d-totality can be used.

Example 26

```
p(X,1)|p(X,2)|...|p(X,d)
```

In addition to axioms of totality, the axioms of *well-definedness* also must be added to complete the function replacement. According to the axiom of well-definedness of a function with arity n, the function maps every n-tuple of elements of the domain of an interpretation to at most one element of the domain of the domain of interpretation. Example 27 shows the axiom of well-definedness where `p_f` is a replacement for function `f` with arity one. Axioms of well-definedness are not translatable to DL because there is no DL syntax for them.

Example 27

```
∼p_f(X,Y1)|∼p_f(X,Y2)|X1=X2
```

In Example 28, function `f` in the axiom `predicate_applied_on_function` is replaced with the predicate `p_f` in the axiom `replacing_f_with_p`. The axiom `3_totality_of_f` is the 3-totality axiom for function `f`, where `e1`, `e2`, and `e3` are the elements of the Herbrand Universe. The axiom `well_definedness_of_f` is the well-definedness axiom for function `f`.

Example 28

```
cnf(predicate_applied_on_function,axiom,
    ( q(f(e1)) )).

cnf(replacing_f_with_p,axiom,
    (  p_f(X,Y) | q(Y) )).

cnf(3_totality_of_f,axiom,
    ( p_f(X,e1)
    | p_f(X,e2)
    | p_f(X,e3) )).

cnf(well_definedness_of_f,axiom,
    (   p_f(X,Y1)
    |   p_f(X,Y2)
    | Y1 = Y2 )).
```

3.2 Translation Procedure

As explained in Chapter 1, the main requirement of the translation from CNF to DL is to preserve the satisfiability of the CNF problems, and hopefully the unsatisfiability of them. Translation from CNF to DL is done by considering the intended semantics of each clause. For example, the CNF clause in Example 29 includes two unary predicates, `p` and `q`, that correspondingly introduce two DL classes `P` and `Q`. This clause expresses the fact that for every element `e` of an interpretation domain, if `p(e)`

then `q(e)`. The corresponding DL axiom of this clause defines a characteristic of the class `P`, which is that the class `P` is a subclass of the class `Q`.

Example 29

`~p(X)|q(X)`

Every DL axiom defines a characteristic of an individual, a class, or a role, or describes the default class `Thing`. For every constant appearing in a CNF problem, an individual with the same name is defined in DL. The characteristics of individuals that are covered in this translation are an individual belonging to a class, an individual not belonging to a class, an individual belonging to the union of classes and the complements of classes, and an individual being in a binary relationship with another individual. The equality or negative equality between two constants in a CNF clause has to be taken care of in a different way to other binary predicates applied to two constants. The corresponding DL axioms of equality between two individuals or inequality between two individuals include "same individual as" and "different individual as" notations. Table 3.1 illustrates CNF clauses and their corresponding DL characteristics of individuals.

For every unary predicate appearing in a CNF problem, a class with the same name and the capitalized first letter is defined in DL. The characteristics of classes that are covered in this translation are a class being a subclass of another class, and a class being equivalent to class `Thing`. Table 3.2 illustrates CNF clauses and their corresponding DL characteristics of classes. Subclass and equivalency characteristics are major characteristics of a DL class. The subclass characteristic implicitly covers the

Table 3.1: CNF Clauses and Equivalent Individual DL Characteristics

CNF clause	DL Equivalent
p(a)	Individual a belongs to the class P
∼p(a)	Individual a does not belongs to the class P
p1(a)|p2(a)|...|pN(a)| ∼q1(a)|∼q2(a)|...|∼qM(a) $N \geq 0$, $M \geq 0$, and $M + N \geq 2$	Individual a belongs to the union of classes P1, P2, ..., PN, and the complements of classes Q1, Q2, ..., QM
r(a,b)	Individual a is in the r relationship with the individual b
∼r(a,b)	Individual a is not in the r relationship with the individual b
a = b	a is a same individual as b
a != b	a is a different individual to b

equivalency characteristic because the equivalency between two classes is expressed by two CNF clauses, each of which expresses that each of the classes is a subclass of the other. A class being equivalent to the union of classes and the complement of classes can be expressed by expressing the subclass characteristic and some description of the default class Thing, which will be explained later in this section. For example, if class A is equivalent to the union of classes B1, B2, .., and BN, and the complements of C1, C2, .., and CM($N \geq 0$, $M \geq 0$, and $N + M \geq 2$). This can be expressed a the CNF clauses in Example 30. The clauses 30.b to 30.d are translated to DL axioms expressing the subclass characteristic of the classes B1, B2, .., and BN. The clauses 30.a and 30.e to 30.g are translated to DL axioms expressing a description of the class Thing, which will be explained later in this section. A class being equivalent to the intersection of classes and the complement of classes can be similarly expressed by expressing the subclass characteristic and some description of the default class Thing.

Example 30

30.a. ∼a(X)|b1(X)|b2(X)|...|bN(X)|∼c1(X)|∼c2(X)|...|∼cM(X)

30.b. ∼b1(X)|a(X)

30.c. ∼b2(X)|a(X)

...

30.d. ∼bN(X)|a(X)

30.e. c1(X)|a(X)

30.f. c2(X)|a(X)

...

30.g. cM(X)|a(X)

Table 3.2: CNF Clauses and Equivalent Class DL Characteristics

CNF clause	DL Equivalent
∼p(X)|q(X)	Class P is a subclass of class Q
p(X)	Class P is equivalent to class Thing

For every binary predicate appearing in a CNF problem, a role with the same name is defined in DL. The role characteristics that are covered in this translation are reflexivity, irreflexivity, symmetry, asymmetry, transitivity, and a role being the inverse of another role. Table 3.3 illustrates CNF clauses and their corresponding DL characteristics of roles.

Table 3.3: CNF Clauses and Equivalent Role DL Characteristics

CNF clause	DL Equivalent
r(X,X)	r is a reflexive relation
∼r(X,X)	r is an irreflexive relation
∼r(X,Y)|r(Y,X)	r is a symmetric relation
∼r(X,Y)|∼r(Y,X)	r is an asymmetric relation
∼r(X,Y)|∼r(Y,Z)|r(X,Z)	r is a transitive relation
∼r(X,Y)|s(Y,X)	r is the inverse of s

Some CNF clauses describe the default class `Thing`, so they are expressed in DL as a restriction on, or a description of the class `Thing`. Table 3.4 illustrates CNF clauses and their corresponding DL axioms that describe the class `Thing`. In Example 30, the clauses 30.a and 30.e to 30.g are translated to DL axioms using the rule represented in the second row of Table 3.4.

Table 3.4: CNF Clauses and Equivalent `Thing` Class Descriptions

CNF clause	DL Equivalent
`~p(X)`	Class `Thing` is equivalent to the complement of the class `P`
`p1(X)|p2(X)|...|pN(X)| ~q1(X)|~q2(X)|...|~qM(X)` ($N \geq 0$, $M \geq 0$, and $M+N \geq 2$)	All individuals belong to the union of classes `P1`, `P2`, ..., and `PN`, and the complements of classes `Q1`, `Q2`, ..., and `QM`
`r(X,a1)|r(X,a2)|...|r(X,aN)| ~r(X,b1)|~r(X,b2)|...| ~r(X,bM)` ($N \geq 0$, $M \geq 0$, and $M+N \geq 1$)	All individuals have `r` relationship with `a1`, `a2`, ..., or `aN`, or do not have `r` relationship with `b1`, `b2`, ..., or `bM`
`p1(X)|p2(X)|...|pI(X)| ~q1(X)|~q2(X)|...|~qJ(X)` `r(X,a1)|r(X,a2)|...|r(x,aK)| ~r(X,b1)|~r(X,b2)|...| ~r(x,bL)` ($I \geq 0$, $J \geq 0$, $K \geq 0$, $L \geq 0$, $I+J \geq 1$, and $K+L \geq 1$)	All individuals belong to the union of classes `P1`, `P2`, ..., and `PI`, and the complements of classes `Q1`, `Q2`, ..., and `QJ`. Also they have `r` relationship with `a1`, `a2`, ..., or `aK`, or they do not have `r` relationship with `b1`, `b2`, ..., or `bL`

In RDF syntax, each individual, each class, each role, and class `Thing` are expressed as an RDF tag, with their characteristics (if any exist) as sub-tags. When the clause-by-clause translation of the whole problem is completed, for each individual, each class, each role, and class `Thing`, all the characteristics, that were found during the translation are gathered and combined into in an RDF tag.

In a problem there might be a CNF clause that cannot be translated to a DL axiom, because either it has no equivalent DL axiom, or the translation rules do not support its translation. Thus a problem might be fully or partially translated with

these rules. However, the unsatisfiability of a partially translated problem might be preserved, because an unsatisfiable subset of the clauses has been translated. The experiment in Section 3.4 shows how successful this translator is in preserving the unsatisfiability of partially translated problems.

The CNF problem in Example 31 can be translated to DL and the result of the translation, without the header and footer of the `.owl` file, is in Eaxmple 32. The CNF problem is unsatisfiable because the negated conjecture `moein_is_sibling_of_negin` is in contradiction with two axioms `negin_is_sibling_of_moein` and `sibling_is_symmetric`. The negated conjecture `ahmad_is_dad_of_youngest_child` is also in contradiction with the axioms `negin_is_child_of_ahmad`, `negin_is_youngest_child`, and `dad_is_reverse_of_child`. It is hoped that the DL ontology resulted from the translation is inconsistent. Figure 3.1 is the result of `HermiTT` reasoning on the DL problem. This shows that the the DL ontology is inconsistent and the reasons for the inconsistency are explained.

Example 31

```
cnf(moein_is_a_person,axiom,
     ( person(moein) )).

cnf(negin_is_a_female,axiom,
     female(negin) )).

cnf(ahmad_is_a_person,axiom,
     person(ahmad) )).

cnf(ahmad_is_not_a_female,axiom,
       female(ahmad) )).

cnf(negin_is_sibling_of_moein,axiom,
     sibling(negin,moein) )).
```

```
cnf(ahmad_is_dad_of_moein,axiom,
      dad(ahmad,moein) )).

cnf(negin_is_child_of_ahmad,axiom,
      child(negin,ahmad) )).

cnf(negin_and_moein_are_different,axiom,
      negin != moein )).

cnf(negin_is_youngest_child,axiom,
      negin = youngest_child )).

cnf(sibling_is_symmetric,axiom,
        sibling(X,Y)
      | sibling(Y,X) )).

cnf(dad_is_reverse_of_child,axiom,
        dad(X,Y)
      | child(Y,X) )).

cnf(female_is_subclass_of_person,axiom,
        female(X)
      | person(X) )).

cnf(moein_is_sibling_of_negin,negated_conjecture,
        sibling(moein,negin) )).

cnf(ahmad_is_dad_of_youngest_child,negated_conjecture,
        dad(ahmad,youngest_child) )).
```

Example 32

```
<owl:NamedIndividual rdf:about="&family-ontology;ahmad">
    <family-ontology:dad rdf:resource="&family-ontology;moein"/>
    <rdf:type rdf:resource="&family-ontology;Person"/>
    <rdf:type>
       <owl:Class>
          <owl:complementOf rdf:resource="&family-ontology;Female"/>
       </owl:Class>
    </rdf:type>
</owl:NamedIndividual>

<owl:ObjectProperty rdf:about="&family-ontology;dad">
```

```
    <owl:inverseOf rdf:resource="&family-ontology;child"/>
</owl:ObjectProperty>
    <rdf:Description>
      <rdf:type rdf:resource="&owl;AllDifferent"/>
      <owl:distinctMembers rdf:parseType="Collection">
        <rdf:Description rdf:about="&family-ontology;negin"/>
        <rdf:Description rdf:about="&family-ontology;moein"/>
      </owl:distinctMembers>
    </rdf:Description>

<rdf:Description>
    <rdf:type rdf:resource="&owl;NegativePropertyAssertion"/>
    <owl:targetIndividual rdf:resource="&family-ontology;negin"/>
    <owl:assertionProperty rdf:resource="&family-ontology;sibling"/>
    <owl:sourceIndividual rdf:resource="&family-ontology;moein"/>
</rdf:Description>

<rdf:Description>
    <rdf:type rdf:resource="&owl;NegativePropertyAssertion"/>
    <owl:targetIndividual rdf:resource="&family-ontology;youngest_child"/>
    <owl:assertionProperty rdf:resource="&family-ontology;dad"/>
    <owl:sourceIndividual rdf:resource="&family-ontology;ahmad"/>
</rdf:Description>

<owl:Class rdf:about="&family-ontology;Female">
    <rdfs:subClassOf rdf:resource="&family-ontology;Person"/>
</owl:Class>

<owl:NamedIndividual rdf:about="&family-ontology;moein">
    <rdf:type rdf:resource="&family-ontology;Person"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&family-ontology;negin">
    <family-ontology:child rdf:resource="&family-ontology;ahmad"/>
    <family-ontology:sibling rdf:resource="&family-ontology;moein"/>
    <owl:sameAs rdf:resource="&family-ontology;youngest_child"/>
    <rdf:type rdf:resource="&family-ontology;Female"/>
</owl:NamedIndividual>

<owl:ObjectProperty rdf:about="&family-ontology;sibling">
    <rdf:type rdf:resource="&owl;SymmetricProperty"/>
</owl:ObjectProperty>
```

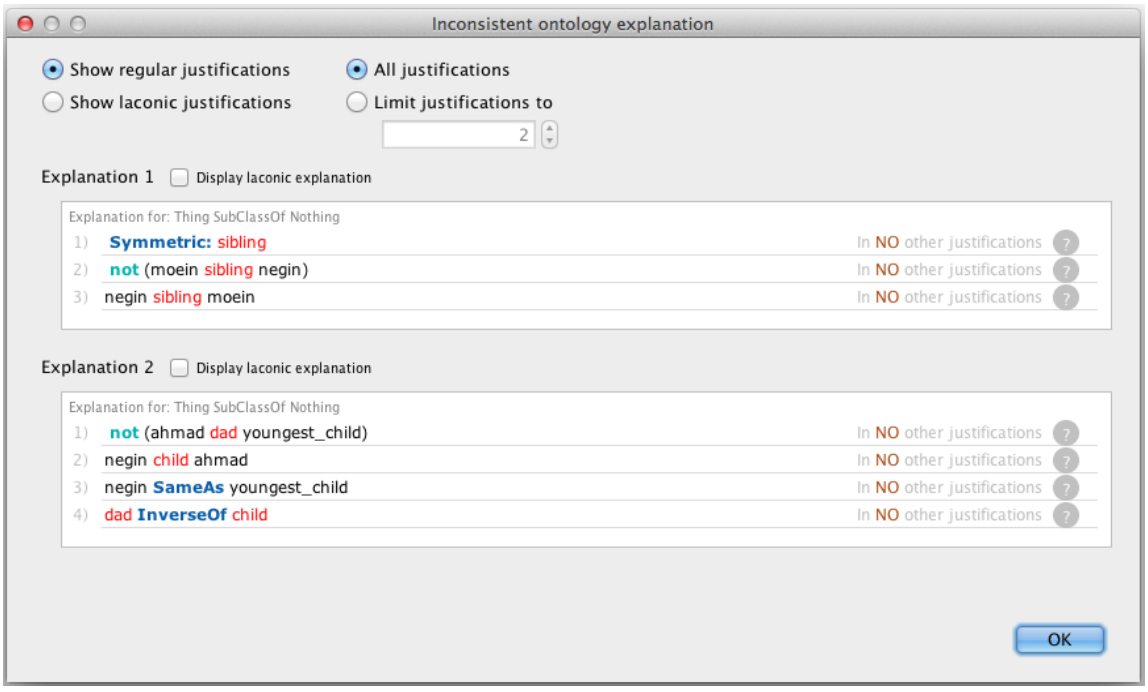

Figure 3.1: The Output of `HermiT`

3.3 Implementation of `Saffron`

`Saffron` consists of three translation related modules, and several support modules that are responsible for reading the input problem and saving the clauses in a processable format. `translation`, `gather`, and `generate-output` are the three translation related modules. The `translation` module tries to translate each CNF clause to a DL axiom. The gather module gathers all the characteristics of each individual, class and role, and descriptions of class `Thing`. Then it passes the collated information to the `generate-output` module to generate the output file in RDF.

3.3.1 The Translation Module

For every CNF clause, the `translation` module checks if there is a possible translation to DL. If it can translate the clause, it passes the resultant DL axiom to the

`gather` module, and proceeds with the rest of the clauses. Otherwise, it keeps the clause as an untranslated clause, and proceeds with the rest of the clauses.

To check all the possible translations of a CNF clause, the translator module first needs to determine the form of the clause. For example, both of the clauses in Example 33 expresses the fact that the binary predicate `r` is symmetric, and when translated to DL, the role `r` has the symmetric characteristic.

Example 33

`~r(X,Y)|r(Y,X)`

`r(X,Y)|~r(Y,X)`

To determine the form of the clause, the `translation` module extracts all the constants, unary predicates, and binary predicates, and variables from the clause being translated, and depending on these sets, and the polarities of the literals determines if the clause has any of the translatable forms. Table 3.5 shows the possible CNF forms and the corresponding DL axioms, depending on the sets of constants, unary predicates, binary predicates, and variables in a CNF clause. For example, all the clauses in Example 34 share the same sets of constants, unary predicates, binary predicates, and variables. However, the clauses 34.a to 34.d are translated to different DL axioms, and the clause 34.e is not translatable.

Example 34

34.a. `r(a,b)`

34.b. `~r(a,b)`

34.c. `r(b,a)`

34.d. ∼`r(b,a)`

34.e. `r(a,a)|r(b,b)`

set of constants = {`a,b`}

set of unary predicates = {}

set of binary predicates = {`r`}

set of variables = {}

3.3.2 The Gather and Generate-output Modules

The characteristics of a specific individual, class or role, and the description of the class `Thing` are distributed in the CNF problem. Since the translation is done clause by clause, the output of the translation module is a set of statements. Each statement describes a characteristic of an individual, a characteristic of a class, a characteristic of a role, and the class `Thing`. The `gather` module of `Saffron` collects all the statements about each individual, each class, each role or class `Thing` into separate sets. The `gather` module then passes the sets to the `generate-output` module to generate the RDF format of the DL problem. The `generate-output` module expresses each set as an RDF tag, possibly with sub-tags. All the RDF tags are output to a file with an `.owl` file extension.

Table 3.5: CNF and DL Equivalent Axioms

CNF clauses	Set of Individuals in the CNF clause	Set of Classes in the CNF clause	Set of Roles in the CNF clause	Set of Variables in the CNF clause	DL axioms
`p(a)`	{a}	{p}	{}	{}	Individual a belongs to the class P
`~p(a)`					Individual a does not belongs to the class P
`p1(a)|p2(a)|...|pN(a)|` `~q1(a)|~q2(a)|...|~qM(a)`	{a}	{p1,p2,...,pN, q1,q2,...,qM —$N \geq 0$, $M \geq 0$, and $M + N \geq 2$}	{}	{}	Individual a belongs to the union of classes `P1`, `P2`, ... and `PN`, and the complement of class `Q1`, the complement of class `Q2`, ... and the complement of class `QM`
r(a,b)	{a,b}	{}	{r}	{}	Individual a is in the r relationship with the individual b
~r(a,b)					Individual a is not in the r relationship with the individual b
`a = b`	{a,b}	{}	{=}	{}	a is a same individual as b
`a != b`					a is a different individual as b
`~p(X)|q(X)`	{}	{p,q}	{}	{X}	Class P is a subclass of class Q
`p(X)`	{}	{p}	{}	{X}	Class P is a equivalent to class `Thing`
`~p(X)`					Class `Thing` is equivalent to the complement of class P

`p1(X)|p2(X)|...|pN(X)|`
`~q1(X)|~q2(X)|...|~qM(X)`
($N \geq 0$, $M \geq 0$, and $M+N \geq 2$)
	{}	{p1,p2,...,pN, q1,q2,...,qM— $N \geq 0$, $M \geq 0$, and $M + N \geq 2$}	{}	{X}	All individuals belong to the union of classes `P1`, `P2`, ..., and `PN`, and the complements of classes `Q1`, `Q2`, ..., and `QM`

`r(X,a1)|r(X,a2)|...|r(X,aN)|` `~r(X,b1)|~r(X,b2)|...|~r(X,bM)`	{a1,a2,...,aN, b1,b2,...,bM— $N \geq 0$, $M \geq 0$, and $M + N \geq 1$}	{}	{r}	{X}	All individuals have `r` relationship with `a1`, `a2`, ..., or `aN`, or do not have `r` relationship with `b1`, `b2`, ..., or `bM`

`p1(X)|p2(X)|...|pI(X)|`
`~q1(X)|~q2(X)|...|~qJ(X)`
`r(X,a1)|r(X,a2)|...|r(x,aK)|`
`~r(X,b1)|~r(X,b2)|...|`
`~r(x,bL)`
	{a1,a2,...,aK, b1,b2,....,bL— $K \geq 0$, $L \geq 0$, and $K + L \geq 1$}	{p1,p2,...,pI, q1,q2,...,qJ— $I \geq 0$, $J \geq 0$, and $I + J \geq 1$}	{r}	{X}	All individuals belong to the union of classes `P1`, `P2`, ..., and `PI`, and the complements of classes `Q1`, `Q2`, ..., and `QJ`. Also they have `r` relationship with `a1`, `a2`, ..., or `aK`, or they do not have `r` relationship with `b1`, `b2`, ..., or `bL`

`r(X,X)`	{}	{ }	{r}	{X}	r is a reflexive relation
~r(X,X)					r is an irreflexive relation
`~r(X,Y)|r(Y,X)`	{}	{}	{r}	{X,Y}	r is a symmetric relation
`~r(X,Y)|~r(Y,X)`					r is an asymmetric relation
`~r(X,Y)|~r(Y,Z)|r(X,Z)`	{}	{}	{r}	{X,Y,Z}	r is a transitive relation
`~r(X,Y)|s(Y,X)`	{}	{}	{r,s}	{X,Y}	r is the inverse of s

3.4 Testing the Translator

Example satisfiable CNF problems has been created that only include the translatable clauses by the current version of `Saffron`. `Saffron` could translate all the clauses of these problems, the way it was expected, and the satisfiablity of all the translated problems were confirmed by `Hermit`. To check if `Saffron` is successful in translating the unsatisfiable subset of clauses of an unsatisfiable CNF problems, it has been tested over 262 unsatisfiable CNF TPTP problems that have no functions with arity

more than zero, and no predicates with arity more than two. All these problems are unsatisfiable. For each problem, the result of the translation is an OWL file in RDF syntax, which is either a full or a partial translation of the CNF problem. It is hoped that the translation preserves their unsatisfiability for partially translated problems. All the output files are passed to `HermiT`, a DL reasoner, to check if the unsatisfiability property is preserved.

By the available translator features, 10 problems are totally translated, and after running the `HermiT` reasoner over these problems, the unsatisfiability of 8 of these is confirmed by `HermiT`, and `HermiT` did not stop reasoning on the other 2 problems in 300 seconds. Of the remaining 252 problems, which are partially translated, the unsatisfiability of 61 problems is confirmed by `HermiT` , and `HermiT` did not time out on any of the problems. On average, 78% of the clauses of each of these 61 problems are successfully translated by the available features of `Saffron`. Thus, the unsatisfiability of 69 problems out of 262 translated problems is preserved. Later, in Chapter 4 the result of translating a CNF problem with only constants and predicates with arity one and two using `Saffron`, and solving using `HermiT` versus using `iProver`, which is the well tuned ATP system for EPR problems are compared. Table 3.6 and Table 3.7 shows all these problems. The last two columns show the number of clauses successfully translated to DL axioms and the percentage of successfully translated clauses over all clauses. On average, 37% of each of the 262 CNF problems were translated. Problems 67 and 71 are the two problems that are 100% translated but their unsatisfiability are not confirmed.

Table 3.6: Unsatisfiability Preserved CNF problems

#	Problem Name	Number of Clauses	Number of Translated Clauses	Percentage
1	HWV039-1	869	804	92
2	HWV039-2	865	801	92
3	HWV040-1	878	811	92
4	HWV040-2	874	808	92
5	HWV041-1	889	822	92
6	HWV041-2	885	819	92
7	HWV042-1	889	822	92
8	HWV042-2	885	819	92
9	HWV043-1	950	883	92
10	HWV043-2	947	881	93
11	HWV044-1	950	883	92
12	HWV044-2	947	881	93
13	HWV045-1	885	809	91
14	HWV045-2	881	806	91
15	HWV046-1	885	809	91
16	HWV046-2	881	806	91
17	HWV047-1	989	910	92
18	HWV047-2	985	907	92
19	HWV048-1	989	910	92
20	HWV048-2	985	907	92
21	HWV049-1	1022	915	89
22	HWV049-2	989	888	89
23	HWV050-1	828	769	92
24	HWV050-2	828	766	92
25	HWV051-1	828	769	92
26	HWV051-2	828	766	92
27	KRS004-1	4	4	100
28	PUZ002-1	12	12	100
29	PUZ029-1	15	15	100
30	PUZ035-1	15	7	46
31	PUZ035-2	16	8	50

Table 3.7: Unsatisfiability Preserved CNF problems-Continued

#	Problem Name	Number of Clauses	Number of Translated Clauses	Percentage
32	SWV421-1.060	140	106	75
33	SWV421-1.065	145	111	76
34	SWV421-1.100	180	146	81
35	SWV421-1.105	185	151	81
36	SWV421-1.200	280	246	87
37	SWV421-1.205	285	251	88
38	SWV421-1.300	380	346	91
39	SWV421-1.305	385	351	91
40	SWV421-1.360	440	406	92
41	SWV421-1.365	445	411	92
42	SWV421-1.400	480	446	92
43	SWV421-1.405	485	451	92
44	SWV421-1.460	540	506	93
45	SWV421-1.465	545	511	93
46	SWV421-1.500	580	546	94
47	SWV421-1.505	585	551	94
48	SWV422-1.060	140	106	75
49	SWV422-1.065	145	111	76
50	SWV422-1.100	180	146	81
51	SWV422-1.105	185	151	81
52	SWV422-1.200	280	246	87
53	SWV422-1.205	285	251	88
54	SWV422-1.300	380	346	91
55	SWV422-1.305	385	351	91
56	SWV422-1.360	440	406	92
57	SWV422-1.365	445	411	92
58	SWV422-1.400	480	446	92
59	SWV422-1.405	485	451	92
60	SWV422-1.460	540	506	93
61	SWV422-1.465	545	511	93
62	SWV422-1.500	580	546	94
63	SWV422-1.505	585	551	94
64	SYN060-1	7	7	100
65	SYN061-1	6	6	100
66	SYN062-1	7	7	100
67	SYN072-1	5	5	100
68	SYN096-1.008	65	65	100
69	SYN099-1.003	50	49	98
70	SYN100-1.005	82	82	100
71	SYN914-1.1	56	56	100

Chapter 4

Experiments

There are different ways for solving a real-world problem using any of the logics, Propositional Logic, Description Logic (DL), First Order Form (FOF), Conjunctive Normal Form (CNF), Effectively Propositional Form(EPR), Typed First order Form-monomorphic (TFF0), Typed First order Form-polymorphic (TFF1), Typed Higher order Form-monomorphic (THF0). A real-world problem can be first expressed in a logic. It can be then solved in that logic using an ATP system for that logic, or translated to a less expressive logic if possible. If it is translated to a less expressive logic, again the same two options of translating down (if possible) to another less expressive logic, or solving using an ATP system for that logic are available. In Figure 1.1 in Chapter 1, different *paths* for a logic problem, from the original logic in which it is expressed to an ATP system by which it is solved, is illustrated.

An experiment has been done to compare the reasoning processes through all available paths for problems in different logics (THF0, TFF1, TFF0, FOF and CNF) in terms of efficiency. Efficiency of the reasoning process through a path is determined

by the number of problems solved over the all sample problems, i.e., the chance of success in translation and solving the problem, and the average time of the whole reasoning process through the path, including translations and solving, of all problems.

The experiment was done on a machine with the following specifications.

NumberOfCPUs : 4

CPUModel: Intel(R) Xeon(TM) CPU 2.80GHz

RAMPerCPU: 756MB

OS: Linux 2.6.32.26-175.fc12.i686.PAE

Annotations. A path titled $logic_1 \cdot logic_2 \cdot \ldots \cdot logic_n$ with a chain of translators and an ATP system $translator_1 > translator_2 > \ldots > translator_{n-1} > atp_n$ implies that the logic problem is originally in logic $logic_1$, and for $1 \leq i \leq n-1$, problems in the logic $logic_i$ is translated to the logic $logic_{i+1}$ using the translator $translator_i$, and eventually the problem in the logic $logic_n$ is solved using its ATP system atp_n.

4.1 Conjunctive Normal Form

Some CNF problems can be translated to DL, and if they are EPR they can be translated to Propositional Logic. The translation of a CNF problem to DL might not be complete because there might be clauses with a function with arity greater than zero or a predicate with arity greater than two. It might not be possible to translate some clauses with only constants and predicates with arity one or two to DL because there might not be equivalent DL semantics for the clause.

The selected ATP system for solving CNF problems is `Vampire-3.0`. If a CNF

problem is EPR, the selected ATP system is `iProver-1.0`. The translator for translating CNF to DL is `Saffron-1.0`, and the translator for EPR to Propositional Logic is `EGround-1.8`. The ATP system for solving DL problems is `HermiT-1.3.8`, and the ATP system for solving Propositional Logic problems is `MiniSAT-2.2.0`.

Two experiments have been carried out over two different sets of sample problems to compare the possible paths for CNF problems. One set is the sample problems from the CASC-J6 CNF division. The other set is the set of CNF problems that are used in Chapter 2.6.4 to test `Saffron`, which include only constants and predicates with arity one or two.

Table 4.1 shows the result of the experiment on the first set of CNF problems. The column *Successful* shows the number of problems that are completely solved, and the percentage over the total number of problems. The column *Average time* shows the average time of the successfully solved problems. The columns *Timed-out* and *Failed* show the number of problems that a stage of their reasoning process is timed-out and failed, and the percentage of such problems over the total number of problems.

Table 4.1: Results of Comparing Paths from CNF over 150 Problems from CASC-J6

Path	Tools	Successful	Average time	Timed-out	Failed
CNF	Vampire-3.0	123 (82%)	2.241	27 (18%)	0 (0%)
CNF.DL	Saffron-1.0 > HermiT-1.3.8	0 (0%)	0.0	1 (0%)	149 (99%)

None of these problems are EPR, so none of them were translated to Propositional Logic. The translation of all of these CNF problems to DL were all partial, with an average of only 17% of the clauses translated to DL. This incompleteness is presumed to be the reason why none of the resultant DL problems could be shown

to be unsatisfiable by `HermiT`, and further work to extend the scope of the translator is necessary for this kind of problem.

Table 4.2 shows the result of the experiment over the second set of problems. Since all these problems are EPR, `iProver-1.0` is used to solve the problems. As illustrated in this table, `iProver-1.0` can solve more problems than `EGround-1.8` > `MiniSAT-2.2.0`, and `EGround-1.8` > `MiniSAT-2.2.0` can solve more problems than `Saffron-1.0` > `HermiT-1.3.8`.

Table 4.2: Results of Comparing Paths from CNF over 262 Problems with only constants and predicates with arity less than two

Path	Tools	Successful	Average time	Timed-out	Failed
CNF	iProver-1.0	129 (49%)	2.199	49 (18%)	84 (32%)
CNF.Prop	EGround-1.8 > MiniSAT-2.2.0	102 (38%)	0.0020	97 (37%)	63 (24%)
CNF.DL	Saffron-1.0 > HermiT-1.3.8	69 (26%)	0.079	0 (0%)	193 (73%)

There are 11 problems among these 262 problems that can be solved through all the three paths. Table 4.3 illustrates the average on the common successfully solved problems, and the number of the problems that are exclusively solved through each path that cannot be solved by others. It is interesting that although the number of the problems that `Saffron-1.0` > `HermiT-1.3.8` can solve is less than the number of problems that `iProver-1.0` or `EGround-1.8` > `MiniSAT-2.2.0` can solve, `Saffron-1.0` > `HermiT-1.3.8` can solve 46 problems that cannot be solved through the two other paths.

When more than one CPU is available, different approaches can be executed on one problem simultaneously to increase the chance of the problem being solved.

Therefore, with N processors, N paths that provide the maximum number of problem solved in this experiment are chosen. Table 4.4 shows which path is the best to be picked as an additional path to maximize the number of problems solved. Out of 262 sample problems in CNF, 175 problems can be solved through all paths together. The two paths shown in Table 4.4 all together can solve all of these problems, so adding more than two CPUs does not increase the chance of a problem being solved. The number of exclusive problems at each row is the number of problems that can be solved only by the path at each row comparing to all other possible paths up to that row. The total solved is the total number of problems that can be solved by all the N. The average time for N processors is the average of minimum computation time among all N paths for every problem.

Table 4.3: Common and Exclusive Solved Problems Through Paths from CNF

Path	Number of Exclusive Problems	Average Time Over Common Problems
CNF.DL	46	0.010
CNF	15	0.001
CNF.Prop	0	0.000

Table 4.4: Paths from CNF in Parallel

CPUs	Path	Exclusive Problems	Total Solved	Average Time
1	CNF	129	129	2.182
2	CNF.DL	46	175	0.526

4.2 First Order Form (FOF)

FOF problems can be translated to CNF. If a problem is translated to CNF then it can be translated to DL or Propositional Logic, in the way explained in Section 4.1.

The selected ATP system for solving FOF problems is `Vampire-3.0`. The translator for translating FOF to CNF is `ECNF-1.8`.

The sample problems in FOF are sample problems for the CSC-J6 ATP System Competition. Table 4.5 shows the result of the experiment on this set of FOF problems. If the problem translated to CNF is EPR then `iProver-1.0` is used as the ATP system. Otherwise, `Vampire-3.0` is used. Table 4.5 shows the combined results for EPR and non-EPR translated problems to CNF. The column *Total* shows the number of available test problems for each path. Table 4.6 shows that among the problems that are translated to CNF, how many are EPR.

Table 4.5: Results of Comparing Paths from FOF

Paths	Tools	Total	Successful	Average time	Timed-out	Failed
FOF	Vampire-3.0	500	396 (79%)	3.05	104 (20%)	0 (0%)
FOF.CNF	ECNF-1.8 > (Vampire-3.0 or iProver-1.0)	500	294 (58%)	1.505	157 (31%)	49 (9%)
FOF.CNF.DL	ECNF-1.8 > Saffron-1.0 > HermiT-1.3.8	500	20 (4%)	0.467	93 (18%)	387 (77%)
FOF.CNF.Prop	ECNF-1.8 > EGround-1.8 > MiniSAT-2.2.0	33	13 (39%)	0.073	3 (9%)	17 (51%)

Table 4.6: Results of Comparing Paths from FOF to CNF

Paths	Tools	Total	Successful	Average time	Timed-out	Failed
FOF.EPR	ECNF-1.8 > iProver-1.0	468	273 (58%)	1.309	147 (31%)	48 (10%)
FOF.CNF	ECNF-1.8 > Vampire-3.0	32	21 (65%)	4.242	10 (31%)	1 (3%)

Table 4.7 shows that by increasing the number of CPUs, which path is the best to be picked as an additional path to maximize the number of problems solved. Out of 500 sample problems in CNF, 425 problems can be solved through all paths together. The three paths shown in Table 4.7 all together can solve all of these problems, so adding the path than three CPUs will not increase the chance of a problem being solved. Among the 16 problems that are exclusively solved by the path FOF.CNF, 13 problems are EPR which are solved using *iProver-1.0*

Table 4.7: Paths from FOF in Parallel

CPUs	Path	Exclusive Problems	Total Solved	Average Time
1	FOF	396	396	3.042
2	FOF.CNF	16	412	2.366
3	FOF.CNF.DL	13	425	2.095

4.3 Typed First-order Form - monomorphic

TFF0 problems can be translated to FOF or CNF. If a problem is translated to FOF, then it can be translated to CNF in the way explained in Section 4.2. If a problem is translated to CNF, then it can be translated to DL or Propositional Logic, in the way explained in Section 4.1.

The selected ATP system for solving TFF0 problems is `Princess-120604`. The translator for translating TFF0 to FOF is `Monotonox-2FOF-e3c1636`, and the translator for translating TFF0 to CNF is `Monotonox-2CNF-e3c1636`.

The sample problems in TFF0 are all the TFF0 problems in the TPTP that were available when this experiment was initiated. Table 4.8 shows the result of the experiment on this set of TFF0 problems.

Table 4.9 shows that by increasing the number of CPUs, which path is the best being picked as an additional path to maximize the number of problems solved. Out of 97 sample problems in CNF, 44 problems can be solved through all paths together. The four paths shown in Table 4.9 all together can solve all of these problems, so adding more than four CPUs will not increase the chance of a problem to be solved.

4.4 Typed First-order From - polymorphic

TFF1 problems can be translated to TFF0 or FOF. If a problem is translated to TFF0, then it can be translated to FOF or CNF in the way explained in Section 4.3. If a problem is translated to FOF, then it can be translated to CNF, in the way explained in Section 4.2.

The selected ATP system for solving TFF1 problems is `Alt-Ergo-0.94`. The translator for translating TFF1 to TFF0 is `Why3-TFF0-0.71`, and the translator for translating TFF1 to FOF is `Why3-FOF-0.71`.

The sample problems in TFF1 are all the Polymorphic typed First-order Form (PFF) problems in the TPTP that were available when the experiment was initiated. Table 4.10 shows the result of the experiment on this set of TFF0 problems. If a problem is eventually translated to CNF, and it is EPR, then it can be translated to Propositional Logic, but none of the CNF problems are EPR.

Table 4.11 shows that by increasing the number of processors which path is the best to be picked as an additional path to the path set to maximize the number of problems solved. Out of the 987 sample problems, 385 problems can be solved by all alternatives together. The four paths shown in Table 4.11 all together can solve all of these problems, so adding more than four CPUs will not increase the chance of a problem being solved.

4.5 Typed Higher-order From - monomorphic

THF0 problems can be translated to TFF0 and FOF. If a problem is translated to TFF0 then it can be translated to FOF or CNF, in the way explained in Section 4.3. If a problem is translated to FOF then it can be translated to CNF, in the way explained in Section 4.2.

The selected ATP system for solving THF0 problems is `Isabelle-HOT-2013`. The translators for translating THF0 to TFF0 and THF0 to FOF are `Isabelle-2TF0` and `Isabelle-2FOF`.

An experiment has been carried out over sample problems from the CASC-J6 THF0 division. Table 4.12 shows the result of the experiment on this set of THF0 problems. There are thirteen paths for all THF0 problems. The Average time is the average time over all successful problems in each row. If a THF0 problem is eventually translated to CNF and it is also EPR, it can be also translated to Propositional Logic. As it is illustrated in Table 4.12, only one problem is EPR. However, this problem was not recognized to be EPR when it is translated through

the path THF0.TFF0.FOF.CNF, so there was no problem solved through the path THF0.TFF0.FOF.CNF.Prop.

Table 4.13 shows that by increasing the number of processors which path is the best to be picked as an additional path to the path set to maximize the number of problems solved. Out of 200 sample problems in THF0, 114 problems can be solved through all paths together. The four paths shown in Table 4.13 all together can solve all of these problems, so adding more than four CPUs will not increase the chance of a problem being solved.

Table 4.8: Results of Comparing Paths from TFF0

Paths	Tools	Total	Successful	Average time	Timed-out	Failed
TFF0.FOF	Monotonox-2FOF-e3c1636 > Vampire-2.6	97	36 (37%)	1.423	58 (59%)	3 (3%)
TFF0.FOF.CNF	Monotonox-2FOF-e3c1636 > ECNF-1.6 > Vampire-2.6	97	31 (31%)	1.025	61 (62%)	5 (5%)
TFF0.CNF	Monotonox-2CNF-e3c1636 > Vampire-2.6	97	24 (24%)	1.766	31 (31%)	42 (43%)
TFF0	Princess-120604	97	12 (12%)	12.061	84 (86%)	1 (1%)
TFF0.CNF.Prop	Monotonox-2CNF-e3c1636 > EGround-1.6 > MiniSAT-2.2.0	27	0 (0%)	0.0	4 (14%)	23 (85%)
TFF0.FOF.CNF.Prop	Monotonox-2FOF-e3c1636 > ECNF-1.6 > EGround-1.6 > MiniSAT-2.2.0	27	0 (0%)	0.0	0 (0%)	27 (100%)
TFF0.CNF.DL	Monotonox-2CNF-e3c1636 > Saffron-1.0 > HermiT-1.3.8	15	2 (13%)	0.231	0 (0%)	13 (86%)
TFF0.FOF.CNF.DL	Monotonox-2FOF-e3c1636 > ECNF-1.6 > Saffron-1.0 > HermiT-1.3.8	10	5 (50%)	0.216	0 (0%)	5 (50%)

Table 4.9: Paths from TFF0 in Parallel

CPUs	Path	Exclusive Problems	Total Solved	Average Time
1	TFF0.FOF	36	36	1.384
2	TFF0.FOF.CNF.DL	5	41	1.238
3	TFF0	2	43	1.31
4	TFF0.CNF.DL	1	44	1.282

Table 4.10: Results of Comparing Paths from TFF1

Paths	Tools	Total	Successful	Average time	Timed-out	Failed
TFF1.FOF.CNF	Why3-FOF-0.71 > ECNF-1.6 > Vampire-2.6	987	348 (35%)	0.761	609 (61%)	30 (3%)
TFF1.TFF0.FOF	Why3-TFF0-0.71 > Monotonox-2FOF-e3c1636 > Vampire-2.6	987	329 (33%)	0.898	627 (63%)	31 (3%)
TFF1.FOF	Why3-FOF-0.71 > Vampire-2.6	987	317 (32%)	0.806	640 (64%)	30 (3%)
TFF1	Alt-Ergo-0.94	987	312 (31%)	1.032	675 (68%)	0 (0%)
TFF1.TFF0.CNF	Why3-TFF0-0.71 > Monotonox-2CNF-e3c1636 > Vampire-2.6	987	276 (27%)	1.689	658 (66%)	53 (5%)
TFF1.TFF0	Why3-TFF0-0.71 > Princess-120604	987	33 (3%)	12.642	924 (93%)	30 (3%)
TFF1.TFF0.FOF.CNF	Why3-TFF0-0.71 > Monotonox-2FOF-e3c1636 > ECNF-1.6 > Vampire-2.6	986	16 (1%)	0.835	32 (3%)	938 (95%)
TFF1.TFF0.FOF.CNF.DL	Why3-TFF0-0.71 > Monotonox-2FOF-e3c1636 > ECNF-1.6 > Saffron-1.0 > HermiT-1.3.8	987	2 (0%)	0.145	1 (0%)	984 (99%)
TFF1.TFF0.CNF.DL	Why3-TFF0-0.71 > Monotonox-2CNF-e3c1636 > Saffron-1.0 > HermiT-1.3.8	987	0 (0%)	0.0	1 (0%)	986 (99%)
TFF1.FOF.CNF.DL	Why3-FOF-0.71 > ECNF-1.6 > Saffron-1.0 > HermiT-1.3.8	987	0 (0%)	0.0	31 (3%)	956 (96%)

Table 4.11: Paths from TFF1 in Parallel

CPUs	Path	Exclusive Problems	Total Solved	Average Time
1	TFF1.FOF.CNF	348	348	0.761
2	TFF1	23	371	0.597
3	TFF1.TFF0.FOF	9	380	0.531
4	TFF1.FOF	5	385	0.498

Table 4.12: Results of Comparing Paths from THF0

THF0.FOF	Isabelle-2FOF > Vampire-3.0	200	78 (39%)	8.953	80 (40%)	42 (21%)
THF0	Isabelle-HOT-2013	200	78 (39%)	11.856	122 (61%)	0 (0%)
THF0.TFF0.CNF	Isabelle-2TF0 > Monotonox-2CNF-e3c1636 > Vampire-3.0	200	52 (26%)	5.745	75 (37%)	73 (36%)
THF0.TFF0	Isabelle-2TF0 > Princess-120604	200	24 (12%)	14.893	173 (86%)	3 (1%)
THF0.TFF0.FOF.CNF	Isabelle-2TF0 > Isabelle-2FOF > ECNF-1.8 > Vampire-3.0	200	0 (0%)	0.0	196 (98%)	4 (2%)
THF0.FOF.CNF	Isabelle-2FOF > ECNF-1.8 > Vampire-3.0	200	0 (0%)	0.0	194 (97%)	6 (3%)
THF0.TFF0.FOF	Isabelle-2TF0 > Isabelle-2FOF > Vampire-3.0	200	77 (38%)	11.411	82 (41%)	40 (20%)
THF0.TFF0.CNF.Prop	Isabelle-2TF0 > Monotonox-2CNF-e3c1636 > EGround-1.8 > MiniSAT-2.2.0	1	0 (0%)	0.0	0 (0%)	1 (100%)
THF0.TFF0.FOF.CNF.Prop	Isabelle-2TF0 > Isabelle-2FOF > ECNF-1.8 > EGround-1.8 > MiniSAT-2.2.0	1	0 (0%)	0.0	0 (0%)	1 (100%)
THF0.FOF.CNF.Prop	Isabelle-2FOF > ECNF-1.8 > EGround-1.8 > MiniSAT-2.2.0	1	0 (0%)	0.0	0 (0%)	1 (100%)

Table 4.13: Paths from THF0 in Parallel

CPUs	Path	Exclusive Problems	Total Solved	Average Time
1	THF0.FOF	78	78	8.838
2	THF0	33	111	9.374
3	THF0.TFF0.FOF	2	113	9.426
4	THF0.TFF0.CNF	1	114	8.148

Chapter 5

Conclusion

5.1 Review of the Thesis

In this research, a survey of some logics has been presented. The logics include Typed Higher Order Form (THF0), Typed First Order Form - polymorphic (TFF1), Typed First Order Form - monomorphic (TFF0), First Order Form (FOF), Conjunctive Normal Form (CNF), Effectively Propositional Form (EPR), Description Logic (DL), and Propositional Logic (Prop). For each logic, properties, syntax and semantics are briefly explained, and common Automated Theorem Proving (ATP) systems, and translators to less expressive logics are introduced. The expressivities of these logics are different. Except for Prop, DL, and EPR, none of these logics are decidable. A satisfiability transforming procedure for translating CNF to DL, and its implementation as `Saffron` is presented. There are different ways of solving a problem expressed in a logical form. It can be solved directly using the ATP system of that logic or if possible it can be translated to a less expressive form. When it is translated to a less

expressive form, again the same two options are available. In an experiment is this research, all possible ways of solving a problem in a logical form is compared.

5.2 Contributions and Conclusions

The results of the experiments over different ways of solving problem show that more TFF0 and TFF1 problems are solved by translations to less expressive logics than by using the ATP systems of the source logics. The number of THF0 problems solved by the THF0 ATP system and by translation to FOF and then using the FOF ATP system is the same. However, the translation approach is quicker.

When more than one CPU is available, simultaneously solving a CNF problem with only constants and predicates with arity less than two, and a THF0 problem through several paths remarkably increases the chance of the problem to be solved, comparing to solving the problem through the best paths from CNF and THF0.

In this research, `Saffron`, a CNF to DL translator is developed. No such translator was available before. DL is more expressive than Prop, and less expressive than EPR. DL, as a specific logic for expressing ontologies is not a part of TPTP world, and there is no TPTP syntax for it. However, it can now be embedded in the TPTP world, so there is a chance to compare this logic with other TPTP logics. Some CNF problems (49 out of 262) are solved by translation to DL using `Saffron`, and then using `HermiT` to solve them. None of these problems can be solved using `iProver`, an appropriate

EPR ATP system, with time limit of 60 seconds. This shows that there are some types of CNF problems that can be uniquely solved using `Saffron` and `HermiT` in a reasonable amount of time.

5.3 Future Work

In the future `Saffron` and `HermiT` can be combined together to solve CNF problems by translating them to DL and then reasoning about them. The implementation of `Saffron` can be optimized, so the translation can be done faster. More features can also be added to `Saffron`, to translate more CNF clauses. This can be done by a study over the output of the experiment carried out in Chapter 4, analysis of problems that are successfully translated, and the problems that their unsatisfiability are confirmed.

References

[1] D. Loveland, *Automated Theorem Proving : A Logical Basis*. Elsevier Science, 1978.

[2] A. Riazanov and A. Voronkov, "The Design and Implementation of Vampire," *AI Communications*, vol. 15(2-3), pp. 91–110, 2002.

[3] P. Rummer, I. Cervesato, H. Veith, and A. Voronkov, "A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic," in *the 15th International Conference on Logic for Programming Artificial Intelligence and Reasoning (Doha, Qatar)*, vol. 5330, pp. 274–289, Springer-Verlag, 2008.

[4] G. Sutcliffe, "The TPTP World - Infrastructure for Automated Reasoning," in *Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning* (E. Clarke and A. Voronkov, eds.), no. 6355 in Lecture Notes in Artificial Intelligence, pp. 1–12, Springer-Verlag, 2010.

[5] K. Claessen, A. Lilliestr\"om, and N. Smallbone, "Sort It Out With Monotonicity Translating Between Many-sorted and Unsorted First-order Logic," in *Bjorner, N., Sofronie-Stokkermans, V. (eds.) CADE-23*, vol. 6803, pp. 207–221, Springer, 2011.

[6] S. Schulz, "E: A Brainiac Theorem Prover," *AI Communications*, vol. 15(2/3), pp. 111–126, 2002.

[7] G. Sutcliffe, "The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0," *Journal of Automated Reasoning*, vol. 43, no. 4, pp. 337–362, 2009.

[8] G. Sutcliffe, "SystemOnTPTP," in *Proceedings of the 17th International Conference on Automated Deduction* (D. McAllester, ed.), no. 1831 in Lecture Notes in Artificial Intelligence, pp. 406–410, Springer-Verlag, 2000.

[9] "The International Joint Conference on Automated Reasoning." `http://www.ijcar.org`.

[10] R. Nieuwenhuis, "The Impact of CASC in the Development of Automated Deduction Systems," *AI Communications*, vol. 15, no. 2-3, pp. 77–78, 2002.

[11] V. P. Nelson, H. T. Nagle, B. D. Carroll, and D. Irwin, *Digital Logic Circuit Analysis and Design*. Prentice Hall, 1995.

[12] H. Hoos and T. Stützle, "Stochastic Local Search, Foundations and Applications," in *A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning*, vol. 1, ch. 6, pp. 209–210, Elsevier, 2001.

[13] M. Huth and M. Ryan, "Logic in Computer Science: Modeling and Reasoning about Systems," pp. 2–5, 2000.

[14] N. Eén and N. Sörensson, "MiniSat - A SAT Solver with Conflict-Clause Minimization," in *Posters of the 8th International Conference on Theory and Applications of Satisfiability Testing* (F. Bacchus and T. Walsh, eds.), 2005.

[15] M. W. Moskewizc, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proc. of 12th Int. Con- ference on Computer Aided Verification*, vol. 1855, 2001.

[16] "SAT-Race 2008 Results." `http://baldur.iti.uka.de/sat-race-2008/results.html`.

[17] P. Hitzler, M. M. Krötzsch, and S. Rudolph, *Foundations of Semantic Web Technologies*. CRC Press, 2008.

[18] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang, "The Manchester OWL Syntax," in *OWLed*, vol. 216, CEUR-WS.org, Novemeber 2006.

[19] D. Tsarkov and I. Horrocks, "FaCT++ Description Logic Reasoner: System Description," *LNCS*, pp. 292–297, 2006.

[20] R. Shearer, B. Motik, and I. Horrocks, "HermiT: A Highly-Efficient OWL Reasoner.," in *OWLED* (C. Dolbear, A. Ruttenberg, and U. Sattler, eds.), vol. 432 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2008.

[21] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski, "SPASS Version 3.5," in *22nd International Conference on Automated Deduction. CADE 2009*, vol. 5663, 2009.

[22] L. Moura and N. Bjorner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems. 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008*, vol. 4963, pp. 337–340, Springer, 2008.

[23] P. Baumgartner, A. Fuchs, and C. Tinelli, "Darwin - A Theorem Prover for the Model Evolution Calculus," in *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning* (G. Sutcliffe, S. Schulz, and T. Tammet, eds.), 2004.

[24] G. Sutcliffe, S. Schulz, K. Claessen, and P. Baumgartner, "The TPTP Typed First-order Form with Arithmetic," in *Bjorner, N. and Voronkov, A. (eds.) LPAR-18*, vol. 7180, Springer, 2012.

[25] A. Nonnengart and C. Weidenbach, "Computing Small Clause Normal Forms," *Journal of Applied Logic*, vol. 1, pp. 335–367, 2001.

[26] K. Korovin, "iProver - An Instantiation-Based Theorem Prover for First-order Logic (System Description)," in *Proceedings of the 4th International Joint Conference on Automated Reasoning* (P. Baumgartner, A. Armando, and D. Gilles, eds.), no. 5195 in Lecture Notes in Artificial Intelligence, pp. 292–298, 2008.

[27] S. Schulz and G. Sutcliffe, "System Description: GrAnDe 1.0," in *Proceedings of the 18th International Conference on Automated Deduction* (A. Voronkov, ed.), no. 2392 in Lecture Notes in Artificial Intelligence, pp. 280–284, Springer-Verlag, 2002.

[28] M. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood, "Deductive composition of astronomical software from subroutine libraries," in *the 12th International Conference on Automated Deduction (CADE-12), Nancy, France, June 1994*, pp. 341–355, Springer-Verlag, 1994.

[29] F. Bobot, S. Conchon, E. Contejean, M. Iguernelala, A. Mahboubi, A. Mebsout, and G. Melquiond, "A Simplex-based extension of Fourier-Motzkin for solving linear integer arithmetic," in *IJCAR 2012: Proceedings of the 6th International Joint Conference on Automated Reasoning* (B. Gramlich, D. Miller, and U. Sattler, eds.), vol. 7364 of *Lecture Notes in Computer Science*, (Manchester, UK), pp. 67–81, Springer, June 2012.

[30] "Interpretation (logic)." `http://en.wikipedia.org/wiki/Interpretation_(logic)`.

[31] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL : A Proof Assistant for Higher-Order Logic*, vol. 2283. Springer, 2002.

[32] C. Benzmüller and L. Paulson in *Festschrift in Honor of Peter B. Andrews on His 70th Birthday* (C. Benzmüller, C. E. Brown, J. Siekmann, and R. Statman, eds.), Studies in Logic, Mathematical Logic and Foundations, ch. Exploring Properties of Normal Multimodal Logics in Simple Type Theory with LEO-II, College Publications, 2008.

[33] C. E. Brown, "Satallax: An Automatic Higher-Order Prover," *LNCS*, vol. 7364, pp. 111–117, 2012.

[34] P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli, "Computing Finite Models by Reduction to Function-Free Clause Logic," *Journal of Applied Logic*, vol. 7, no. 1, pp. 58–74, 2009.