2015-12-10

Automated Theorem Proving by Translation to Description Logic

Negin Arhami
University of Miami, n.arhami@umiami.edu

UNIVERSITY OF MIAMI

AUTOMATED THEOREM PROVING BY TRANSLATION TO DESCRIPTION LOGIC

By

Negin Arhami

A DISSERTATION

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Coral Gables, Florida

December 2015

UNIVERSITY OF MIAMI

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

AUTOMATED THEOREM PROVING BY TRANSLATION TO DESCRIPTION LOGIC

Negin Arhami

Approved:

Geoff Sutcliffe, Ph.D. Associate Professor of Computer Science	Ubbo Visser, Ph.D. Associate Professor of Computer Science
Dilip Sarkar, Ph.D. Associate Professor of Computer Science	Dean of the Graduate School
Peter Lewis, Ph.D. Associate Professor of Philosophy	

ARHAMI, NEGIN (Ph.D., Computer Science)
Automated Theorem Proving by Translation to Description Logic (December 2015)

Abstract of a dissertation at the University of Miami.

Thesis supervised by Professor Geoff Suttcliffe.
No. of pages in text. (132)

Many Automated Theorem Proving (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 logical forms. On the other hand, the ATP systems for less expressive forms have benefited from more years of development and testing. There is a trade-off between the expressivity of a logical form, and the capabilities of the available ATP systems. Different ATP systems and translators can be combined to solve a problem expressed in a given logical form. In this research, an experiment has been designed and carried out to compare all different possible ways of trying to solve a problem, using the following logical forms in increasing order of expressivity: Propositional Logic, Description Logic, Effectively Propositional form, Conjunctive Normal Form, First Order Form, Typed First order form-monomorphic, Typed First order form-polymorphic, Typed Higher order form-monomorphic. In this dissertation, the properties, syntax, and semantics of each target logical form are briefly described. For each form, the most popular ATP systems and translators for translating to less expressive forms are introduced.

Problems in logics more expressive than Conjunctive Normal Form can be translated directly to Conjunctive Normal Form, or indirectly by translation via inter-

mediate logics. No translator was available to translate from Conjunctive Normal Form to Description Logic, which sits between Effectively Propositional form and Propositional Logic in terms of expressivity. `Saffron` a Conjunctive Normal Form to Description Logic translator, has been developed, which fills the gap between Conjunctive Normal Form and Description Logic. Moreover, Description Logic Form (DLF), a new syntax for Description Logic, has been designed. Automated theorem proving by translation to Description Logic is now an alternative way of solving problems expressed in logics more expressive than Description Logic, by combining necessary translators from those logics to Conjunctive Normal Form, `Saffron`, and a Description Logic ATP system.

For

My Family, My Strength, My Backbone, and My Role Models:

Ahmad Arhami, Masomeh Parchamdar,

Dr. Ashkon Arhami, Dr. Ehsan Arhami, and Moeindokht Arhami

Acknowledgements

Above all, I wish to express my greatest appreciation 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.

Most importantly, I would like to express my deepest gratitude to my beloved family, to whom this dissertation is dedicated. Their generous support, continuous encouragement, and constant love have sustained me throughout my life. My family, Ahmad Arhami, Masomeh Parchamdar, Dr. Ashkon Arhami, Dr. Ehsan Arhami, and Moeindokht Arhami have always been my role models because of their patience, diligence and strength.

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

I would like to thank the other members of my committee, Dr. Peter Lewis and Dr. Dilip Sarkar, for reviewing my research work and dissertation.

I would also like to thank other faculties and staffs of the department of Computer Science at University of Miami for providing a comfortable educational environment.

Last but not least, My sincere thanks goes to Dr. Ahamad Shafie Dehabad, Dr. Farzad Didehvar, and Dr. Abbas Nowzari, my undergraduate and graduate professors at my home universities, for believing in me and establishing the basis for the start of my Ph.D. program.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sound reasoning deals primarily 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.

Example 1

$$\begin{aligned} \text{A} = \{ &\text{ All even numbers are divisible by two,} \\ &\text{ Four is an even number}\} \\ \text{C} = &\text{ Four is divisible by two} \end{aligned}$$

Testing whether or not 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. The other perspective is the *syntactic* perspective. In this perspective, without any mathematical knowledge, 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". Sound reasoning processes typically use the syntactic perspective to establish that a conjecture is a theorem.

Logic is used to formalize the notions of a domain of interest. The logical forms of interest in this research are Propositional Logic, Description Logic, Effectively Propositional form, Conjunctive Normal Form, First Order Form, Typed First order form-monomorphic, and Typed First order form-polymorphic, Typed Higher order form-monomorphic. A logic consists of a syntax and a semantics. The syntax is used to write statements (about some domain of interest) as a set of formalized statements, *formulae*. A *logic problem* consists of some *axiom* formulae (the fact statements) and a *conjecture* formula (the conclusion).

The semantics of a logic is expressed in terms of *interpretations*. An interpretation of a logic problem relates the symbols of the logic problem to entities in the domain of interest, and assigns `TRUE` or `FALSE` to each formula of the logic 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 logic problem is a *theorem* if it is a logical consequence of the axioms of the logic problem. On the contrary, the conjecture of a logic problem is a *nontheorem* if it is not a logical

consequence of the axioms of the logic problem. The set of axioms and a conjecture of a logic problem is *countersatisfiable*[1] if and only if the conjecture is nontheorem.

A transformation $T : F \longmapsto T(F)$ maps a Formulae F to transformed set of formulae $T(F)$. The transformation T is *satisfiability preserving* means if $A = \{F_1, F_2, ..., F_n\}$ is satisfiable, then $\bigcup_{i=1}^{n} T(F_i)$ is satisfiable. The transformation T is *countersatisfiability preserving* means if C is not a logical consequence of A, then $T(C)$ is not a logical consequence of $\bigcup_{i=1}^{n} T(F_i)$. The transformation T is *countersatisfiability-satisfiability preserving* means if C is not a logical consequence of A, then $\bigcup_{i=1}^{n} T(F_i) \cup T(\sim C)$ is satisfiable. A transformation of a set of formulae is *sound* if and only if it is satisfiability preserving, countersatisfiability preserving, and countersatisfiability-satisfiability preserving.

One way to check if the conjecture of a logic problem is a theorem is *proof by contradiction*. Proof by contradiction checks if the set consisting of the axioms and the negated conjecture is unsatisfiable. A common approach is to apply sound 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 the `FALSE` formula, then it is known that the original set is unsatisfiable, and hence that the conjecture is a theorem of the axioms. From now on in this dissertation, a logic problem expressed in this form, i.e. a set consisting of the axioms and the negated conjecture, is called a *PbC logic problem* (**P**roof **b**y **C**ontradiction logic problem).

[1] The SZS ontology [1] supplies the definitions of status values such as "theorem", and "countersatisfiable".

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

To solve a (PbC) logic problem, it is checked if a (PbC) logic problem is theorem (unsatisfiable) or nontheorem (satisfiable). From now on, "problems" refers to both "logic problems" and "PbC logic problems", unless explicitly mentioned.

1.1 Automated Theorem Proving

Automated Theorem Proving (ATP) [2] is concerned with the development of automatic techniques and computer programs for solving problems. An *ATP system* is a computer program that automatically solves a problem. Many logics have ATP systems available. Examples include `Vampire` [3] for First Order Form and Conjunctive Normal Form, `CVC4` [4] for Typed First order form-monomorphic, and `Satallax` [5] for Typed Higher order form.

It is possible to translate a problem in one logic to another logic. A translation from a problem in *source_logic* to *destination_logic* is a transformation in which the

formulae of the problem, $\{F_1, F_2, ..., F_n\}$, are in *source_logic*, and $\bigcup_{i=1}^{n} T(F_i)$ are in *destination_logic*. Many sound translators for translating a problem in one logic to another one have been implemented. Examples of sound translators are `ECNF` [6] for translating from First Order Form to Conjunctive Normal Form and `Monotonox` [7] for translating from Typed First order form-monomorphic to First Order Form or Conjunctive Normal Form. For the goals of this research, only translations from more expressive logic to less expressive ones are of interest.

1.2 The Thousands of Problems for Theorem Provers World

The *Thousands of Problems for Theorem Provers* (TPTP) world [8] includes the TPTP problem library [9], and the *Thousands of Solutions for Theorem Provers* (TSTP) solution library. TPTP library is a set of standard test problems for ATP systems. The TSTP library is a set of ATP systems' solutions for the TPTP problems. The TPTP world also includes tools to support these libraries. The TPTP allows as an appropriate, convenient and repeatable testing, evaluating and comparing ATP systems. The home page for the TPTP is `http://www.tptp.org`.

1.2.1 The TPTP Library

The TPTP library is the source of test problems in this research. The TPTP problems cover six main defined fields: logic, mathematics, computer science, science and engineering, social sciences, and other. Each field is divided into *domains*, for example, **H**ard**W**are **V**erification(HWV) and **S**oft**W**are **V**erification.

TPTP problems and solutions are written in the TPTP syntax. The TPTP syntax enables convenient communication between different systems and researchers. The TPTP syntax is flexible and extensible, and easily processed by both humans and computers. The syntax is similar to Prolog syntax, so ATP systems developed in Prolog have less overhead caused by preprocessing input TPTP problems. Additionally, if an ATP system produces the solution in the TPTP syntax, the solution can be processed and evaluated by the tools provided by TPTP. The TPTP tools are explained in Section 1.2.2.

Several TPTP problems might share a common set of axioms. For the purpose of avoiding writing the same set of axioms repeatedly in different TPTP problems, the common axioms can be written in a separate TPTP axiom file. TPTP problems may include a TPTP axiom file using an `include` directive.

The latest version of the TPTP supports the following logical forms: Propositional Logic (PL), Description Logic (DL), First Order Form (FOF), Conjunctive Normal Form (CNF), Effectively Propositional Form (EPR), Typed First order form-monomorphic (TF0), Typed First order form-polymorphic (TF1), and Typed Higher order form-monomorphic (TH0).

Each TPTP problem and axiom file includes up to three sections. Example 2 is an example of a TPTP problem. The first section is a header with information about the file for users. This section is in a form of comments. The second section contains include directives. The last section contains annotated formulae.

Example 2

```
%--------------------------------------------------------------------------
% File     :  SWV041+1 :  TPTP v6.1.0.  Bugfixed v3.3.0.
% Domain   :  Software Verification
% Problem  :  Unsimplified proof obligation gauss_init_0077
% Version  :  [DFS04] axioms :  Especial.
% English  :  Proof obligation emerging from the init-safety verification for
%             the gauss program.  init-safety ensures that each variable or
%             individual array element has been assigned a defined value before
%             it is used.

% Refs       :  [Fis04] Fischer (2004), Email to G. Sutcliffe
%            :  [DFS04] Denney et al.  (2004), Using Automated Theorem Provers
% Source     :  [Fis04]
% Names      :  gauss_init_0077 [Fis04]

% Status     :  Theorem
% Rating     :  0.28 v6.1.0, 0.23 v6.0.0, 0.22 v5.5.0
% Syntax     :  Number of formulae      :    100 (   64 unit)
%               Number of atoms         :    271 (   81 equality)
%               Maximal formula depth   :     18 (    4 average)
%               Number of connectives   :    176 (    5 ~  ;   17  |;   95  &)
%                                            (    5 <=>;   54 =>;    0 <=)
%                                            (    0 <~>;    0 ~|;    0 ~&)
%               Number of predicates    :      6 (    1 propositional; 0-2 arity)
%               Number of functors      :     38 (   20 constant; 0-4 arity)
%               Number of variables     :    170 (    0 singleton; 170 !;  0 ?)
%               Maximal term depth      :      9 (    1 average)
% SPC        :  FOF_THM_RFO_SEQ

% Comments   :
% Bugfixes   :  v3.3.0 - Bugfix in SWV003+0
%--------------------------------------------------------------------------
%----Include NASA software certification axioms
include('Axioms/SWV003+0.ax').
%--------------------------------------------------------------------------
%----Proof obligation generated by the AutoBayes/AutoFilter system
fof(gauss_init_0077,conjecture,
      ( ( geq(minus(n330,n1),n0)
      & geq(minus(n410,n1),n0) )
   => !  [A] :
        ( ( leq(n0,A)
          & leq(A,n2) )
       => !  [B] :
            ( ( leq(n0,B)
              & leq(B,minus(n0,n1)) )
           => a_select3(tptp_const_array2(dim(n0,n3),dim(n0,n2),uninit),B,A) = init ) )
)).

%----Automatically generated axioms

fof(gt_5_4,axiom,
    ( gt(n5,n4) )).

...

fof(finite_domain_3,axiom,
    ( !  [X] :
        ( ( leq(n0,X)
          & leq(X,n3) )
       => ( X = n0
          | X = n1
          | X = n2
          | X = n3 ) ) )).
%--------------------------------------------------------------------------
```

The general form of an annotated formula is *logical_form*(*name*, *formula_role*, *formula*, *annotations*). The *logical_form* keywords for each logical form are shown in Table 1.1. Each annotated formula is identified by a unique *name* field. The semantics of the annotated formula is described by the *formula_role* field, and is used by ATP systems in reasoning. Examples of a formula roles are `axiom`, `conjecture`, `type`, and `definition`. The *formula* field is the formula. The *annotations* field is optional. It may contain the source that the formula came from, some information for users, or both. Example 2 includes four annotated FOF formulae.

Table 1.1: Logical Forms Supported by TPTP

Logic	*Logical Form* Keyword
PL	`cnf`
EPR	`cnf`
CNF	`cnf`
FOF	`fof`
TF0	`tff`
TF1	`tff`
TH0	`thf`

1.2.2 The TPTP and TSTP Tools

A suite of tools is available to facilitate the maintenance and use of the TPTP and TSTP libraries. These tools, as well as many other related tools, are available online at `www.tptp.org`. Examples of tools and online interfaces for supporting the TPTP and TSTP libraries include `TPTP2X` [9, 8], `TPTP4X` [9, 8], `SystemOnTPTP` [10], and `SystemB4TPTP` [10]. `TPTP2X` provides underlying features for maintenance and export of the TPTP and TSTP libraries. Examples of such features are generating problems, collecting useful information and computing statistics for the headers of TPTP

problems, and pretty printing TPTP problems and TSTP solutions. The `TPTP2X` also provides features for preprocessing the input TPTP problems and post-processing the solutions output by ATP systems. It can expand TPTP problem `include` directives to the full version of the problem. It can convert TPTP format to the input formats of ATP systems. The `TPTP2X` is implemented in Prolog, which makes it easy to modify and extend. `TPTP4X` is an implementation of some of the `TPTP2X` features in C. It includes some added features such as advanced `include` directive processing in the TPTP format.

`SystemOnTPTP` is an online interface that allows the use of ATP systems and tools for solving TPTP problems. The interface allows a problem to be selected from the TPTP library, or provided by the user. Additionally, features for preprocessing of the input problem are provided using internal tools such as `TPTP2X` and `TPTP4X`. `SystemOnTPTP` allows multiple ATP systems to run on a problem. It also controls and limits the CPU time and memory usage of the selected ATP systems. Finally, it converts the output of an ATP system to the TPTP format.

The `SystemB4TPTP` is an online interface, which allows the use of the `TPTP2X` and `TPTP4X` tools for conversion between different formats of a certain logic compatible with a range of parsers, and translators for translating from one logic to another.

1.3 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 international

Conference on Automated Deduction (CADE) and *International Joint Conference on Automated Reasoning* (IJCAR) [11]. The CADE conference was started in 1974, and was held every two years in the beginning, and every year since 1996. The IJCAR conference was started in 2001, and was held every two years.

At each CADE and IJCAR conference, the *CADE ATP System Competition* (CASC) [12] is held to introduce and evaluate the performance of ATP systems. CASC evaluates the performance of ATP systems. The source of test problems for the competition is the TPTP library. To evaluate the ATP systems, 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. The ATP system used for each logic is the winner of the latest competition if more than one ATP system is available for the logic.

1.4 Research Goals

A problem expressed in a certain logical form can be either solved using an ATP system for the logic, or translated to a less expressive logic. If it is translated to a less expressive logic, again the same two options of solving using an ATP system of the less expressive logic, or translating down (if possible), are available. This continues until no further translation is possible. In Figure 1.1, the plain arrows indicate the process of translation, and the dashed arrows indicate the process of solving using an ATP system (See Chapter 5 for more about this). Each path from each logical form

to an ATP system is an alternative way of solving problems expressed in that logical form.

The ATP systems for less expressive logical forms have benefited from more years of development and testing than the ATP systems of more expressive logical forms. There is is a trade-off between the power of the ATP systems for a logical form and the expressivity of the logical form. Moreover, a problem translated to a less expressive logical form may have features that negatively affect the performance of the corresponding ATP systems. For each logical form, the effectiveness of solving problems through the possible paths starting from that logical form, as illustrated in Figure 1.1, is compared and discussed in this research. Introducing alternative ways of solving problems through paths that include translation to DL is a major concern of this research. To this end, a CNF to DL translator has developed, and a new TPTP syntax for DL has been designed.

1.5 Road Map

This dissertation is organized as follows. Chapter 1 contains an introduction to the research and related issues, to build up a basis for understanding the dissertation. Chapter 2 contains a survey of the logics related to the research. Chapter 3 describes a new syntax for Description Logic, and related libraries and tools. Chapter 4 describes a translation procedure from CNF to DL, and its implementation as a tool called `Saffron`. Additionally, this chapter includes a mathematical proof for the soundness of the translation procedure, and empirical tests of the soundness of the implemen-

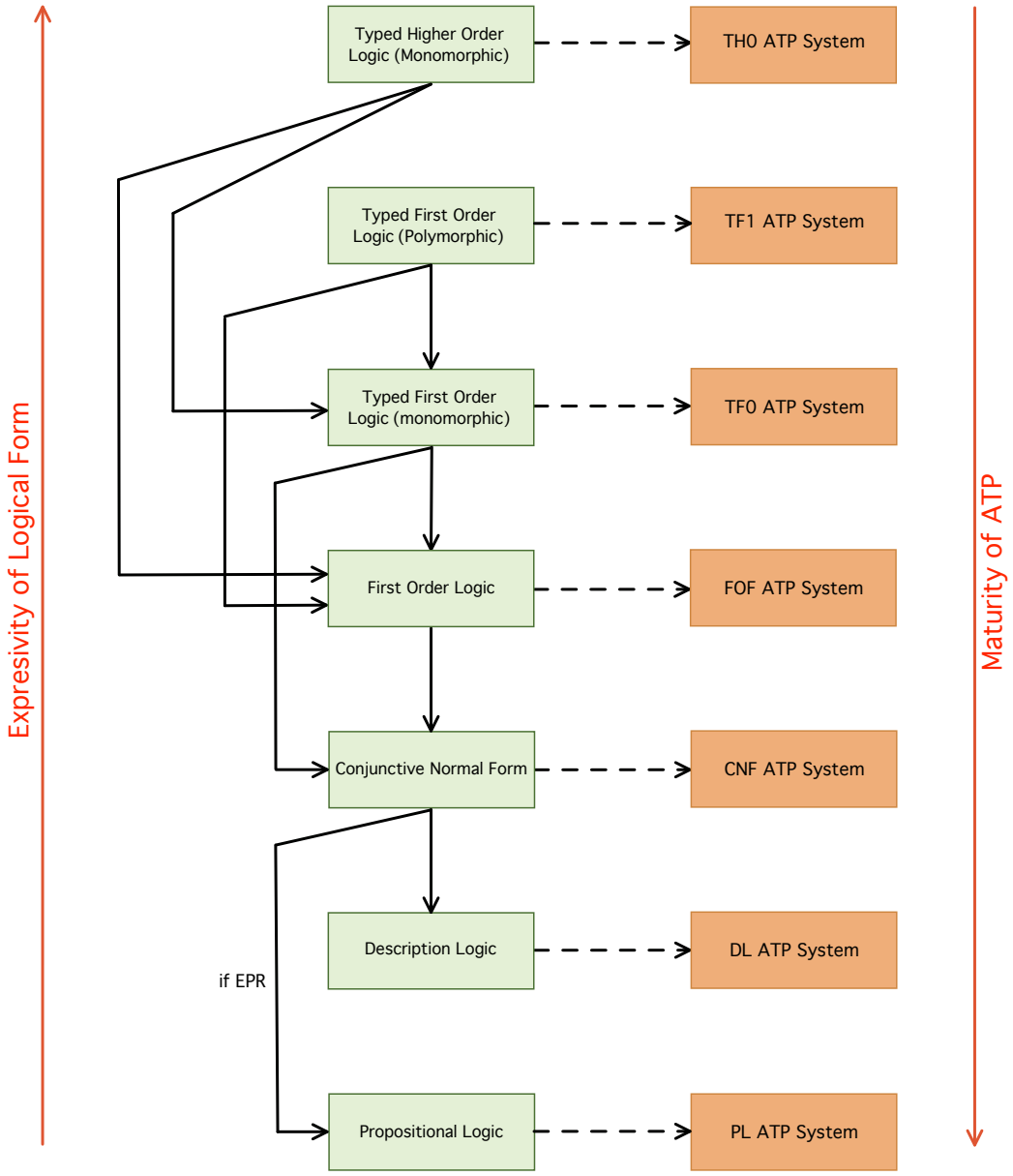

Figure 1.1: Possible Combinations of Translators and ATP Systems

tation of `Saffron`. Chapter 5 describes the experiment to evaluate different ways of solving problems in different logical forms. Additionally, this chapter includes analysis and discussion of the results of the experiment. Finally, Chapter 6 contains a summary of the whole dissertation, conclusions, and suggestions for future work.

Chapter 2

Survey of The TPTP Logics

In this chapter the logical forms Propositional Logic (PL), Description Logic (DL), First Order Form (FOF), Conjunctive Normal Form (CNF), Effectively Propositional form (EPR), Typed First order form-monomorphic (TF0), Typed First order form-polymorphic (TF1), and Typed Higher order form-monomorphic (TH0) are described.

2.1 Propositional Logic (PL)

Propositional Logic is a decidable logic. Examples of applications of PL include designing digital circuits [13], encoding constraint satisfaction problems [14], and reasoning about systems [15].

2.1.1 Syntax

Available syntaxes for PL include Dimacs [16] and the TPTP syntax. The syntax of PL consists of propositions and logical operators. Example 3 are propositions

in the TPTP syntax. By combining different propositions using logical operators, propositional formulae can be constructed.

Example 3

`coffee_helps_me_stay_awake`

`coffee_is_my_favorite_drink`

PL logical operators and some examples for each operator in both mathematical and the TPTP syntax are as follows:

- **Negation** is a unary prefix operator, and negates the proposition. The symbols ¬ and `~` are the mathematical and TPTP symbols for the negation operator respectively.

  Example 4

  $\neg coffee\_is\_my\_favorite\_drink$

  `~coffee_is_my_favorite_drink`

- **Conjunction** is a binary infix operator. It corresponds to English "and". The symbols ∧ and `&` are the mathematical and TPTP symbols for the conjunction operator respectively.

  Example 5

  $coffee\_helps\_me\_stay\_awake \wedge coffee\_is\_my\_favorite\_drink$

  `coffee_helps_me_stay_awake & coffee_is_my_favorite_drink`

- **Disjunction** is a binary infix operator. It corresponds to English "and/or". The symbols $\vee$ and `|` are the mathematical and TPTP symbols for the disjunction operator respectively.

  Example 6

  $coffee\_helps\_me\_stay\_awake \vee coffee\_is\_my\_favorite\_drink$

  ```
  coffee_helps_me_stay_awake | coffee_is_my_favorite_drink
  ```

- **Implication** is a binary infix operation. It corresponds to English "if-then". The symbols $\Rightarrow$ and `=>` are the mathematical and TPTP symbols for the implication operator respectively.

  Example 7

  $coffee\_helps\_me\_stay\_awake \Rightarrow coffee\_is\_my\_favorite\_drink$

  ```
  coffee_helps_me_stay_awake => coffee_is_my_favorite_drink
  ```

- **Equivalence** is a binary infix operation. It corresponds to English "if-and-only-if". The symbols $\Leftrightarrow$ and `<=>` are the mathematical and TPTP symbols for the equivalence operator respectively.

  Example 8

  $coffee\_helps\_me\_stay\_awake \Leftrightarrow coffee\_is\_my\_favorite\_drink$

  ```
  coffee_helps_me_stay_awake <=> coffee_is_my_favorite_drink
  ```

2.1.2 Semantics

A PL interpretation assigns either `TRUE` or `FALSE` to each proposition. The truth value of a formula is determined by 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. For example, Table 2.1 shows the truth tables of 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 PL are `MiniSat` [17] and `zChaff` [18]. In this research `MiniSat` is used for reasoning about problems expressed in PL. 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 [19].

2.2 Description Logic (DL)

Description Logic is a decidable logic, and more expressive than PL. Examples of applications of DL include knowledge representation [20], semantic web technologies [20], and expressing *ontologies* [21]. An ontology includes a categorization of elements of a domain of interest, definitions of the relationships between these categories, and knowledge about the domain. Example 9 illustrates a simple ontology.

Example 9

9.a. Coffee is a drink.

9.b. Negin is a person.

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

A *DL problem* is an ontology expressed in DL. The building blocks of a DL ontology are *individuals*, *classes*, and *roles*. Individuals are elements of the domain of interest. Classes are bags of individuals, and roles are relationship between two individuals. In Example 9, "coffee" and "Negin" are individuals, "drink" and "person" are classes, and "favorite drink" is a role.

There are different variations of DL. *ALC* is the basic DL, which includes the language constructs, such as classes, roles, individuals, class membership, role instances, conjunction of classes, disjunction of classes 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:

- The letter **S** represents all the ALC features.
- The letter **R** represents limited complex role inclusion axioms such as reflexivity, irreflexivity, and role disjointness.
- The letter **O** represents nominals (closed classes with a finite number of elements).
- The letter **I** represents inverse properties.
- The letter **Q** represents qualified cardinality restrictions.

SROIQ is the target DL in this research. `Saffron`, the CNF to DL translator presented in the Chapter 4 can produce a DL problem with SROI features.

2.2.1 Syntax

Available syntaxes for DL include the RDF/XML, OWL/XML, and Manchester OWL [22] syntaxes. A TPTP syntax for DL is proposed in this research, and explained in Chapter 3. The syntax of DL include individuals, classes, roles and DL operators. Example 10 is the ontology in Example 9, in the DL mathematical syntax.

Example 10

$coffee : drink$

$negin : person$

$favorite\_drink(negin, coffee)$

An ontology expressed in DL RDF/XML syntax is a set of RDF/XML tags. An RDF/XML tag might include sub-tags. Individuals, classes, and roles and their characteristics are expressed in `owl:NamedIndividual`, `owl:Class` and `owl:ObjectProperty` tags correspondingly. Example 11 is the ontology in Example 9, in the DL RDF/XML syntax.

Example 11

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

<owl:Class rdf:about="person"/>

<owl:ObjectProperty rdf:about="favorite_drink"/>

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

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

In a DL problem, every individual belongs to the default class *Thing*, and may belong to other classes. Since every member of a class is also a member of the default class *Thing*, every class is a subclass of the class *Thing*. The empty class in DL is the class *Nothing*. A class may be declared by its name or can be anonymous and constructed from its member individuals, or from operations on other classes, individuals, or roles. Let a and b be individuals, p and q be classes (named or anonymous), and r and s be roles, then Table 2.2 shows some possible constructions of a class in DL. The classes are in mathematical syntax.

A named class may be equal to, a subclass of, disjoint with, or the complement of a named class or an anonymous class. The definition of these relationships between classes are the same as the corresponding relationships between two sets.

Table 2.2: Complex DL Classes

Class	Description
p	p
$\neg p$	Complement of p
$p \sqcup q$	Union of p and q
$p \sqcap q$	Intersection of p and q
$\forall r.a$	Class of all individuals x where $r(x, a)$
$\forall r.p$	Class of all individuals x where if $r(x, y)$ then $p(y)$
$\exists r.p$	Class of all individuals x where there exists a y where $r(x, y)$ and $p(y)$

2.2.2 Semantics

A DL interpretation [23] consists of a non-empty set Δ^I, the domain of interpretation, and an interpretation function $.^I$ such that:

- $.^I : a \longmapsto a^I \in \Delta_M^I$,
- $.^I : p \longmapsto p^I \subseteq \Delta_M^I$, and
- $.^I : r \longmapsto r^I \subseteq \Delta_M^I \times \Delta_M^I$.
- $Thing^I = \Delta^I$
- $Nothing^I = \emptyset$
- $(\neg p)^I = \Delta^I \backslash p^I$
- $(p \sqcap q)^I = p^I \cap q^I$

- $(p \sqcup q)^I = p^I \cup q^I$
- $(\forall r.a)^I = \{x \in \Delta^I |\ (x, a^I) \in r^I\}$
- $(\forall r.p)^I = \{x \in \Delta^I |$ For every y in Δ^I, if $(x, y) \in r^I$ then $y \in p^I\}$
- $(\exists r.p)^I = \{x \in \Delta^I |$ There exists y in Δ^I such that $(x, y) \in r^I$ and $y \in p^I\}$
- $a^I \in p^I\ iff\ a : p$
- $(a^I, b^I) \in r^I\ iff\ r(a, b)$
- $a^I = b^I\ iff\ a{=}b$
- $r = s^-\ iff\ \forall x \in \Delta^I : \forall y \in \Delta^I : ((x, y) \in r^I \Leftrightarrow (y, x) \in (s^-)^I)$
- $r \sqsupseteq (s1 \circ s2)\ iff\ \forall x \in \Delta^I : \forall y \in \Delta^I : \forall z \in \Delta^I : (((x, y) \in s1^I \wedge (y, z) \in s2^I) \Rightarrow (x, z) \in r^I)$

2.2.3 ATP Systems

`Konclude` [24], `FaCT++` [25] and `HermiT` [26] are examples of ATP systems for DL. `Konclude` is the ATP system used in this research. `Konclude` is an ATP system for large and expressive ontologies. The supported ontology language is SROIQV(D). `Konclude` implements a sound and complete tableau calculus enhanced with sophisticated preprocessing methods and tableau saturation. Furthermore, the system can take advantage of multiple cores within a shared memory environment. `Konclude` won three out of nine benchmark categories at the OWL Reasoner Evaluation Com-

petition 2013 (ORE 2013) [27] and five out of six disciplines at the OWL Reasoner Evaluation Competition 2014 (ORE 2014) [28].

2.3 First Order Form (FOF), Conjunctive Normal Form (CNF) and Effectively Propositional Form (EPR)

2.3.1 First Order Form (FOF)

First Order Form is a semi-decidable logic, and more expressive than PL or DL. Some notions can be expressed in FOF, but not in PL and DL. Examples 12.a and 12.b can be expressed in FOF, but not in PL or DL.

Example 12

12.a. Coffee is some person's favorite drink.

12.b. Coffee helps everybody stay awake.

2.3.1.1 Syntax

Available syntaxes for FOF include the TPTP syntax. Example 13 and 14 are the Example 12 in the mathematical and TPTP syntaxes.

Example 13

13.a. $\exists X : \ is(coffee, favorite\_drink(X))$

13.b. $\forall Y : \ helps\_stay\_awake(coffee, Y)$

Example 14

14.a. `?[X] is(coffee,favorite_drink(X))`

14.b. `![Y] helps_stay_awake(coffee,Y)`

The syntax of FOF consists of three sets of symbols, variables, functions, and predicates. In Example 13 `X` and `Y` are variables. The symbols `coffee` and `favorite_drink` are functions. The symbols `is` and `helps_stay_awake` are predicates. The number of arguments of a function or a predicate is called the *arity*. Functions and predicates can have any arity greater than or equal to zero. Functions of arity zero are called *constants*. Functions of arity one and two are called *unary* and *binary* functions. Predicates of arity zero are called propositions, and are the same as propositions in PL. Predicates of arity one and two are called *unary* and *binary* predicates. In Example 13 `coffee` is a constant, and `favorite_drink` is a unary function, and `is` and `helps_stay_awake` are both binary predicates. *Terms* are built from functions and variables, and *atoms* are built from 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 PL, with two extra quantifiers, the Existential Quantifier $\exists$ and the Universal Quantifier

$\forall$. For example, $\exists$ is the existential quantifier in Example 13.a, which quantifies over the variable `X`, and $\forall$ is the universal quantifier in Example 13.b, which quantifies over the variable `Y`. The interpretation of the quantifiers are explained in the next section. The TPTP symbols for $\exists$ and $\forall$ are `?` and `!` respectively.

2.3.1.2 Semantics

A FOF interpretation consists of a *domain* D, a *function map* F, and a *predicate map* R. The domain of an interpretation is a 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. An interpretation maps ground terms to domain elements and closed formulae to {`TRUE, FALSE`}. The interpretation of logical operators of FOF, other than quantifiers, are the same as for PL. The interpretations of the existential quantifier and the universal quantifier are as follows.

- *Existential Quantifier* $\exists$. If an existential quantifier quantifies over a variable in a formula, then the formula is `TRUE` if and only if setting the variable to at least one element of the domain of the interpretation makes the formula `TRUE`.

- *Universal Quantifier* $\forall$. If an existential quantifier quantifies over a variable in a formula, then the formula is `TRUE` if and only if setting the variable to all the elements of the domain of the interpretation makes the formula `TRUE`.

Herbrand Interpretation

The *Herbrand Universe* of a FOF problem[1] 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 is the identity function, and the predicate map is a subset of the Herbrand Base that maps to `TRUE`.

2.3.1.3 ATP Systems

Many FOF ATP systems are available. Examples of FOF ATP systems are `Vampire` [29], `SPASS` [30], and `E` [6].

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-J7 [31] FOF and LTB divisions in 2014. 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.

[1] Technically, the Herbrand Universe is of the FOF language that has been used to write the problem.

2.3.1.4 Translators

FOF problems can be translated to less expressive logical forms. There is a sound procedure to translate FOF problems to CNF. There are some translators available, such as `ECNF` and `VCNF`. In this research, `ECNF` is used. `ECNF` is a subsystem of the `E` ATP system. Some FOF problems can be translated to DL, but there is no translator available for this purpose yet.

2.3.2 Conjunctive Normal Form(CNF)

2.3.2.1 Syntax

A CNF problem is of the form of PbC logic problems. The axioms and the negated conjecture of the CNF problem have a special structure, and are called *clauses*. A clause is a disjunction of *literals*. A literal is either an atom or a negated atom. An atom in CNF is the same as an atom in FOF. In a CNF problem, all variables are implicitly universally quantified. An empty clause has no literals. A CNF problem is either represented as a set of clauses, or a conjunction of clauses. Examples 15 and 16 are the two representations of one CNF problem with three clauses. In this dissertation, CNF problems are assumed to be a set of clauses.

Example 15

{ (a(X,Y) ∨ b ∨ c(T,S)),

(¬a(X,V) ∨ b),

(d(R) ∨ ¬b ∨ c(V,S) ∨ e(X,V,T)) }

Example 16

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

2.3.2.2 Semantics

A CNF interpretation is the same as a a special case of FOF interpretation. 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 [32].

2.3.2.3 ATP Systems

Most FOF ATP systems can also be used for CNF. All the FOF ATP systems, mentioned in Section 2.3.1.3 are also CNF ATP systems.

2.3.2.4 Translators

The translation of CNF problems to DL is the topic of Chapter 4. In this research a translator for this purpose is developed, and is described in Chapter 4.

2.3.3 Effectively Propositional Form (EPR)

EPR is a decidable fragment of CNF. EPR problems are CNF problems that are known to be reducible to PL problems. A problem with no functions of arity greater than zero has a finite Herbrand Universe, and is EPR. In addition, if a problem has a function of arity greater than zero, but does not have any variables or equality, then

it is an EPR problem. It can be tested if a problem has no functions of arity greater than zero. Additionally, in a problem with functions of arity greater than zero, it can be checked if there are any variables or equality.

2.3.3.1 Syntax

The EPR syntax is the same as the CNF syntax.

2.3.3.2 Semantics

An EPR interpretation is the same the CNF interpretation. The herbrand interpretation is used for EPR problems with finite herbrand universe.

2.3.3.3 ATP Systems

All CNF ATP systems can be used for solving EPR problems. In this research `iProver` [33], which is an ATP system for EPR, CNF, and FOF, is used for solving EPR problems. `iProver` is the winner of the EPR division at all CASCs from 2008 to 2014 [31].

2.3.3.4 Translators

Since EPR is a fragment of CNF, translation from EPR to DL is possible and is explained in Chapter 4. All EPR problems can be translated to PL since EPR problems and the set of all of their ground instances are equisatisfiable [34]. In this research, `EGround` [34] is used to translate EPR problems to PL. 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 (TF0)

Typed First order form - monomorphic [35] is the monomorphically typed first-order logic, with predefined and user-defined types. It supports ad-hoc polymorphism for only equality and some defined symbols. For example, arithmetic function and predicate symbols are ad-hoc polymorphic over the types such as integer type and rational number type. In this research, the arithmetic capabilities are refused.

2.4.1 Syntax

The syntax described here is the syntax used in the TPTP. The defined types are `$i` for individuals, `$o` for booleans, and types such as `$int` for integers and `$rat` for rational numbers. User-defined types include atomic types, function types and predicate types. Atomic types are of type `$tType`. In the TPTP syntax, user-defined types are defined in type formulae. Example 17 can be expressed in TF0.

In Example 17, `person`, `beverage`, and `syrup` are user-defined atomic types. These atomic types can be defined in type formulae, as in Example 18.

Example 17

17.a. Negin is a person.

17.b. Coffee is a beverage.

17.c. Vanilla syrup is a syrup.

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

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

Example 18

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

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

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

Constants are by default of type `$i`, unless they are defined to be of user-defined atomic types. In Example 17, "Negin", "coffee" and "vanilla syrup" are constants of types "person", "beverage" and "syrup" correspondingly. Statements 17.a, 17.b, and 17.c are expressed in TF0, as in Example 19.a, 19.b, and 19.c.

Example 19

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

19.b. `tff(coffee_type, type, coffee: beverage).`

19.c. `tff(vanilla_syrup_type, type, vanilla_syrup: syrup).`

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

A function type for a function of arity greater than zero is a defined as a mapping from the types of its arguments to its return type. If a function is not defined, the default type of its arguments and its return type is `$i`. In Example 17, "mixture" is a binary function. The function type `mixture` is defined in TF0 as Example 19.d.

Propositions are of type `$o`. A predicate type for a predicate of arity greater than zero is defined as a mapping from the types of its arguments to type `$o`. If a predicate type is not defined, the default type of its arguments is `$i`. In Example 17, "help stay awake" is a binary predicate. The predicate type `help_stay_awake` is defined in TF0, as in Example 20.

Example 20

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

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`. In Example 17, statement 17.e is expressed in TF0, as in Example 21. The types of variables `S` and `P` are `syrup` and `person`.

Example 21

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

2.4.2 Semantics

A TF0 interpretation is a generalization of a FOF interpretation. Types are interpreted by non-empty, pairwise disjoint *sub-domains*. The domain of interpretation D is the union of the sub-domains. A variable V of type t is evaluated to the elements of D_t where D_t is the sub-domain for type t. For example, for the predicate `p : (t1 * t2 * ...* tn) > $o`, and all possible n-tuples `(a1,a2,...,an)`, the predicate map maps `p(a1,a2,...,an)` to `TRUE` and `FALSE` where `a1` $\in D_i$, D_i is the sub-domain of type `ti`, $1 \leq$ `i` $\leq n$.

2.4.3 ATP Systems

Available ATP systems for TF0 include `CVC4` [4], `Princess` [36] and `SNARK` [37]. `CVC4` is the ATP system used in this research. `CVC4` is an open-source ATP system for proving the satisfiability of a version of FOF problems and TFA problems. `CVC4` can be used as a stand-alone tool or as a library, with essentially no limit on its use for research or commercial purposes. An interactive text-based interface and a rich C++ API for embedding in other systems are some of the features of this ATP system. `CVC4` is a project of New York University and University of Iowa. `CVC4` is the winner of the CASC-J7 [31] TFA division in 2014.

2.4.4 Translators

TF0 problems can be translated to FOF and CNF. There is a countersatisfiability preserving translation procedure from TF0 to FOF, and a countersatisfiability-

satisfiability preserving translation procedure from TF0 to CNF. `Monotonox2FOF` and `Monotonox2CNF` are variations of `Monotonox` [7], which is a translator of TF0 and FOF. `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. `Monotonox` applies the combination of these three approaches to reduce the complexity of the constructed formulae, while preserving the satisfiablity.

2.5 Typed First order form - polymorphic (TF1)

Typed First-order Form - polymorphic [38] is distinguished from TF0 by allowing polymorphism over the types of the arguments and the return values of functions and over the types of the arguments of predicates.

2.5.1 Syntax

The syntax described here is the syntax used in the TPTP. Similar to TF0, the defined types are `$i` for constants and `$o` for booleans. Monomorphic user-defined types include atomic types, function types and predicate types that are the same as user-defined types TF0. Polymorphic function and predicate types are expressed using *type signatures*.

In Example 22, the type definition `a_polymorphic_function_type` defines the polymorphic function `polymorphic_function` of arity $(m + n) > 0$, where `type_11`, `type_12`, ..., and `type_1m` are atomic user-defined types. The `!>[T1: $tType, T2: $tType, ..., Tn: $tType, RT: $tType]` is the type signature, and expresses that the last n arguments and the return value of this function are polymorphic. In the axiom `a_polymorphic_function_application`, `type_21`, `type_22`, ..., `type_2n`, and `ret_type` are user-defined atomic types; `term_11`, `term_12`, ..., `term_1m` are terms of types `type_11`, `type_12`, ..., `type_1m`; `term_21`, `term_22`,..., `term_2n` are terms of types `type_21`, `type_22`, ..., and `type_2n`; and `ret_value` is a term of type `ret_type`.

Example 22

```
tff(a_polymorphic_function_type,type,(
    polymorphic_function:
     !>[T1: $tType, T2: $tType, ..., Tn: $tType, RT: $tType]:
      ((type_11 * type_12 * ... * type_1m * T1 * T2 * ... * Tn) > RT))).

tff(a_polymorphic_function_application,axiom,(
    ret_value = polymorphic_function(type_21,type_22, ...,type_2n,ret_type,
                                     term_11,term_12,...,term_1m,
                                     term_21,term_22,...,term_2n))).
```

Example 23 can be expressed in TF1 as Example 24. In Example 24, `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 `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 polymor-

phic function `mixture` is used. In `mixture(beverage,coffee,S)`, the first argument, `beverage`, expresses the type of the polymorphic type of this function.

Example 23

Axioms = { Coffee is a beverage,
Vanilla syrup is a syrup,
Caramel syrup is a syrup,
The mixture of a beverage and a syrup 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 24

```
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

A TF1 interpretation includes of a non-empty collection D of non-empty sets, the *domains*. The union of all domains is called the universe, U. Interpretation of the monomorphic terms and atoms is the same to TF0 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 variable is considered. For example, the predicate

```
p :
     !>[T1:$tType, T2:$tType,..., Tn:$tType]:
      ( t1 * t2 * ...* tm, T1 * T2 * ...* Tn) > $o
```

is a polymorphic predicate with $(m + n) > 0$ arguments, such that the last n arguments have polymorphic types. The predicate map of the predicate `p` maps `p(a1,a2,..,am,b1,b2,..,bn)` to `TRUE` ad `FALSE` for all possible $(m + n)$-tuples `(a1,a2,..,am,b1,b2,..,bn)` where `ai` $\in D_i$, D_i is the domain of `ti`, $1 \leq$ `i` $\leq m$, `bj` $\in D_j$, D_j is any possible domain for `bj`, and $1 \leq$ `j` $\leq n$.

2.5.3 ATP Systems

The only ATP system for TF1 is `Alt-Ergo` [39], which is used in this research. `Alt-Ergo` is an open-source ATP system, written in the Caml language. It pro-

vides full supports for the proof of mathematical theories expressed in TF1 such as polymorphic functional arrays and enumerated types theories. `Alt-Ergo` is highly parametrized and modular.

2.5.4 Translators

TF1 problems can be translated to TF0 and FOF. There are countersatisfiability preserving translation procedures from TF1 to TF0 and FOF. `Why3_TFF` [40] and `Why3_FOF` [40] are the only available translators for translating TF1 to TF0 and FOF, and are used in this research.

2.6 Typed Higher order form-monomorphic (TH0)

The Typed Higher order form - monomorphic [41] is distinguished from TF0 by addition of λ-terms, quantification over functions, and minor modifications of the syntax. The λ-calculus is used for function definition (function abstraction is also a term used in many papers).

2.6.1 Syntax

The syntax described here is the syntax used in the TPTP. Similar to TF0, the defined types are `$i` for constants, `$o` for booleans. User-defined types include atomic types and function types. Atomic types are of type `$tType`. Function types are of the form `a` or `b > c`. Type of `a` is either a defined or an atomic type. Types of `b` and `c` are either a defined type, an atomic type or a function types of the form `b > c`.

Predicates in TH0 are functions whose return type is type `$o`. The types of variables are function types. The axioms and the conjecture in Example 25 can be expressed in TH0 as in Example 26. The atomic types are `beverage`, and `syrup`, and the function types are `coffee`, `hot`, `heat`, `mix`, `hot_mixture`, and `cold_mixture`.

In TH0 TPTP syntax, the symbol `@` is used for function application. In Example 26 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`).

In TH0 TPTP syntax, the symbol `^` is used for function definition. The symbol `^` is the TPTP symbol for λ. In a function definition, the symbol `^` quantifies over a local variable which is used in the body of the function definition. In Example 26, the axiom `cold_mixture_definition`, variables `B` and `S` of types `beverage` and `syrup` are local variables that are used in the definition of the function `cold_mixture`.

Example 25

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 26

```
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

A TH0 interpretation is a generalization of a TF0 interpretation. In a TH0 interpretation, quantification over functions is allowed, so a variable of a function type of arity n corresponds to any n-tuple of the domain of an interpretation mapped to the domain of interpretation. There are two kinds of interpretations commonly employed for TH0. *Full semantics* [42] 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 TF0 interpretation. *Henkin semantics* [43], which is essentially semantics for typed higher-order with finite types, is 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.

2.6.3 ATP Systems

Available ATP systems for TH0 include `Satallax` [5], `Isabelle-HOT` [44], and `LEO-II` [45]. `Satallax` is the ATP system used in this research. `Satallax` is an ATP system, written in the Objective Caml language. It provides supports for the problems expresses in higher-order logical froms, particularly Church's simple type theory with extensionality and choice operators. `Satallax` was the winner of CAC-J7 [31] THF devision.

2.6.4 Translators

TH0 problems can be translated to TF0 and FOF. There are countersatisfiability preserving procedures for translating a TH0 problems to TF0, and similarly to FOF. `Isabelle-2TF0` and `Isabelle-2FOF` [44] are the only available stand-alone translators for translating TH0 to TF0 and FOF, and are used in this research. `Isabelle-2FOF` can also be used for translating TF0 to FOF.

Chapter 3

New Description Logic Syntax

Description Logic Form (DLF) is the new TPTP syntax for the DL SROIQ. The design of DLF is a fundamental step toward adding DL to the TPTP world, so the DL community will be able to benefit from the TPTP and TSTP tools. DLF, which is compact, natural and easily processable by both human and computers, has been designed in this research. The BNF enables the construction of parsers for DLF problems and solutions in other languages other than Prolog. Appendix A contains the BNF for DLF.

3.1 Description Logic Form (DLF)

Like all TPTP problem files, DLF problem files include up to three sections: header, include directives and the

annotated formulae. The header and include directives are the same as TPTP problem or axiom files in logical forms other than DLF. The annotated formulae of a DL

problem can include *definitions*, *type formulae*, and *logic formulae* that are explained in the Section 3.1.1.

3.1.1 DLF Identifiers

Like the TPTP syntax for other logics, DLF uses only standard ASCII characters. In the TPTP syntax, an *atomic word* starts with a lower_case character and contains alphanumeric and underscore, or is any sequence of characters wrapped in a pair of 'single quotes'. A *defined word* is the character `$` followed by an atomic word.

One major application of DL is semantic web technologies. DLF needs to be compatible with existing DL syntaxes, specially exchange syntaxes such as OWL/XML and RDF/XML, and make it possible to import ontologies from the semantic web. DLF allows the definition of *namespaces* and *schemas*, such as XML namespace and RDF schema. A *reference word* is the character `&` followed by an atomic word. The *delimiters* are `#`, `;` and `/`. They are used between two identifiers, of which the first one is the name of a namespace, an schema, or an ontology. The use of reference words and delimiters is explained in Section 3.1.2.

The syntax for the name of an individual, a class, or a role is an atomic word, a reference word, two atomic words concatenated by a delimiter, or a reference word followed by a delimiter and an atomic word. A *class term* is a named class or an anonymous class. A class term is either *unitary class term* or a *binary class term*. Constructions of unitary class terms and binary class terms are shown in Table 3.1 and Table 3.2 respectively, where `a` is an individual and `r` is a role.

Table 3.1: Construction of a Unitary Class Term in DLF

Unitary Class Term	Meaning
`&thing`	*Thing*
`&nothing`	*Nothing*
`class_name`	The name of a class
`(class_term)`	A unitary or binary class term `class_term` wrapped in a pair of parentheses
`[a_1, a_2, ..., a_n]`	$\{a\_1, a\_2, \ldots, a\_n\}$
`-unitary_class_term`	$\neg unitary\_class\_term$
`! role(_,a)`	$\forall r.a$
`! role(_,class_term)`	$\forall r.class\_term$
`? role(_,class_term)`	$\forall r.class\_term$

Table 3.2: Construction of a Binary Class Term in DLF

Binary Class Term	Meaning
`unitary_class_term`	*unitary_class_term*
`++ unitary_class_term`	$\sqcup$ *unitary_class_term*
`unitary_class_term`	*unitary_class_term*
`** unitary_class_term`	$\sqcap$ *unitary_class_term*
`unitary_class_term`	*unitary_class_term*
`<+> unitary_class_term`	$\uplus$ *unitary_class_term*

All class binary operators are left associative.

3.1.2 DLF Definitions

Definitions include setting aliases for repeatedly-used identifiers, such as namespaces and schemas. A definition formula has the form: `dlf(`*name*`, definition,` *dlf_definition_formula*`,` *annotations*`)`. The form of a *dlf_definition_formula* is *alias* `:=` *identifier*, where the operator `:=` is definition operator, the *alias* is an atomic word, and the *identifier* is an atomic word, a defined word, a reference word, a reference word followed by a delimiter, or a reference word followed by a delimiter and an atomic word.

XML and OWL namespaces, and XML and RDF, schemas are examples of common namespaces and schemas defined in a DL ontology. A set of definitions, including some commonly used definitions, is shown in Example 27. The definitions in Example 27 are saved in an TPTP axiom file `SYN001~0.ax` that can be included in DL problems where they are needed.

Example 27

```
%----------------------------------------------------------------
%----Library header definitions
dlf(owl_defn,definition,
    owl := 'http://www.w3.org/2002/07/owl' ).

dlf(rdf_defn,definition,
    rdf := 'http://www.w3.org/1999/02/22-rdf-syntax-ns' ).

dlf(xml_defn,definition,
    xml := 'http://www.w3.org/XML/1998/namespace' ).

dlf(xsd_defn,definition,
    xsd := 'http://www.w3.org/2001/XMLSchema' ).

dlf(rdfs_defn,definition,
    rdfs := 'http://www.w3.org/2000/01/rdf-schema' ).

dlf(class_defn,definition,
    class := $tType ).

dlf(thing_defn,definition,
    thing := &owl#'Thing' ).

dlf(nothing_defn,definition,
    nothing := &owl#'Nothing' ).
%----------------------------------------------------------------
```

To reference a syntactically defined identifier, a reference word is used. At the time of preprocessing of a problem by an ATP system, all reference words are replaced by the corresponding identifiers. In Example 27 the reference word `&owl` in the last two formulae is replaced with `'http://www.w3.org/2002/07/owl'`. Similarly, later in a problem that includes this axiom file `SYN001~0.ax`, the reference words `&thing`

and `&nothing` can be used instead of the long identifiers, `'http://www.w3.org/2002/07/owl#Thing'` and `'http://www.w3.org/2002/07/owl#Nothing'`.

To specify the namespace or schema of an identifier, a delimiter and the identifier are appended to the end of the related namespace or schema. The reference word for the namespace or schema can also be used instead of the full name of the namespace or schema.

It is common to define the ontology at the beginning of a DL ontology. The keyword `$ontology` is reservered as an alias for the ontology identifier. The keyword `$base` keyword is reserved for the base namespace. The identifier defined as `$base` is replaced whenever the namespace for an identifier is not mentioned. It is usually set to the ontology name followed by a delimiter as the default delimiter. Example 28 includes the use of the `$ontology` and the `$base` keywords.

3.1.3 DLF Type Formulae

The general form of a type formula is `dlf(`*name*`, type,` *dlf_type_formula, annotations*`)`. The building blocks of a DL ontology, individuals, classes, and roles are declared in type formulae,

individual: *class_term*,
class_name: `$tType`,
role: `(` *unitary_class_term* `*` *unitary_class_term* `) > $o`.

DLF type formulae are similar to some TF0 type formulae. The defined words $tType and `$o` are borrowed from TF0, which are the types for atomic user-defined types and booleans respectively. The syntax for a role type declaration is similar to

the syntax for declaring a binary predicate in TF0. In DLF the left hand side of the `*` is the domain and the right hand side of the `*` is the range of the role. The domain and range of a role are unitary class terms.

3.1.4 DLF Logic Formulae

The general form of a DLF logic formula is `dlf(`*name*`,` *logic_formula_role, dlf_logic_formula*`,` *annotations*`)`. Examples of *logic_formula_role* are `axiom` and `conjecture`. Table 3.3 shows different forms of a *dlf_logic_formula*.

Table 3.3: DLF Logic Formulae

DLF logic formula	Meaning
`class_name = class_term`	$class\_name = class\_term$
`class_name != class_term`	$class\_name \neq class\_term$
`class_name <> class_term`	$class\_name \sqcap class\_term = Nothing$
`class_name << class_term`	$class\_name \sqsubseteq class\_term$
`role(a_1,a_2)`	$r(a\_1, a\_2)$
`~role(a_1,a_2)`	$\neg r(a\_1, a\_2)$
`$reflexive(role)`	`role` is a reflexive role
`$irreflexive(role)`	`role` is an irreflexive role
`$symmetric(role)`	`role` is a symmetric role
`$asymmetric(role)`	`role` is an asymmetric role
`$transitive(role)`	`role` is a transitive role
`$functional(role)`	`role` is a well-defined function
`role_1 -= role_2`	$role\_1 = role\_2^{-}$
`role_1 << role_2`	$role\_1 \sqsubseteq role\_2$
`role_1 >> role_2`	$role\_1 \sqsupseteq role\_2$
`role_1 >> role_2 @ role_3 @ ...@ role_n` where $n \geq 3$	$role\_1 \sqsupseteq role\_2 \circ role\_3 \circ ... \circ role\_n$

Examples 28 and 29 express the ontology in Example 9 of Chapter 2 in DLF and RDF/XML syntaxes.

Example 28

```
%------------------------------------------------------------------------
include('SYN001^0.ax').
%------------------------------------------------------------------------
%---Rest of the Header
dlf(ontology_defn,definition,
    $ontology := 'http://www.tptp.org/ontologies/CoffeeOntology' ).


dlf(base_defn,definition,
    $base := &$ontology# ).
%------------------------------------------------------------------------
dlf(person_type,type,(
    person:  $tType )).

dlf(drink_type,type,(
    drink:  $tType )).

dlf(favoriteDrink_type,type,(
    favoriteDrink:  ( drink * person ):  $o )).

dlf(coffee_type,type,(
    coffee:  drink )).

dlf(negin_type,type,(
    negin:  person )).

dlf(coffee_is_negins_favorite_drink,axiom,(
    favoriteDrink(coffee,negin) )).
%------------------------------------------------------------------------
```

Example 29

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
]>

<rdf:RDF xmlns="http://www.tptp.org/ontologies/CoffeeOntology#"
  xml:base="http://www.tptp.org/ontologies/CoffeeOntology"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <owl:Ontology rdf:about="http://www.tptp.org/ontologies/CoffeeOntology"/>

  <owl:ObjectProperty
    rdf:about="http://www.tptp.org/ontologies/CoffeeOntology#favoriteDrink">
     <rdfs:domain
      rdf:resource="http://www.tptp.org/ontologies/CoffeeOntology#drink"/>
     <rdfs:range
      rdf:resource="http://www.tptp.org/ontologies/CoffeeOntology#person"/>
  </owl:ObjectProperty>

  <owl:Class
      rdf:about="http://www.tptp.org/ontologies/CoffeeOntology#drink"/>

  <owl:Class
      rdf:about="http://www.tptp.org/ontologies/CoffeeOntology#person"/>

  <owl:NamedIndividual
    rdf:about="http://www.tptp.org/ontologies/CoffeeOntology#coffee">
       <rdf:type
      rdf:resource="http://www.tptp.org/ontologies/CoffeeOntology#drink"/>
  </owl:NamedIndividual>

  <owl:NamedIndividual
    rdf:about="http://www.tptp.org/ontologies/CoffeeOntology#negin">
       <rdf:type
      rdf:resource="http://www.tptp.org/ontologies/CoffeeOntology#person"/>
  </owl:NamedIndividual>
</rdf:RDF>
```

Chapter 4

`Saffron`, a CNF to DL Translator

As explained in Chapter 1, a major contribution of this research is to introduce alternative ways of solving problems by translation to DL. Problems in logics more expressive than CNF can be translated directly to CNF, or indirectly by translation via intermediate logics. A CNF problem may be translated to DL. No CNF to DL translator was available when this research was started. A satisfiability preserving translation procedure from CNF to DL, and its implementation as `Saffron`, are provided as a part of this research. `Saffron` is implemented in Prolog. The input is a CNF problem in TPTP syntax, and the output is a DL problem in either RDF/XML and DLF.

4.1 Input CNF Problems for `Saffron`

The CNF to DL translation is based on translating each CNF clause to a set of DL formulae that has equivalent semantics. Constants, unary predicates, and binary

predicates in CNF correspond to individuals, classes, and roles that are the building blocks of DL ontologies. Clauses with only constants, unary predicates, and binary predicates might be translated directly to DL. It might not be possible to translate some such clauses to DL, because there might not be equivalent DL semantics for the clauses. The current version of `Saffron` supports translation of many forms of clauses that have equivalent semantics in DL.

Definition 4.1 CNF clauses that can be translated to DL using `Saffron` are referred to as *DL-able clauses*. Since DL-able clauses have no functions of arity greater than zero, they are EPR.

Definition 4.2 CNF problems that includes only DL-able clauses are referred to as *DL-able problems*. Since these problems have no functions of arity greater than zero, they are EPR (their Herbrand Universe is finite).

A DL-able problem can be fully translated to DL, i.e, 100% of its clauses can be translated to DL. A problem with one or more CNF clauses that are not DL-able can be partially translated to DL, i.e, less than 100% of its clauses can be translated to DL. The unsatisfiability of a partially translated problem might be preserved, because an unsatisfiable subset of the clauses has been translated. The experiment in Section 4.5 shows how successful this translator is in preserving the unsatisfiability of partially translated problems.

4.1.1 Preparing CNF Problems for Translation

Propositions, predicates of arity greater than two, and functions of arity greater than zero cannot be translated directly to DL because there is no DL syntax for such predicates and functions. A *splitting technique* is applied in the released versions of `Saffron` to preprocess problems with propositions, and remove propositions. This technique is explained in Section 4.1.1.1. However, there is no known approach for replacing a predicate of arity greater than two with only propositions, unary predicates and/or binary predicates. Known approaches for replacing functions of arity greater than zero with predicates can be used to transform CNF problems with unary functions to problems with only unary predicates, binary predicates, or both. The transformation is explained in Section 4.1.1.2, and it could be used later to extend `Saffron`.

4.1.1.1 The Splitting Technique

Many CNF problems contain propositions. Propositions do not have a corresponding notion in DL. In order to solve these problems by translation to DL, splitting has been used. Considering each proposition in turn, the proposition is assigned `TRUE` and then `FALSE`. In each case the resultant clause set is simplified by removing clauses that contain `TRUE` or `~FALSE`, and removing `FALSE` and `~TRUE` literals from the other clauses. Once all propositions have been removed, the resultant clause sets are translated to DL. If there are n propositions, there are 2^n sets to translate. If any one of the translated sets is satisfiable, the original CNF is satisfiable. If all of

the translated sets are unsatisfiable, the original CNF is unsatisfiable. (Many readers will recognize this approach from the DPLL algorithm [46].)

4.1.1.2 Transforming Problems with Unary Functions to Problems with Unary Predicates, Binary Predicates, or Both

A function replacement approach for CNF problems is presented in [47]. This approach was considered for EPR problems with functions, but no variables or equality to replace the unary functions with binary predicates. In this approach a function of arity n ($n > 0$) is replaced with a predicate of arity $n + 1$. After replacing the functions with predicates, *axioms of totality* must be added. According to the axiom of totality for a function of arity n, the function should map every n-tuple of elements of the domain of an interpretation to some element of the domain. 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 a Skolem function. Since the existential quantifier quantifies over the return value of a function of arity n, the arity of the introduced Skolem function is also n. As the goal of this process is to remove all the functions of 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 30 shows the axioms of d-totality where `p` is a replacement for unary function `f`, 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 30

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

In addition to axioms of totality, the axioms of *well-definedness* must be added to complete the function replacement. According to the axiom of well-definedness of a function of arity n, the function maps every n-tuple of elements of the domain of an interpretation to at most one element of the domain. The axiom of well-definedness of a unary function replace with a binary predicate can be translated to DL. The translated DL axiom of well-definedness expresses that the role corresponding to the replaced binary predicate is functional. Example 31 and 32 show an axiom of well-definedness where `p_f` is the replacement of the unary function `f` in CNF and DLF correspondingly.

Example 31

```
~p_f(X,Y1) | ~p_f(X,Y2) | Y1=Y2
```

Example 32

```
$functional(p_f)
```

The Example 33 includes a unary function `f` that can be replaced by a binary predicate as in Example 34. In the axiom `predicate_applied_on_function` of Example 33, function `f` is replaced with the predicate `p_f` in the axiom `replacing_f_with_p` of Example 34. In Example 34, the axiom `the_3_totality_of_f` is the 3-totality axiom for function `f`, where `e1`, `e2`, and `e3` are the only elements of the Herbrand Universe. The axiom `well_definedness_of_f` is the well-definedness axiom for function `f`.

Example 33

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

Example 34

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

cnf(the_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 )).
```

4.2 The Translation Procedure

Definition 4.3 The translation function $saffron : c \longmapsto saffron(c)$ translates a CNF clause c in a CNF problem to a set of DL formulae $saffron(c)$. If c is not DL-able, $saffron(c)$ is an empty set. For each constant, unary predicate, and binary predicate in c, an individual, a class, and a role are correspondingly defined in $saffron(c)$ with the same names.

Definition 4.4 The translation of a CNF problem $problem = \{c_1, c_2, ..., c_N\}$ is $saffron(problem) = \bigcup_{i=1}^{N} saffron(c_i)$. $saffron(problem) = saffron(problem\_dl)$ where $problem\_dl$ is the set of all DL-able clauses of the CNF problem $problem$. DL-able clauses are EPR clauses, so $problem\_dl$ is also an EPR problem.

Notation. The symbols a and b are constants in the CNF problem $problem\_dl$ or individuals in the DL problem $saffron(problem)$, p and q are unary predicates in $problem\_dl$, and classes in $saffron(problem)$. The symbols r, s, and u are binary predicates other than equality or inequality in $problem\_dl$, and roles in $saffron(problem)$.

The symbols `&thing` and `&nothing` are DLF referenced words for `'http://www.w3.org/2002/07/owl#Thing'` and `'http://www.w3.org/2002/07/owl#Nohing'`.

Notation. $+\!\!+_{i=1}^{n}$`p_i` $\equiv$ `p_1 ++ p_2 ++ ... ++ p_n` and $*\!*_{i=1}^{n}$`p_i` $\equiv$ `p_1 ** p_2 ** ... ** p_n` are used in this chapter, where $+\!\!+$ and $*$ are the union and intersection symbols of DLF. $|_{i=1}^{n} l_i \equiv l_1$ `|` l_2 `|` ... `|` l_n and $\&_{i=1}^{n} l_i \equiv l_1$ `&` l_2 `&` ... `&` l_n are used in this chapter, where l_i is a literal, and | and & are the logical disjunction and the logical conjunction symbols.

Every DL formula defines a characteristic of an individual, a class, or a role, or describes the default class *Thing*. The characteristics of individuals that are covered in this translation are an individual belonging to a named class or certain types of anonymous classes, an individual being in a binary relationship with another individual, and an individual not 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 formulae of equality between two individuals or inequality between two individuals use the "same individual as" and "different individual from" notations in RDF/XML, and `=` and `!=` in DLF. Table 4.1 illustrates CNF clauses c and their corresponding DL formulae $saffron(c)$, that describe characteristics of individuals. The DL formulae are in DLF syntax. For example, the CNF clause $|_{i=1}^{n}$`p_i(a)` `|` $|_{j=1}^{m}$`~q_j(a)`, where $n \geq 0$, $m \geq 0$, and $(m+n) \geq 2$ in the third row of the Table 4.1, is translated to the DL formula `a:` $+\!\!+_{i=1}^{n}$`p_i ++` $+\!\!+_{j=1}^{m}$`-q_j`. The CNF clause expresses that at least one of `p_i(a)` or `~q_j(a)` is `TRUE`. The DL formula

expresses that the individual `a` belongs to the union of the classes `p_i` and the complements of the classes `q_i`.

Table 4.1: CNF Clauses and Equivalent Individual DL Characteristics

Clause c	$saffron(c)$
`p(a)`	{ `a: p` }
`~p(a)`	{ `a: -p` }
$\vert_{i=1}^{n}$`p_i(a) |` $\vert_{j=1}^{m}$`~q_j(a)` where $n \geq 0$, $m \geq 0$, and $(m+n) \geq 2$	{`a:` $+\!\!+_{i=1}^{n}$`p_i ++` $+\!\!+_{j=1}^{m}$`-q_j`}
`r(a,b)`	{ `r(a,b)` }
`~r(a,b)`	{ `~r(a,b)` }
`a=b`	{ `a=b` }
`a!=b`	{ `a!=b` }
$\vert_{i=1}^{n}$`r(a,b_i)` where $n \geq 2$	{ `a: ?r(_,[b_1,b_2,...b_n])` }
$\vert_{i=1}^{n}$`(a = b_i)` where $n \geq 2$	{ `a: [b_1,b_2,...b_n]` }

The characteristics of classes that are covered in this translation are a class being a subclass of a named class or an anonymous class, and equivalent to the class $Thing$, or the class $Nothing$. Table 4.2 illustrates CNF clauses c and their corresponding DL formulae $saffron(c)$, that describe characteristics of classes. For example, the CNF clause `~p(X) | q(X)` in the second row of the Table 4.2 is translated to the DL formula `p << q`. The CNF clause is equivalent to the FOF formula `![X]: (p(X) => q(X))`, and expresses that for all `X` if `p(X)` then `q(X)`. The DL formula expresses that the class `p` is a subclass of the class `q`. In some cases, the subclass characteristic implicitly covers the equivalency characteristic because the equivalency between two classes is expressed in more than one CNF clauses. For example, the equivalency between two named classes is expressed by two CNF clauses, each of which expresses that each of the classes is a subclass of the other.

Table 4.2: CNF Clauses and Equivalent Class DL Characteristics

Clause c	$saffron(c)$
`p(X)`	{ `p = &thing` }
`~p(X) | q(X)`	{ `p << q` }
`~p(X) | r(X,Y)`	{ `p << ? r(_,&thing)` }
`p(X) | ~r(X,Y)`	{ `? r(_,&thing) << p` }
`p(X) | ~q(Y) | ~r(X,Y)`	{ `? r(_,q) << p` }
$\mid_{j1=1}^{m1}$`q_j1(Y)` $\mid$ $\mid_{j2=1}^{m2}$`~qp_j2(Y)` $\mid$ $\mid_{k=1}^{h}$`~r_k(X,Y)` where $(m1 + m2) \geq 1$, and $h \geq 1$	{(`**`$_{k=1}^{h}$`r_k(_,` `**`$_{j1=1}^{m1}$`(-q_j1)` `**` `**`$_{j2=1}^{m2}$`qp_j2)) << &nothing` }
$\mid_{i1=1}^{n1}$`p_i1(X)` $\mid$ $\mid_{i2=1}^{n2}$`~pp_i2(X)` $\mid$ $\mid_{j1=1}^{m1}$`q_j1(Y)` $\mid$ $\mid_{j2=1}^{m2}$`~qp_j2(Y)` $\mid$ $\mid_{k=1}^{h1}$`~r_k(X,Y)` where $(n1 + n2) \geq 1$, $(m1 + m2) \geq 1$, and $h \geq 1$	{ (`**`$_{k=1}^{h}$`r_k(_,` `**`$_{j1=1}^{m1}$`(-q_j1)` `**` `**`$_{j2=1}^{m2}$`qp_j2)) << ( ++$_{i1=1}^{n1}$`p_i1 ++` ++$_{i2=1}^{n2}$`(-pp_i2)) }
$\mid_{i1=1}^{n1}$`p_i1(X)` $\mid$ $\mid_{i2=1}^{n2}$`~pp_i2(X)` $\mid$ $\mid_{i3=1}^{n3}$`s_i3(X,a_i3)` $\mid$ $\mid_{i4=1}^{n4}$`~sp_i4(X,a'_i4)` $\mid$ $\mid_{j1=1}^{m1}$`q_j1(Y)` $\mid$ $\mid_{j2=1}^{m2}$`~qp_j2(Y)` $\mid$ $\mid_{j3=1}^{m3}$`u_j3(Y,b_j3)` $\mid$ $\mid_{j4=1}^{m4}$`~up_j4(Y,bp_j4)` $\mid$ $\mid_{k=1}^{h}$`~r_k(X,Y)` where $(n1 + n2 + n3 + n4) \geq 1$, $(m1 + m2 + m3 + m4) \geq 1$, and $h \geq 1$	{ `**`$_{k=1}^{h}$`?r_k(_,(` `**`$_{j1=1}^{m1}$`(-q_j1)` `**` `**`$_{j2=1}^{m2}$`qp_j2` `**` `**`$_{j3=1}^{m3}$`-(! u_j3(_,b_j3))` `**` `**`$_{j4=1}^{m4}$`! up_j4(_,bp_j4)) ) << (++$_{i1=1}^{n1}$`p_i1 ++` ++$_{i2=1}^{n2}$`(-pp_i2) ++` ++$_{i3=1}^{n3}$`!s_i3(_,a_i3) ++` ++$_{i4=1}^{n4}$`-(! sp_i4(_,a'_i4)))` }

The role characteristics that are covered in this translation are reflexivity, irreflexivity, symmetry, asymmetry, transitivity, a role being subrole of another role, a role being subrole of the inverse of another role, and a role being super-role of the chain of two roles. Table 4.3 illustrates CNF clauses c and their corresponding DL formulae $saffron(c)$, that describe characteristics of roles. For example, the CNF clause `~r(X,Y) | s(Y,X)` in the seventh row of the Table 4.3 is translated to the formulae {`inv_s -= s`, `r << inv_s`}. The CNF clause is equivalent to the FOF formula `![X,Y]:`

`(r(X,Y) => s(Y,X))`, and expresses that for all `X` and `Y` if `r(X,Y)` then `s(Y,X)`. The DL formula expresses that the role `r` is a subrole of the inverse of the role `s`.

Table 4.3: CNF Clauses and Equivalent Role DL Characteristics

Clause c	$saffron(c)$
`r(X,X)`	{ `$reflexive(r)` }
`~r(X,X)`	{ `$irreflexive(r)` }
`~r(X,Y) | r(Y,X)`	{ `$symmetric(r)` }
`~r(X,Y) | ~r(Y,X)`	{ `$asymmetric(r)` }
`~r(X,Y) | ~r(Y,Z) | r(X,Z)`	{ `$transitive(r)` }
`~r(X,Y) | s(X,Y)`	{ `r << s` }
`~r(X,Y) | s(Y,X)`	{ `inv_s -= s,` `r << inv_s` }
`~r_1(X,Y) | ~r_2(Y,Z) | s(X,Z)`	{ `s >> (r_1 @ r_2)` }
`~r_1(X,Y) | ~r_2(X,Z) | s(Y,Z)`	{ `inv_s -= s,` `inv_r_2 -= r_2,` `inv_s >> (inv_r_2 @ r_1)` }

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`. The characteristics of the class `&thing` that are covered in this translation are the class `&thing` being equivalent to certain kinds of anonymous classes. Table 4.4 illustrates CNF clauses and their corresponding DL formulae that describe the characteristics of the class `&thing`. For example, the CNF clause $|_{i=1}^{n}$`(X = a_i)` in the third row of the Table 4.4 is translated to the DL formula `&thing = [a_1,a_2,... ,a_n]`. The CNF clause expresses that for all `X`, `X` is equal to at least one of the constants `a_i`. The DL formula expresses that the class `&thing` consists of individuals `a_i`.

In RDF/XML syntax, each individual, each class, each role, and class `&thing` are expressed as an RDF/XML tag, with their characteristics (if any) as sub-tags. When the clause-by-clause translation of the whole problem is completed, for each individual, each class, each role, and the class `&thing`, all the characteristics that were

Table 4.4: CNF Clauses and Equivalent `&thing` Class Descriptions

Clause c	$saffron(c)$
`~p(X)`	`&thing = ~p`
$\|_{i=1}^{m}$`p_i(X) |` $\|_{j=1}^{n}$`~q_j(X)` where $m \geq 0$, $n \geq 0$, and $(m+n) \geq 2$	`{ &thing = ( ++`$_{i=1}^{m}$`p_i ++ ++`$_{j=1}^{n}$`-q_j ) }`
$\|_{i=1}^{n}$`(X = a_i)` where $n \geq 1$	`{ &thing = [a_1,a_2,...,a_n] }`
$\|_{i=1}^{n}$`r(X,a_i)` where $n \geq 1$	`{&thing = ?r(_,[a_1,a_2,...,a_n])}`
$\|_{i1=1}^{m1}$`p_i1(X) |` $\|_{i2=1}^{m2}$ `~q_i2(X) |` $\|_{j=1}^{n}$`(X = a_j)` where $(m1+m2) \geq 1$, and $n \geq 1$,	`{&thing = ( ++`$_{i1=1}^{m1}$`p_i1 ++ ++`$_{i2=1}^{m2}$`-q_i2 ++ [a_1,a_2,...,a_n]) }`
$\|_{i1=1}^{m1}$`p_i1(X) |` $\|_{i2=1}^{m2}$`~q_i2(X) |` $\|_{j1=1}^{n1}$`r_j1(X,a_j1) |` $\|_{j2=1}^{n2}$`~r_j2(X,b_j2)` where $(m1+m2) \geq 0$, and $(n1+n2) \geq 1$	`{&thing = ( ++`$_{i1=1}^{m1}$`p_i1 ++ ++`$_{i2=1}^{m2}$`-q_i2 ++ ++`$_{j1=1}^{n1}$`!r_j1(_,a_j1) ++ ++`$_{j2=1}^{n2}$`-(!r_j2(_,b_j2)) }`

found during the translation are gathered and combined into an RDF/XML tag.

In DLF syntax, the type of individuals are expressed in a DLF type formula. If an individual belongs to more than one class, then the type of the individual is the intersection of those classes. The types of classes are always `$tType`. The type of roles are always `&thing * &thing > $o` because no CNF clause is translated in a way that provide the domain or the range of a role. All other CNF clauses are translated to DLF logic formulae with the same TPTP formula role as the original CNF clause.

Example 35 can be expressed in CNF as in Example 36. The CNF problem in Example 36 can be translated to DL, and the result of the translation in DLF is Example 37, and in RDF/XML, without the header and footer of the `.owl` file, is Example 38. The CNF problem is unsatisfiable because the negated conjecture `sarah_is_sibling_of_mary` is in contradiction with the two axioms `mary_is_sibling_of_sarah` and `sibling_is_symmetric`. The negated conjecture `joe_is_dad_of_youngest_child` is also in contradiction with the axioms `mary_is_child_of_joe`, `mary_is_youngest_child`,

and `dad_is_subproperty_of_inverse_of_child`. `Konclude` is used to confrim that the resultant DL ontology is inconsistent.

Example 35

Sarah is a person. Mary is a female. Joe is a person. Joe is not a female. Mary is Sarah's sibling. Joe is Sarah's dad. Mary is Joe's child. Mary and Sarah are different people. Mary is the youngest child. Sibling is symmetric. Every dad's child is a child of the dad. Every female is a person.

Therefore: Sarah is Mary's sibling. Joe is the youngest child's dad.

Example 36

```
cnf(sarah_is_a_person,axiom,
    ( person(sarah) )).

cnf(mary_is_a_female,axiom,
    female(mary) )).

cnf(joe_is_a_person,axiom,
    person(joe) )).

cnf(joe_is_not_a_female,axiom,
    ~female(joe) )).

cnf(mary_is_sibling_of_sarah,axiom,
    sibling(mary,sarah) )).

cnf(joe_is_dad_of_sarah,axiom,
    dad(joe,sarah) )).

cnf(mary_is_child_of_joe,axiom,
    child(mary,joe) )).

cnf(mary_and_sarah_are_different,axiom,
    mary != sarah )).
```

```
cnf(mary_is_youngest_child,axiom,
    mary = youngest_child )).

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

cnf(dad_is_subproperty_of_inverse_of_child,axiom,
    ~dad(X,Y)
    | child(Y,X) )).

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

cnf(sarah_is__not_sibling_of_mary,negated_conjecture,
    ~sibling(sarah,mary) )).

cnf(joe_is__not_dad_of_youngest_child,negated_conjecture,
    ~dad(joe,youngest_child) )).
```

Example 37

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

dlf(female_type,type,
  female: $tType)).

dlf(sarah_is_a_person,type,
  sarah: person )).

dlf(mary_is_a_female,type,
  mary: female )).

dlf(youngest_child_type,type,
  youngest_child: &thing).

dlf(joe_is_a_person_and_joe_is_not_a_female,type,
  joe: ( person ** -female ) )).

dlf(sibling_type,type,
  sibling: ( &thing * &thing ) > $o ).
```

```
dlf(dad_type,type,
   dad: ( &thing * &thing ) > $o )).

dlf(child_type,type,
   child: ( &thing * &thing ) > $o )).

dlf(inv_child_type,type,
   inv_child: ( &thing * &thing ) > $o )).

dlf(mary_is_sibling_of_sarah,axiom,
   sibling(mary,sarah) )).

dlf(joe_is_dad_of_sarah,axiom,
   dad(joe,sarah) )).

dlf(mary_is_child_of_joe,axiom,
   child(mary,joe) )).

dlf(mary_and_sarah_are_different,axiom,
   mary != sarah )).

dlf(mary_is_youngest_child,axiom,
   mary = youngest_child )).
dlf(sibling_is_symmetric,axiom,
   $symmetric(sibling) ).

dlf(inv_child_is_inverse_of_child,axiom,
   inv_child -= child ).

dlf(dad_is_subproperty_of_inverse_of_child,axiom,
   dad << inv_child ).
```

Example 38

```
<owl:NamedIndividual rdf:about="&family-ontology;joe">
    <family-ontology:dad rdf:resource="&family-ontology;sarah"/>
    <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;child"/>
<owl:ObjectProperty rdf:about="&family-ontology;dad">
    <rdfs:subPropertyOf>
      <rdf:Description>
        <owl:inverseOf rdf:resource="&family-ontology;child"/>
      </rdf:Description>
    </rdfs:subPropertyOf>
</owl:ObjectProperty>

<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDifferent"/>
    <owl:distinctMembers rdf:parseType="Collection">
      <rdf:Description rdf:about="&family-ontology;mary"/>
      <rdf:Description rdf:about="&family-ontology;sarah"/>
    </owl:distinctMembers>
</rdf:Description>

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

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

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

<owl:NamedIndividual rdf:about="&family-ontology;mary">
    <family-ontology:child rdf:resource="&family-ontology;joe"/>
    <family-ontology:sibling rdf:resource="&family-ontology;sarah"/>
    <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>
```

4.3 Implementation of `Saffron`

`Saffron` has two implementations. One implementation outputs the resultant DL problem in RDF/XML, and the other implementation outputs the resultant DL problem in DLF. Both implementations 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. The three translation related modules are named `translation`, `gather`, and `generate-output`.

The two implementations are very different in the `gather`, and `generate-output` modules, and slightly different in the `translation` module. In both implementations, the `translation` module translates the CNF problem clause-by-clause. Each clause is translated to a set of DL formulae. In the RDF/XML implementation the output from the `translation` module is passed to the `gather` module for further processing, and then the output of the `gather` module is passed to the `generate-output` module. In the DLF implementation the output from the `translation` module that needs further processing is passed to the `gather` module, and the rest of the output is passed directly to the `generate-output` module. In both DLF and RDF/XML implementations, the `generate-output` module uses the data received from the other

module(s) to generate the output DL problem. Appendix B is the pseudo code for the implementation that outputs RDF/XML syntax.

4.3.1 The `translation` Module

For each CNF clause, the `translation` module tries to translate the clause. If the clause is not DL-able, it is kept as an untranslated clause to be used later for statistics purposes, such as calculating the translation percentage. The translation module uses a translation table that maps clause structures to their translation in DL. The translation table is illustrated in Tables 4.5 and 4.6.

To translate a CNF clause c, all the constants, unary predicates, binary predicates, and distinct variables of the clause c are extracted. The clause c is then filtered out if it is obvious that c is not DL-able. The filtering is done by checking the sizes of the sets of constants, unary predicates, binary predicates, and distinct variables of the clause. If the sizes of these sets does not match any of the DL-able clause structures in the translation table, then c is filtered out, and kept as untranslated. The filtering process narrows down the search space for a possible translation of the clause c.

The CNF clauses of Example 39 have no constants, no unary predicates, and two binary predicates. Clause 39.a has four distinct variables, and other clauses have two distinct variables. There is no DL-able clause structure with four distinct variables, so the clause 39.a is filtered out. However, clauses 39.b and 39.c are kept for the next step because the sizes of their sets of constants, unary predicates, binary predicates and variables match the DL-able clause structures in two rows of Table 4.5.

Example 39

39.a. `dad(X,Y) | child(Z,T)`

39.b. `~dad(X,X) | child(Y,Y)`

39.c. `~dad(X,Y) | child(X,Y)`

39.d. `~dad(X,Y) | child(Y,X)`

The last step of the translation is to determine if there is an exact match for c. All features of the clause c, such as polarity of its literals, and the order of appearance of variables and constants in the arguments of binary predicates, are taken into account when finding a match.

In Example 39, the clauses 39.b, 39.c and 39.d share same sets of constants, unary predicates, binary predicates, and distinct variables. However, the clause 39.b is not DL-able, and the clauses 39.c and 39.d match the clause structures in two rows of the Table 4.5. Examples 40.a and 40.b are the translation for clauses 39.c and 39.d correspondingly.

Example 40

40.a. {`dad << child`}

40.b. {`inv_child -= child`,
`dad << inv_child` }

In the RDF/XML implementation, the output of the `translation` module on a DL-able clause c is a set of *pairs*. The first argument of the pair contains information

Table 4.5: CNF and DL Equivalent Formulae

Clause c	Individuals	Classes	Roles	Variables	$saffron(c)$
`p(a)`	{a}	{p}	{}	{}	{ `a: p` }
`~p(a)`					{ `a: -p` }
$\mid_{i=1}^{m}$`p_i(a)` $\mid \mid_{j=1}^{n}$`~q_j(a)`	$\{a\}$	$\{p\_i, q\_j \mid 0 \leq i \leq m, 0 \leq j \leq n, (m+n) \geq 2\}$	{}	{}	{ `a:` $+\!\!+_{i=1}^{n}$`p_i ++` $+\!\!+_{j=1}^{m}$`-q_j` }
r(a,b)	{a,b}	{}	{r}	{}	{ `r(a,b)` }
~r(a,b)					{ `~r(a,b` }
`a=b`	{a,b}	{}	{=}	{}	{ `a=b` }
`a != b`					{ `a != b` }
$\mid_{i=1}^{n} r(a, b\_i)$ where $n \geq 2$	$\{a, b\_i \mid 0 \leq i \leq n, n \geq 2\}$	{}	{r}	{}	{ `a: ?r(_,[b_1,b_2,...b_n])` }
$\mid_{i=1}^{n} a = b\_i$ where $n \geq 2$	$\{a, b\_i \mid 0 \leq i \leq n, n \geq 2\}$	{}	{=}	{}	{ `a: [b_1,b_2,...b_n]` }
`p(X)`	{}	{p}	{}	{X}	{ `p=&thing` }
`~p(X)`					{ `&thing=~p` }
`~p(X) | q(X)`	{}	{p,q}	{}	{X}	{ `p << q` }
`~p(X) | r(X,Y)`	{}	{p}	{r}	{X,Y}	{ `p << ?r(_,&thing)` }
`p(X) | ~r(X,Y)`					{ `?r(_,&thing) << p` }
`p(X) | ~q(Y) | ~r(X,Y)`	{}	{p,q}	{r}	{X,Y}	{ `?r(_,q) << p` }
$\mid_{i=1}^{m}$`p_i(X)` $\mid \mid_{j=1}^{n}$`~q_j(X)`	{}	$\{p\_i, q\_j \mid 0 \leq i \leq m, 0 \leq j \leq n, (m+n) \geq 2\}$	{}	$\{X\}$	{ `&thing =` $+\!\!+_{i=1}^{n}$`p_i ++` $+\!\!+_{j=1}^{m}$`-q_j` }
$\mid_{i=1}^{m}$`X = a_i`	$\{a\_i \mid 0 \leq i \leq n,\ n \geq 2\}$	{}	{=}	{X}	{ `&thing= [a_1,a_2,...,a_n]` }
$\mid_{i=1}^{m}$`r(X,a_i)`	$\{a\_i, \mid 0 \leq i \leq n,\ n \geq 2\}$	{}	{r}	{X}	{ `&thing= ?r(_,[a_1,a_2,...,a_n])` }
`r(X,X)`	{}	{ }	{r}	{X}	{ `$reflexive(r)` }
~r(X,X)					{ `$irreflexive(r)` }
`~r(X,Y) | r(Y,X)`	{}	{}	{r}	{X,Y}	{ `$symmetric(r)` }
`~r(X,Y) | ~r(Y,X)`					{ `$asymmetric(r)` }
`~r(X,Y) | ~r(Y,Z) | r(X,Z)`	{}	{}	{r}	{X,Y,Z}	{ `$transitive(r)` }
`~r(X,Y) | s(X,Y)`	{}	{}	{r,s}	{X,Y}	{ `r << s` }
`~r(X,Y) | s(Y,X)`					{ `inv_s -= s, r << inv_s` }
`~r1(X,Y) | ~r2(Y,Z) | s(X,Y)`	{}	{}	{r1,r2,s}	{X,Y,Z}	{ `s >> (r1 @ r2)` }
`~r1(X,Y) | ~r2(X,Z) | s(Y,Z)`					{ `inv_r2 -= r2, inv_s -= s, inv_s >> (inv_r2 @ r1)` }

Table 4.6: CNF and DL Equivalent Formulae

Clause c	Individuals	Classes	Roles	Variables	$saffron(c)$
$\mid_{i1=1}^{n1} p\_i1(X) \mid \mid_{i2=1}^{n2} \sim pp\_i2(X) \mid$ $\mid_{j1=1}^{m1} q\_j1(Y) \mid \mid_{j2=1}^{m2} \sim qp\_j2(Y) \mid$ $\mid_{k=1}^{h1} \sim r_k(X,Y)$ where $(n1 + n2) \geq 1$, $(m1 + m2) \geq 1$, and $h \geq 1$	$\{\}$	$\{p\_i1, pp\_i2,$ $q\_j1, qp\_j2 \mid$ $0 \leq i1 \leq n1,$ $0 \leq i2 \leq n2,$ $0 \leq j1 \leq m1,$ $0 \leq j2 \leq m2\}$	$\{r_k \mid 0 \leq k \leq h\}$	{X,Y}	$\{ \; (* *_{k=1}^{h} r_k(\_, * *_{j1=1}^{m1} -q\_j1 * *$ $* *_{j2=1}^{m2} qp\_j2)) \; \texttt{<<}$ $(+ +_{i1=1}^{n1} p\_i1 \; + + \; + +_{i2=1}^{n2} -pp\_i2) \}$
$\mid_{i1=1}^{n1} p\_i1(X) \mid \mid_{i2=1}^{n2} \sim pp\_i2(X) \mid$ $\mid_{i3=1}^{n3} s\_i3(X, a\_i3) \mid$ $\mid_{i4=1}^{n4} \sim sp_{i4}(X, a'\_i4) \mid$ $\mid_{j1=1}^{m1} q\_j1(Y) \mid \mid_{j2=1}^{m2} \sim qp\_j2(Y) \mid$ $\mid_{j3=1}^{m3} u\_j3(Y, b\_j3) \mid$ $\mid_{j4=1}^{m4} \sim up_{j4}(Y, bp\_j4) \mid$ $\mid_{k=1}^{h} \sim r_k(X,Y)$ where $(n1 + n2 + n3 + n4) \geq 1$, $(m1 + m2 + m3 + m4) \geq 1$, and $h \geq 1$	$\{a\_i3, a'\_i4,$ $b\_j3, bp\_j4 \mid$ $0 \leq i3 \leq n3,$ $0 \leq i4 \leq n4,$ $0 \leq j3 \leq m3,$ $0 \leq j4 \leq m4\}$	$\{p\_i1, pp\_i2,$ $q\_j1, q\_j2 \mid$ $0 \leq i1 \leq n1,$ $0 \leq i2 \leq n2,$ $0 \leq j1 \leq m1,$ $0 \leq j2 \leq m2\}$	$\{s\_i3, sp_{i4},$ $u_{j4}, up_{j4}, r_k \mid$ $0 \leq i3 \leq n3,$ $0 \leq i4 \leq n4,$ $0 \leq j3 \leq m3,$ $0 \leq j4 \leq m4\}$	{X,Y}	$\{ \; **_{k=1}^{h} ?r_k(\_, (* *_{j1=1}^{m1} -q\_j1 * *$ $* *_{j2=1}^{m2} qp\_j2 * *$ $* *_{j3=1}^{m3} -(!\, u\_j3(\_, b\_j3)) * *$ $**_{j4=1}^{m4} !\, up_{j4}(\_, bp\_j4)))$ $\texttt{<<}$ $(+ +_{i1=1}^{n1} p\_i1 \; + + \; + +_{i2=1}^{n2} -pp\_i2 \; + +$ $+ +_{i3=1}^{n3} !\, s\_i3(\_, a\_i3) \; + +$ $+ +_{i4=1}^{n4} -(!\, sp_{i4}(\_, a'\_i4)))\}$
$\mid_{i1=1}^{m1} p\_i1(X) \mid \; \mid_{i2=1}^{m2} \sim q_{i2}(X) \mid$ $\mid_{j=1}^{n} (X = a_j)$ where $m1 \geq 0$, $m2 \geq 0$, $(m1 + m2) \geq 1$, $n \geq 1$,	$\{a_j \mid 0 \leq j \leq n,$ $n \geq 1\}$	$\{p\_i1, q_{i2} \mid$ $0 \leq i1 \leq m1,$ $0 \leq i2 \leq m2,$ $(m1 + m2) \geq 1\}$	$\{=\}$	{X}	$\{ \, \&thing \; = \; (+ +_{i1=1}^{m1} p\_i1 \; + +$ $+ +_{i2=1}^{m2} -q_{i2} \; + +$ $[a_1, a_2, \ldots, a_n]) \}$
$\mid_{i1=1}^{m1} p\_i1(X) \mid \; \mid_{i2=1}^{m2} \sim q_{i2}(X) \mid$ $\mid_{j1=1}^{n1} r_{j1}(X, a_{j1}) \mid$ $\mid_{j2=1}^{n2} \sim r'_{j2}(X, b_{j2})$ where $m1 \geq 0$, $m2 \geq 0$, $(m1 + m2) \geq 1$, and $(n1 + n2) \geq 1$	$\{a_{j1}, b_{j2} \mid$ $0 \leq j1 \leq n1,$ $0 \leq j2 \leq n2,$ $(n1 + n2) \geq 1\}$	$\{p\_i1, q_{i2} \mid$ $0 \leq i1 \leq m1,$ $0 \leq i2 \leq m2,$ $(m1 + m2) \geq 1\}$	$\{r_{j1}, r'_{j2} \mid$ $0 \leq j1 \leq n1,$ $0 \leq j2 \leq n2,$ $(n1 + n2) \geq 1\}$	{X}	$\{ \, \&thing \; =$ $(+ +_{i1=1}^{m1} p\_i1 \; + +$ $+ +_{i2=1}^{m2} -q_{i2} \; + +$ $+ +_{j1=1}^{n1} !\, r_j(\_, a_{j1}) \; + +$ $+ +_{j2=1}^{n2} -(!\, r'_j(\_, b_{j2})) \}$

that uniquely specifies an RDF/XML tag. The second argument of the pair is a DL formula expressed as a sub-tag. All the pairs are passed to the `gather` module to create completed RDF/XML tags.

In the DLF implementation, the output of the `translation` module on a DL-able clause c is a set of DL formulae and the list of the names of the constants, unary predicates, and binary predicates of c. DL formulae expressing that an individual belongs to a class, along with the names of constants, unary predicates and binary predicates of the clause c, are passed to the `gather` module. All other DL formulae are expressed as DLF logic formulae, and passed directly to the `generate-output` module.

4.3.2 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 a CNF problem. After the `translation` module has translated all the DL-able clauses, the `gather` module processes the data received from the `translation` module.

In the RDF/XML implementation, the `gather` module collects all the resultant pairs from the `translation` module. It merges the sub-tags of all the pairs whose first arguments specify the same RDF/XML tag, to create a completed RDF/XML tag that describes the individual, class, role, or `&thing`. The output of the `gather` module is a set of RDF/XML tags that are passed to the `generate-output` module.

In the DLF implementation, the `gather` module collects all the DL formulae and the list of names received from the `translation` module. A type formula needs to be created for each individual, each class, and each role in DLF syntax. If an individual belongs to more than one class, then the type of the individual is the intersection of those classes. If an individual does not belong to any class, the type of the individual is `&thing`. The types of classes are always `$tType`. The types of roles are always `&thing * &thing > $o`, because no CNF clause is translated in a way to provide the domain or range of a role. The output of the `gather` module is a set of DLF type formulae that is passed to the `generate-output` module.

The `generate-output` module creates an appropriate definition header for the DL problem, depending on the desired syntax of the output DL problem. The definition header and the set of either RDF/XML tags or DLF formulae are written to an output file.

4.4 Proof of the Soundness of the Translation

It is necessary to prove that the CNF to DL translation procedure is sound. In this section a mathematical proof is provided.

Theorem 4.1 The CNF problem *problem_dl* (all the DL-able clauses of the CNF problem *problem*) is satisfiable if and only if the DL problem $saffron(problem\_dl)$ is satisfiable.

Proof The CNF problem *problem_dl* is satisfiable if and only if it has a model. The DL problem $saffron(problem\_dl)$ is satisfiable if and only if it has a model. To

prove the theorem, it is necessary to prove that *problem_dl* has a model if and only if $saffron(problem)$ has a model. Since a CNF problem has a model if and only if it has a Herbrand model, it is enough to prove that *problem_dl* has a Herbrand model HM if and only if $saffron(problem)$ has a model M. A CNF problem has a Herbrand model HM if and only if all the clauses are `TRUE` in HM. Thus, it is necessary to prove that every clause c is `TRUE` in HM if and only if $saffron(c)$ is `TRUE` in M. A bidirectional proof-by-construction is presented here. In one direction, HM is constructed from M, and in the other direction M is constructed from HM.

Models for CNF problems are explained in detail in Section 2.3.2.2. The components of the Herbrand model $HM = (D_{HM}, F_{HM}, R_{HM})$ for *problem_dl* are defined as follows:

- D_{HM} is the domain of HM, which is the Herbrand Universe. Since *problem* is EPR, the Herbrand Universe is finite and consists of the constants of the *problem_dl*. If *problem_dl* has no constants, a dummy element is added to D_{HM}.

- F_{HM} is the function map. It is the identity function. All the constants in *problem_dl*, which are the only functions in *problem_dl*, are mapped to themselves.

- R_{HM} is the predicate map. It is the subset of the Herbrand Base (HB_{TRUE}) that is `TRUE` . That means a ground atom A is `TRUE` in HM if and only if $A \in HB_{TRUE}$.

Models for DL problems are explained in detail in Section 2.2.2. The components of the model $M = (\Delta_M^I, .^I)$ for $saffron(problem)$ are defined as follows:

- Δ_M^I is the domain of M. It is the set consisting of the interpretations of the individuals in $saffron(problem\_dl)$. (Recall from Section 4.2, the individuals in $saffron(problem\_dl)$ are the constants in $problem\_dl$).

- $.^I$ is the interpretation function for M.

HM and M can be constructed one from another by the following rules, which are the basis of the translation function $saffron$.

- $a \in D_{HM}$ iff $a^I \in \Delta_M$.
- $p(a)$ is `TRUE` in HM
  iff
  $p(a) \in HB_{TRUE}$
  iff
  $a^I \in p^I$ in M.

- $r(a, b)$ is `TRUE` in HM
  iff
  $r(a, b) \in HB_{TRUE}$
  iff
  $(a^I, b^I) \in r^I$ in M.

- $a = b$ is `TRUE` in HM
  iff
  $(a = b) \in HB_{TRUE}$
  iff
  $a^I = b^I$ in M.

For each translation rule $c \stackrel{saffron}{\longmapsto} saffron(c)$, a lemma is proven that c is `TRUE` in *HM iff* $saffron(c)$ is `TRUE` in M. The syntax for the clause c, which also appears in the first line of each lemma, is CNF. Steps from CNF syntax to CNF semantics are defined in Section 2.3.2.2. The syntax for the DL formulae in $saffron(c)$, which also appear in the last line of each lemma, is DLF. Steps from DL syntax to DL semantics are defined in Section 2.2.2.

Notation. The variables X, Y and Z range over D_{HM}, and the variables x, y and z range over Δ_M.

Lemma 4.2 Translating `p(a)` $\stackrel{saffron}{\longmapsto}$ {`a : p`}:

Proof
`p(a)` in *HM*
iff
$a^I \in p^I$ in M
iff
`a : p` $\Box$.

Lemma 4.3 Translating `~p(a)` $\stackrel{saffron}{\longmapsto}$ { `a : -p`}:

Proof
`~p(a)` in *HM*
iff
$a^I \notin p^I$ in M
iff
$a^I \in \Delta_M^I \backslash p^I$ in M
iff
$a^I \in (\neg p)^I$ in M
iff
`a : -p` in M $\Box$.

Lemma 4.4 Translating $|_{i=1}^{n}$`p_i(a)` `|` $|_{i=1}^{m}$`~q_i(a)` $\stackrel{saffron}{\longmapsto}$

{ `a: (`$+\!\!+_{i=1}^{n}$`p_i ++` $+\!\!+_{i=1}^{m}$`(-q_i))`} $((m+n) \geq 2)$:

Proof

$|_{i=1}^{n}$`p_i(a) | |_i` $= 1^m$`q_i(a)` in HM
iff
$\bigvee_{i=1}^{n} p\_i(a) \sim (\bigwedge_{i=1}^{m} q\_i(a))$ in HM
iff
$\bigvee_{i=1}^{n} (a^I \in p\_i^I) \vee \sim \bigwedge_{i=1}^{m} (a^I \in q\_i^I)$ in M
iff
$\bigvee_{i=1}^{n} (a^I \in p\_i^I) \vee \bigvee_{i=1}^{m} (a^I \notin q\_i^I)$ in M
iff
$\bigvee_{i=1}^{n} (a^I \in p\_i^I) \vee \bigvee_{i=1}^{m} (a^I \in (\Delta_M^I \backslash q\_i^I))$ in M
iff
$\bigvee_{i=1}^{n} (a^I \in p\_i^I) \vee \bigvee_{i=1}^{m} (a^I \in (\neg q\_i)^I)$ in M
iff
$a^I \in (\bigcup_{i=1}^{n} (p\_i^I) \cup \bigcup_{i=1}^{m} (\neg q\_i)^I)$ in M
iff
`a: (++`$_{i=1}^{n}$`p_i ++ ++`$_{i=1}^{m}$`(-q_i))` in M $\square$.

Lemma 4.5 Translating `r(a,b)` $\overset{saffron}{\longmapsto}$ { `r(a,b)`}:

Proof

`r(a,b)` in HM
iff
$(a^I, b^I) \in r^I$ in M
iff
`r(a,b)` in M $\square$.

Lemma 4.6 Translating `~r(a,b)` $\overset{saffron}{\longmapsto}$ {`~r(a,b)`}:

Proof

`~r(a,b)` in HM
iff
$(a^I, b^I) \notin r^I$ in M
iff
`~r(a,b)` in M $\square$.

Lemma 4.7 Translating `a = b` $\overset{saffron}{\longmapsto}$ {`a = b`}:

Proof

`a = b` in *HM*
iff
$a^I = b^I$ in M
iff
`a = b` in M $\square$.

Lemma 4.8 Translating `a != b` $\overset{saffron}{\longmapsto}$ {`a != b`}:

Proof

`a != b` in *HM*
iff
$a^I \neq b^I$ in M
iff
`a != b` in M $\square$.

Lemma 4.9 Translating $|_{i=1}^{n}$`r(a,b_i)` $\overset{saffron}{\longmapsto}$ {`a: ?r(_,[b_1,b_2,...,b_n])`}($n \geq$ 2):

Proof

$|_{i=1}^{n}$`r(a,b_i)` in *HM*
iff
$\bigvee_i^n((a^I, b\_i^I) \in r^I)$ in M
iff
$(a^I, b\_1^I) \in r^I \vee (a^I, b\_2^I) \in r^I \vee ... \vee (a^I, b\_n^I) \in r^I$ in M
iff
$\exists y \in \Delta_M^I : ((a^I, y) \in r^I \wedge (\bigvee_{i=1}^n (y = b\_i^I))$ in M
iff
$\exists y \in \Delta_M^I : ((a^I, y) \in r^I \wedge y \in \{b\_1^I, b\_2^I, .., b\_n^I\})$ in M
iff
$\exists y \in \Delta_M^I : ((a^I, y) \in r^I \wedge y \in \{b\_1, b\_2, .., b\_n\}^I)$ in M
iff
$a^I \in (\exists r.\{b\_1, b\_2, .., b\_n\})^I$ in M
iff
`a: ?r(_,[b_1,b_2,...,b_n])` in M $\square$.

Lemma 4.10 Translating $|_{i=1}^{n}$`(a=b_i)` $\overset{saffron}{\longmapsto}$ {`a: [b_1,b_2,...,b_n]`} ($n \geq 2$):

Proof

$|_{i=1}^{n}$`(a=b_i)` in *HM*
iff
$\bigvee_{i=1}^{n}(a^I = b\_i^I)$ in M
iff
$a^I \in \{b\_1^I, b\_2^I, .., b\_n^I\}$ in M
iff
$a^I \in \{b\_1, b\_2, .., b\_n\}^I$ in M
iff
`a: [b_1,b_2,...,b_n]` in M $\square$.

Lemma 4.11 Translating `p(X)` $\overset{saffron}{\longmapsto}$ {`p` = &`thing`}:

Proof

`p(X)` in *HM*
iff
$\forall X \in D_{HM} : p(X)$ in *HM*
iff
$\forall x \in \Delta_M^I : x \in p^I$ in M
iff
$p^I = \Delta_M^I = \top^I$ in M
iff
`p = &thing` $\square$.

Lemma 4.12 Translating `~p(X)` $\overset{saffron}{\longmapsto}$ {`&thing = (-p)`}:

Proof

`~p(X)` in *HM*
iff
$\forall X \in D_{HM} :\sim p(X)$ in *HM*
iff
$\forall x \in \Delta_M^I : x \notin p^I$ in M
iff
$\forall x \in \Delta_M^I : x \in \Delta_M^I \backslash p^I$ in M
iff
$\forall x \in \Delta_M^I : x \in (\neg p)^I$ in M
iff
$(\neg p)^I = \Delta_M^I = \top^I$ in M

iff
$\top^I = (\neg p)^I$ in M
iff
`&thing = (-p)` in M $\Box$.

Lemma 4.13 Translating `~p(X) | q(X)` $\stackrel{saffron}{\longmapsto}$ {`p << q`}:

Proof
`~p(X) | q(X)` in HM
iff
$\forall X \in D_{HM} : (p(X) \Rightarrow q(X))$ in HM
iff
$\forall x \in \Delta_M^I : (x \in p^I \Rightarrow x \in q^I)$ in M
iff
$p^I \subseteq q^I$ in M
iff
`p << q` in M $\Box$.

Lemma 4.14 Translating `~p(X) | r(X,Y)` $\stackrel{saffron}{\longmapsto}$ {`p << ?r(_,&thing)`}:

Proof
`~p(X) | r(X,Y)` in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (\sim p(X) \vee r(X,Y))$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (\sim p(X) \vee (r(X,Y) \wedge Y \in D_{HM}))$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (\sim (p(X) \wedge \sim (r(X,Y) \wedge Y \in D_{HM}))$ in HM
iff
$\sim (\exists X \in D_{HM} : \exists Y \in D_{HM} : (p(X) \wedge \sim (r(X,Y) \wedge Y \in D_{HM})))$ in HM
iff
$\sim (\exists x \in \Delta_M^I : \exists y \in \Delta_M^I : (x \in p^I \wedge \sim ((x,y) \in r^I \wedge y \in \Delta_M^I)))$ in M
iff
$\sim (\exists x \in \Delta_M^I : (x \in p^I \wedge \sim (x \in (\exists r.\top)^I)))$ in M
iff
$\forall x \in \Delta_M^I : \sim (x \in p^I \wedge x \notin (\exists r.\top)^I)$ in M
iff
$\forall x \in \Delta_M^I : (x \notin p^I \vee x \in (\exists r.\top)^I)$ in M
iff
$\forall x \in \Delta_M^I : (x \in p^I \Rightarrow x \in (\exists r.\top)^I)$ in M
iff

$p^I \subseteq (\exists r.\top)^I$ in M
iff
`p << ?r(_,&thing)` in M $\Box$.

Lemma 4.15 Translating `p(X) | ~r(X,Y)` $\overset{saffron}{\longmapsto}$ `{?r(_,&thing) << p}`:

Proof
`p(X) | ~r(X,Y)` in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (p(X) \vee \sim r(X,Y))$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (p(X) \vee \sim r(X,Y) \vee Y \notin D_{HM})$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (\sim(r(X,Y) \wedge \sim p(X) \wedge Y \in D_{HM}))$ in HM
iff
$\sim(\exists X \in D_{HM} : \exists Y \in D_{HM} : (r(X,Y) \wedge \sim p(X) \wedge Y \in D_{HM})$ in HM
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I : ((x,y) \in r^I \wedge y \in \Delta_M^I \wedge x \notin p^I))$ in M
iff
$\sim(\exists x \in \Delta_M^I : (x \in (\exists r.\top)^I \wedge x \notin p^I))$ in M
iff
$\forall x \in \Delta_M^I : \sim(x \in (\exists r.\top)^I \wedge x \notin p^I)$ in M
iff
$\forall x \in \Delta_M^I : (x \notin (\exists r.\top)^I \vee x \in p^I)$ in M
iff
$\forall x \in \Delta_M^I : (x \in (\exists r.\top)^I \Rightarrow x \in p^I)$ in M
iff
$(\exists r.\top)^I \subseteq p^I$ in M
iff
`?r(_,&thing) << p` in M $\Box$.

Lemma 4.16 Translating `~r(X,Y) | ~q(Y) | p(X)` $\overset{saffron}{\longmapsto}$ `{?r(_,q) << p}`:

Proof
$\sim r(X,Y) \mid \sim q(Y) \mid p(X)$
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (\sim r(X,Y) \vee \sim q(Y) \vee p(X))$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (\sim(r(X,Y) \wedge q(Y) \wedge \sim p(X)))$ in HM
iff
$\sim(\exists X \in D_{HM} : \exists Y \in D_{HM} : (r(X,Y) \wedge q(Y) \wedge \sim p(X)))$ in HM

iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I : ((x,y) \in r^I \wedge y \in q^I \wedge x \notin p^I))$ in M
iff
$\sim(\exists x \in \Delta_M^I : (x \in (\exists r.q)^I \wedge x \notin p^I))$ in M
iff
$\sim(\exists x \in \Delta_M^I : \sim(x \notin (\exists r.q)^I \vee x \in p^I))$ in M
iff
$\forall x \in \Delta_M^I : (x \notin (\exists r.q)^I \vee x \in p^I)$ in M
iff
$\forall x \in \Delta_M^I : (x \in (\exists r.q)^I \Rightarrow x \in p^I)$ in M
iff
$(\exists r.q)^I \subseteq p^I$ in M
iff
`?r(_,q) << p` in M $\Box$.

Lemma 4.17 Translating $|_{j1=1}^{m1}$`q_j1(Y) | `$|_{j2=1}^{m2}$`~qp_j2(Y) | `$|_{k=1}^{h}$`~r_k(X,Y)` $\xrightarrow{saffron}$

{ `(` $**_{k=1}^{h}$`?r_k(_,` $**_{j1=1}^{m1}$`(-q_j1) **` $**_{j2=1}^{m2}$`qp_j2)) << &nothing` } $(m1+m2) \geq 1$,

and $h \geq 1$:

Proof
$|_{j1=1}^{m1}$`q_j1(Y) | `$|_{j2=1}^{m2}$`~qp_j2(Y) | `$|_{k=1}^{h}$`~r_k(X,Y)`
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} :$
$(\bigvee_{j1=1}^{m1} q\_j1(Y) \vee \bigvee_{j2=1}^{m2} \sim qp\_j2(Y) \vee \bigvee_{k=1}^{h} \sim r_k(X,Y))$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} :$
$\sim(\bigwedge_{j1=1}^{m1} \sim q\_j1(Y) \wedge \bigwedge_{j2=1}^{m2} qp\_j2(Y) \wedge \bigwedge_{k=1}^{h} r_k(X,Y))$ in HM
iff
$\sim(\exists X \in D_{HM} : \exists Y \in D_{HM} :$
$(\bigwedge_{j1=1}^{m1} \sim q\_j1(Y) \wedge \bigwedge_{j2=1}^{m2} qp\_j2(Y) \wedge \bigwedge_{k=1}^{h} r_k(X,Y)))$ in HM
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$(\bigwedge_{j1=1}^{m1} y \in (\neg q\_j1)^I \wedge \bigwedge_{j2=1}^{m2} y \in qp\_j2^I \wedge \bigwedge_{k=1}^{h} (x,y) \in r_k^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$(y \in (\bigcap_{j1=1}^{m1} (\neg q\_j1)^I \cap \bigcap_{j2=1}^{m2} qp\_j2^I) \wedge \bigwedge_{k=1}^{h} (x,y) \in r_k^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$y \in (\prod_{j1=1}^{m1} (\neg q\_j1) \sqcap \prod_{j2=1}^{m2} qp\_j2)^I \wedge \bigwedge_{k=1}^{h} (x,y) \in r_k^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$

$\bigwedge_{k=1}^{h}(y \in (\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2)^I \wedge (x,y) \in r_k^I))$ in M
iff
$\sim(\exists x \in \Delta_M^I :$
$\bigwedge_{k=1}^{h}(x \in (\exists r\_k.(\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2))^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I :$
$x \in \bigcap_{k=1}^{h}(\exists r\_k.(\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2))^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I :$
$x \in (**_{k=1}^{h} ?r_k(\_, \bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2))^I)$ in M
iff
$(\bigsqcap_{k=1}^{h} \exists r\_k.(\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2))^I \subseteq \emptyset$ in M
iff
$**_{k=1}^{h}$`?r_k(_, `$**_{j1=1}^{m1}$`(-q_j1) ** `$**_{j2=1}^{m2}$`qp_j2)) << &nothing` in M $\Box$.

Lemma 4.18 Translating $|_{i1=1}^{m1}$`p_i1(X) | `$|_{i2=1}^{m2}$`~p_i2(X) | `$|_{j1=1}^{m1}$`q_j1(Y) | `

$|_{j2=1}^{m2}$`~qp_j2(Y) | `$|_{k=1}^{h}$`~r_k(X,Y)` $\xrightarrow{saffron}$

`{ (`$**_{k=1}^{h}$`?r_k(_, `$**_{j1=1}^{m1}$`(-q_j1) ** `$**_{j2=1}^{m2}$`qp_j2)) <<`

`(`$++_{i1=1}^{n1}$`p_i1 ++ `$++_{i2=1}^{n2}$`(-p_i2)) }`

$(n1 + n2) \geq 1$, $(m1 + m2) \geq 1$, and $h \geq 1$:

Proof
$|_{i1=1}^{m1}$`p_i1(X) | `$|_{i2=1}^{m2}$`~p_i2(X) | `$|_{j1=1}^{m1}$`q_j1(Y) | `$|_{j2=1}^{m2}$`~qp_j2(Y) |`
$|_{k=1}^{h}$`~r_k(X,Y)` in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} :$
$(\bigvee_{i1=1}^{n1} p\_i1(X) \vee \bigvee_{i2=1}^{n2} \sim pp\_i2(X) \vee \bigvee_{k=1}^{h} \sim r_k(X,Y) \vee \bigvee_{j1=1}^{m1} q\_j1(Y) \vee$
$\bigvee_{j2=1}^{m2} \sim qp\_j2(Y))$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : \sim(\sim(\bigvee_{i1=1}^{n1} p\_i1(X) \vee \bigvee_{i2=1}^{n2} \sim pp\_i2(X))$
$\wedge \bigwedge_{k=1}^{h} r_k(X,Y) \wedge \bigwedge_{j1=1}^{m1} \sim q\_j1(Y) \wedge \bigwedge_{j2=1}^{m2} qp\_j2(Y))$ in HM
iff
$\sim(\exists X \in D_{HM} : \exists Y \in D_{HM} : (\sim(\bigvee_{i1=1}^{n1} p\_i1(X) \vee \bigvee_{i2=1}^{n2} \sim pp\_i2(X))$
$\wedge \bigwedge_{k=1}^{h} r_k(X,Y) \wedge \bigwedge_{j1=1}^{m1} \sim q\_j1(Y) \wedge \bigwedge_{j2=1}^{m2} qp\_j2(Y)))$ in HM
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I : (\sim(\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (-pp\_i2)^I)$
$\wedge \bigwedge_{j1=1}^{m1} y \in (\neg q\_j1)^I \wedge \bigwedge_{j2=1}^{m2} y \in qp\_j2^I \wedge \bigwedge_{k=1}^{h}(x,y) \in r_k^I)$ in M
iff

$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I : (x \notin (\bigcup_{i1=1}^{n1}(-p\_i1)^I \cup \bigcup_{i2=1}^{n2} pp\_i2^I) \wedge$
$(y \in (\bigcap_{j1=1}^{m1}(\neg q\_j1)^I \cap \bigcap_{j2=1}^{m2} qp\_j2^I) \wedge \bigwedge_{k=1}^{h}(x,y) \in r_k^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I : x \notin (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I \wedge$
$y \in (\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2)^I \wedge \bigwedge_{k=1}^{h}(x,y) \in r_k^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I : x \notin (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I \wedge$
$\bigwedge_{k=1}^{h}(y \in (\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2)^I \wedge (x,y) \in r_k^I))$ in M
iff
$\sim(\exists x \in \Delta_M^I : x \notin (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I \wedge$
$\bigwedge_{k=1}^{h} x \in (\exists r\_k.(\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2)))^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : x \notin (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I \wedge$
$x \in \bigcap_{k=1}^{h}(\exists r\_k.(\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2))^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : x \notin (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I \wedge$
$x \in (\bigsqcap_{k=1}^{h} \exists r\_k.(\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2))^I)$ in M
iff
$\forall x \in \Delta_M^I : \sim(x \notin (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I \wedge$
$x \in (\bigsqcap_{k=1}^{h} \exists r\_k.(\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2))^I)$ in M
iff
$\forall x \in \Delta_M^I : (x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I$
$\vee x \notin (\bigsqcap_{k=1}^{h} ?r_k(\_, (\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2)))^I)$ in M
iff
$\forall x \in \Delta_M^I : (x \in (\bigsqcap_{k=1}^{h} ?r_k(\_, (\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2)))^I \Rightarrow$
$x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I)$ in M
iff
$(\bigsqcap_{k=1}^{h} ?r_k(\_, (\bigsqcap_{j1=1}^{m1}(\neg q\_j1) \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2)))^I \subseteq (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2))^I$ in M
iff
$**_{k=1}^{h}$`?r_k(_, `$**_{j1=1}^{m1}$`(-q_j1) **` $**_{j2=1}^{m2}$`qp_j2)) <<`
`(++`$_{i1=1}^{n1}$`p_i1 ++ ++`$_{i2=1}^{n2}$`(-p_i2))` in M $\square$.

Lemma 4.19 Translating

$|_{i1=1}^{n1}$`p_i1(X) | `$|_{i2=1}^{n2}$`~pp_i2(X) | `$|_{i3=1}^{n3}$`s_i3(X,a_i3) | `$|_{i4=1}^{n4}$`~sp_i4(X,a'_i4) |`

$|_{j1=1}^{m1}$`q_j1(Y) | `$|_{j2=1}^{m2}$`~qp_j2(Y) | `$|_{j3=1}^{m3}$`u_j3(Y,b_j3) | `$|_{j4=1}^{m4}$`~up_j4(Y,bp_j4) |`

$|_{k=1}^{h}$`~r_k(X,Y)` $\xrightarrow{saffron}$

`(` $**_{k=1}^{h}$`?r_k(_, ** (` $**_{j1=1}^{m1}$`-q_j1 **` $**_{j2=1}^{m2}$`qp_j2 **` $**_{j3=1}^{m3}$`-(!u_j3(_,b_j3)) **`

$**_{j4=1}^{m4}$`!up_j4(_,bp_j4)) ))`

`<< ( ++`$_{i1=1}^{n1}$`p_i1 ++ ++`$_{i2=1}^{n2}$`-pp_i2 ++ ++`$_{i3=1}^{n3}$`!s_i3(_,a_i3) ++`

`++`$_{i4=1}^{n4}$`-(!sp_i4(_,a'_i4)))`

$((n1 + n2 + n3 + n4) \geq 1$, $(m1 + m2 + m3 + m4) \geq 1$, and $h \geq 1)$:

Proof

$|_{i1=1}^{n1}$`p_i1(X) |` $|_{i2=1}^{n2}$`~pp_i2(X) |` $|_{i3=1}^{n3}$`s_i3(X,a_i3) |` $|_{i4=1}^{n4}$`~sp_i4(X,a'_i4) |` $|_{j1=1}^{m1}$`q_j1(Y) |` $|_{j2=1}^{m2}$`~qp_j2(Y) |` $|_{j3=1}^{m3}$`u_j3(Y,b_j3) |` $|_{j4=1}^{m4}$`~up_j4(Y,bp_j4) |` $|_{k=1}^{h}$`~r_k(X,Y)` in HM

iff

$\forall X \in D_{HM} : \forall Y \in D_{HM} :$
$(\bigvee_{i1=1}^{n1} p\_i1(X) \vee \bigvee_{i2=1}^{n2} \sim pp\_i2(X) \vee \bigvee_{i3=1}^{n3} s\_i3(X, a\_i3) \vee \bigvee_{i4=1}^{n4} \sim sp\_i4(X, a'\_i4) \vee$
$\bigvee_{j1=1}^{m1} q\_j1(Y) \vee \bigvee_{j2=1}^{m2} \sim qp\_j2(Y) \vee \bigvee_{j3=1}^{m3} u\_j3(Y, b\_j3) \vee \bigvee_{j4=1}^{m4} \sim up\_j4(Y, bp\_j4) \vee$
$\bigvee_{k=1}^{h} \sim r\_k(X, Y))$ in HM

iff

$\forall X \in D_{HM} : \forall Y \in D_{HM} :$
$\sim(\sim(\bigvee_{i1=1}^{n1} p\_i1(X) \vee \bigvee_{i2=1}^{n2} \sim pp\_i2(X) \vee \bigvee_{i3=1}^{n3} s\_i3(X, a\_i3) \vee \bigvee_{i4=1}^{n4} \sim sp\_i4(X, a'\_i4))$
$\wedge \bigwedge_{j1=1}^{m1} \sim q\_j1(Y) \wedge \bigwedge_{j2=1}^{m2} qp\_j2(Y) \wedge \bigwedge_{j3=1}^{m3} \sim u\_j3(Y, b\_j3) \wedge \bigwedge_{j4=1}^{m4} up\_j4(Y, bp\_j4) \wedge$
$\bigwedge_{k=1}^{h} r\_k(X, Y))$ in HM

iff

$\sim(\exists X \in D_{HM} : \exists Y \in D_{HM} :$
$(\sim(\bigvee_{i1=1}^{n1} p\_i1(X) \vee \bigvee_{i2=1}^{n2} \sim pp\_i2(X) \vee \bigvee_{i3=1}^{n3} s\_i3(X, a\_i3) \vee \bigvee_{i4=1}^{n4} \sim sp\_i4(X, a'\_i4))$
$\wedge \bigwedge_{j1=1}^{m1} \sim q\_j1(Y) \wedge \bigwedge_{j2=1}^{m2} qp\_j2(Y) \wedge \bigwedge_{j3=1}^{m3} \sim u\_j3(Y, b\_j3) \wedge \bigwedge_{j4=1}^{m4} up\_j4(Y, bp\_j4) \wedge$
$\bigwedge_{k=1}^{h} r\_k(X, Y))$ in HM

iff

$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$(\sim(\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg pp\_i2)^I \vee \bigvee_{i3=1}^{n3} (x, a\_i3) \in s\_i3^I \vee \bigvee_{i4=1}^{n4} (x, a'\_i4) \notin sp\_i4^I) \wedge$
$\bigwedge_{j1=1}^{m1} y \in (\neg q\_j1)^I \wedge \bigwedge_{j2=1}^{m2} y \in qp\_j2 \wedge \bigwedge_{j3=1}^{m3} (y, b\_j3) \notin u\_j3^I \wedge \bigwedge_{j4=1}^{m4} (y, bp\_j4) \in up\_j4^I \wedge$
$\bigwedge_{k=1}^{h} (x, y) \in r\_k^I)$ in M

iff

$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$(\sim(\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg pp\_i2)^I \vee \bigvee_{i3=1}^{n3} x \in (!s\_i3(\_, a\_i3))^I \vee \bigvee_{i4=1}^{n4} x \in (\neg !sp\_i4(\_, a'\_i4))^I) \wedge$
$\bigwedge_{j1=1}^{m1} y \in (\neg q\_j1)^I \wedge \bigwedge_{j2=1}^{m2} y \in qp\_j2 \wedge \bigwedge_{j3=1}^{m3} y \notin (!u\_j3(\_, b\_j3))^I \wedge \bigwedge_{j4=1}^{m4} y \in (!up\_j4(\_, bp\_j4))^I \wedge$
$\bigwedge_{k=1}^{h} (x, y) \in r\_k^I)$ in M

iff

$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$(\sim(x \in (\bigcup_{i1=1}^{n1} p\_i1^I \cup \bigcup_{i2=1}^{n2} (\neg pp\_i2)^I \cup \bigcup_{i3=1}^{n3} (!s\_i3(\_, a\_i3))^I \cup$
$\bigcup_{i4=1}^{n4} (\neg !sp_{i4}(\_, a'\_i4)))^I)) \wedge$

$(y \in (\bigcap_{j1=1}^{m1}(\neg q\_j1)^I \cap \bigcap_{j2=1}^{m2} y \in qp\_j2 \cap \bigcap_{j3=1}^{m3}(\neg!u\_j3(\_, b\_j3))^I \cap$
$\bigcap_{j4=1}^{m4}(!up\_j4(\_, bp\_j4)))^I)) \wedge \bigwedge_{k=1}^{h}(x,y) \in r\_k^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$(\sim(x \in (\bigsqcup_{i1=1} n1p\_i1 \sqcup \bigsqcup_{i2=1}^{n2}(\neg pp\_i2) \sqcup \bigsqcup_{i3=1}^{n3}!s\_i3(\_, a\_i3) \sqcup$
$\bigsqcup_{i4=1}^{m4}(\neg!sp\_i4(\_, a'\_i4))))^I)) \wedge$
$y \in (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3}(\neg!u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4}!up\_j4(\_, bp\_j4))^I \wedge$
$\bigwedge_{k=1}^{h}(x,y) \in r\_k^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I : \exists y \in \Delta_M^I :$
$(\sim(x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3}!s\_i3(\_, a\_i3) \sqcup$
$\bigsqcup_{i4=1}^{m4}(\neg!sp_{i4}(\_, a'\_i4))))^I)) \wedge$
$\bigwedge_{k=1}^{h}(y \in (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3}(\neg!u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4}!up\_j4(\_, bp\_j4))^I \wedge (x,y) \in r\_k^I))$ in M
iff
$\sim(\exists x \in \Delta_M^I :$
$(\sim(x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3}!s\_i3(\_, a\_i3) \sqcup$
$\bigsqcup_{i4=1}^{m4}(\neg!sp_{i4}(\_, a'\_i4))))^I)) \wedge$
$\bigwedge_{k=1}^{h} x \in (?r\_k(\_, (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3}(\neg!u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4}!up\_j4(\_, bp\_j4))))^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I :$
$(\sim(x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3}!s\_i3(\_, a\_i3) \sqcup$
$\bigsqcup_{i4=1}^{m4}(\neg!sp_{i4}(\_, a'\_i4))))^I)) \wedge$
$x \in \bigcap_{k=1}^{h}(?r\_k(\_, (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3}(\neg!u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4}!up\_j4(\_, bp\_j4))))^I)$ in M
iff
$\sim(\exists x \in \Delta_M^I :$
$(\sim(x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3}!s\_i3(\_, a\_i3) \sqcup$
$\bigsqcup_{i4=1}^{m4}(\neg!sp_{i4}(\_, a'\_i4)))^I)) \wedge$
$x \in (\bigsqcap_{k=1}^{h}?r\_k(\_, (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3}(\neg!u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4}!up\_j4(\_, bp\_j4))))^I)$ in M
iff
$\forall x \in \Delta_M^I :$
$\sim(\sim(x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3}!s\_i3(\_, a\_i3) \sqcup$
$\bigsqcup_{i4=1}^{m4}(\neg!sp_{i4}(\_, a'\_i4))^I) \wedge$
$x \in (\bigsqcap_{k=1}^{h}?r\_k(\_, (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3}(\neg!u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4}!up\_j4(\_, bp\_j4))))^I)$ in M
iff
$\forall x \in \Delta_M^I :$
$(x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3}!s\_i3(\_, a\_i3) \sqcup \bigsqcup_{i4=1}^{n4}(\neg!sp_{i4}(\_, a'\_i4))^I \vee$

$x \notin (\bigsqcap_k 1h?r\_k(\_, (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3} (\neg !u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4} !up\_j4(\_, bp\_j4))))^I)$ in M
iff
$\forall x \in \Delta_M^I$:
$x \in (\bigsqcap_k 1h?r\_k(\_, (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap \bigsqcap_{j3=1}^{m3} (\neg !u\_j3(\_, b\_j3)) \sqcap$
$\bigsqcap_{j4=1}^{m4} !up\_j4(\_, bp\_j4))))^I \Rightarrow$
$(x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3} !s\_i3(\_, a\_i3) \sqcup \bigsqcup_{i4=1}^{n4} (\neg !sp_{i4}(\_, a'\_i4))^I$ in M
iff
$(\bigsqcap_k 1h?r\_k(\_, (\bigsqcap_{j1=1}^{m1} \neg q\_j1 \sqcap \bigsqcap_{j2=1}^{m2} qp\_j2 \sqcap$
$\bigsqcap_{j3=1}^{m3} (\neg !u\_j3(\_, b\_j3)) \sqcap \bigsqcap_{j4=1}^{m4} !up\_j4(\_, bp\_j4))))^I \subseteq$
$(\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} -pp\_i2 \sqcup \bigsqcup_{i3=1}^{n3} !s\_i3(\_, a\_i3) \sqcup \bigsqcup_{i4=1}^{n4} (\neg !sp_{i4}(\_, a'\_i4))^I$ in M
iff
`( **`$_{k=1}^{h}$`?r_k(_, ** ( **`$_{j1=1}^{m1}$`-q_j1 **  **`$_{j2=1}^{m2}$`qp_j2 **`
**`$_{j3=1}^{m3}$`-(!u_j3(_,b_j3)) ** **`$_{j4=1}^{m4}$`!up_j4(_,bp_j4)))) <<`
`(++`$_{i1=1}^{n1}$`p_i1 ++  ++`$_{i2=1}^{n2}$`-pp_i2 ++`
`++`$_{i3=1}^{n3}$`!s_i3(_,a_i3) ++  ++`$_{i4=1}^{n4}$`-(!sp_i4(_,a'_i4)))` in M $\Box$.

Lemma 4.20 Translating `r(X,X)` $\xmapsto{saffron}$ {`$reflexive(r)`}:

Proof
`r(X,X)` in HM
iff
$\forall X \in D_{HM} : r(X, X)$ in HM
iff
$\forall x \in \Delta_M^I : (x, x) \in r^I$ in M
iff
`$reflexive(r)` in M $\Box$.

Lemma 4.21 Translating `~r(X,X)` $\xmapsto{saffron}$ {`$irreflexive(r)`}:

Proof
`~r(X,X)` in HM
iff
$\forall X \in D_{HM} :\sim r(X, X)$ in HM
iff
$\forall x \in \Delta_M^I : (x, x) \notin r^I$ in M
iff
`$irreflexive(r)` in M $\Box$.

Lemma 4.22 Translating `(~r(X,Y) | r(Y,X))` $\stackrel{saffron}{\longmapsto}$ {`$symmetric(r)`}:

Proof

`(~r(X,Y) | r(Y,X))`
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (r(X,Y) \Rightarrow r(Y,X))$ in HM
iff
$\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : ((x,y) \in r^I \Rightarrow (y,x) \in r^I)$ in M
iff
`$symmetric(r)` in M $\Box$.

Lemma 4.23 Translating `(~r(X,Y) | ~r(Y,X))` $\stackrel{saffron}{\longmapsto}$ {`$asymmetric(r)`}:

Proof

`(~r(X,Y) | ~r(Y,X))` in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (r(X,Y) \Rightarrow \sim r(Y,X))$ in HM
iff
$\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : ((x,y) \in r^I \Rightarrow (y,x) \notin r^I)$ in M
iff
`$asymmetric(r)` in M $\Box$.

Lemma 4.24 Translating `(~r(X,Y) | ~r(Y,Z) | r(X,Z))` $\stackrel{saffron}{\longmapsto}$ {`$transitive(r)`}:

Proof

`(~r(X,Y) | ~r(Y,Z) | r(X,Z))`
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : \forall Z \in D_{HM} : (((r(X,Y) \wedge r(Y,Z)) \Rightarrow r(X,Z))$ in HM
iff
$\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : \forall z \in \Delta_M^I : (((x,y) \in r^I \wedge (y,z) \in r^I) \Rightarrow (x,z) \in r^I)$ in M
iff
`$transitive(r)` in M $\Box$.

Lemma 4.25 Translating `(~r(X,Y) | s(X,Y))` $\stackrel{saffron}{\longmapsto}$ {`r << s`}:

Proof

`(~r(X,Y) | s(X,Y))` in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (r(X,Y) \Rightarrow s(X,Y))$ in HM

iff
$\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : ((x,y) \in r^I \Rightarrow (x,y) \in s^I)$
iff
$r^I \subseteq s^I$ in M
iff
`r << s` in M $\square$.

Lemma 4.26 Translating `(~r(X,Y) | s(Y,X))` $\stackrel{saffron}{\longmapsto}$ {`inv_s -= s, r << inv_s`}:

Proof
`(~r(X,Y) | s(Y,X))` in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : (r(X,Y) \Rightarrow s(Y,X))$ in HM
iff
$\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : ((x,y) \in r^I \Rightarrow (y,x) \in s^I)$ in M.

$inv\_s = s^- \wedge \forall x \in \Delta_M^I : \forall y \in \Delta_M^I : ((x,y) \in r^I \Rightarrow (x,y) \in inv\_s^I)$ in M
iff
$inv\_s = s^- \wedge r^I \subseteq inv\_s^I$ in M
iff
{`inv_s -= s, r << inv_s`} in M $\square$.

Lemma 4.27 Translating `~r1(X,Y) | ~r2(Y,Z) | s(X,Z)` $\stackrel{saffron}{\longmapsto}$

{ `s >> r1 @ r2` }:

Proof
`~r1(X,Y) | ~r2(Y,Z) | s(X,Z)` in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : \forall Z \in D_{HM} :\sim r1(X,Y) \vee \sim r2(Y,Z) \vee s(X,Z)$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : \forall Z \in D_{HM} :\sim (r1(X,Y) \wedge r2(Y,Z)) \vee s(X,Z)$ in HM
iff
$\forall X \in D_{HM} : \forall Y \in D_{HM} : \forall Z \in D_{HM} : ((r1(X,Y) \wedge r2(Y,Z)) \Rightarrow s(X,Z))$ in HM
iff
$\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : \forall z \in \Delta_M^I : (((x,y) \in r1^I \wedge (y,z) \in r2^I) \Rightarrow (x,z) \in s^I)$in M
iff
$\forall x \in \Delta_M^I : \forall z \in \Delta_M^I : ((x,z) \in (r1 \circ r2)^I \Rightarrow (x,z) \in s^I)$ in M

iff

$(r1 \circ r2)^I \subseteq s^I$ in M

iff

`s >> r1@ r2` in M $\Box$.

Lemma 4.28 Translating `~r1(X,Y) | ~r2(X,Z) | s(Y,Z)` $\stackrel{saffron}{\longmapsto}$

{ `inv_s -= s, inv_r2 -= r2, inv_s >> inv_r2 @ r1` }:

Proof

`~r1(X,Y) | ~r2(X,Z) | s(Y,Z)`

iff

$\forall X \in D_{HM} : \forall Y \in D_{HM} : \forall Z \in D_{HM} : \sim r1(X,Y) \mid \sim r2(X,Z) \mid s(Y,Z)$ in HM

iff

$\forall X \in D_{HM} : \forall Y \in D_{HM} : \forall Z \in D_{HM} : (r1(X,Y) \wedge r2(X,Z)) \Rightarrow s(Y,Z)$ in HM

iff

$\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : \forall z \in \Delta_M^I : (((x,y) \in r1^I \wedge (x,z) \in r2^I) \Rightarrow (y,z) \in s^I)$ in M.

$inv\_s = s^- \wedge inv\_r2 = r2^- \wedge$

$(\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : \forall z \in \Delta_M^I : (((x,y) \in r1^I \wedge (z,x) \in inv\_r2^I) \Rightarrow (z,y) \in inv\_s^I))$ in M

iff

$inv\_s = s^- \wedge inv\_r2 = r2^- \wedge$

$(\forall x \in \Delta_M^I : \forall y \in \Delta_M^I : \forall z \in \Delta_M^I : ((z,y) \in (inv\_r2 \circ r1)^I \Rightarrow (z,y) \in inv\_s^I))$ in M

iff

$inv\_s = s^- \wedge inv\_r2 = r2^- \wedge (inv\_r2 \circ r1)^I \subseteq inv\_s^I$ in M

iff

{`inv_s -= s, inv_r2 -= r2, inv_s >> inv_r2 @ r1`} in M $\Box$.

Lemma 4.29 Translating $|_{i=1}^n$`p_i(X)` | $|_{i=1}^m$`~q_i(X)` $\stackrel{saffron}{\longmapsto}$

{ `&thing =` $+\!\!+_{i=1}^n$`p_i ++` $+\!\!+_{i=1}^m$`~q_i` } ($n \geq 0, m \geq 0$, and $m + n \geq 2$):

Proof

$|_{i=1}^n$`p_i(X) | `$|_{i=1}^m$`~q_i(X)` in HM

iff

$\forall X \in D_{HM} : (\bigvee_{i=1}^n p\_i(X) \vee \bigvee_{i=1}^m \sim q_i(X))$ in HM

iff

$\forall x \in \Delta_M^I : (\bigvee_{i=1}^n (x \in p\_i^I) \vee \bigvee_{i=1}^m (x \notin q\_i))$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i=1}^n (x \in p\_i^I) \vee \bigvee_{i=1}^m (x \in (\Delta_M^I \backslash q\_i^I))$ in M
iff
$\forall x \in \Delta_M^I : (\bigvee_{i=1}^n (x \in p\_i^I) \vee \bigvee_{i=1}^m (x \in (\neg q\_i)^I))$ in M
iff
$\forall x \in \Delta_M^I : x \in (\bigcup_{i=1}^n (p\_i^I) \cup \bigcup_{i=1}^m (\neg q\_i)^I)$ in M
iff
$\forall x \in \Delta_M^I : x \in (\bigsqcup_{i=1}^n p\_i \sqcup \bigsqcup_{i=1}^m (\neg q\_i))^I$ in M
iff
$(\bigsqcup_{i=1}^n p\_i \sqcup \bigsqcup_{i=1}^m (\neg q\_i))^I = \Delta_M^I = \top^I$ in M
iff
`&thing =` $+\!\!+_{i=1}^n$`p_i ++` $+\!\!+_{i=1}^m$`~q_i` in M $\square$.

Lemma 4.30 Translating $|_{i=1}^n$`(X = a_i(X))` $\stackrel{saffron}{\longmapsto}$ { `&thing = [a_1,a_2,...,a_n]` }

($n \geq 0, m \geq 0$, and $m + n \geq 2$):

Proof
$|_{i=1}^n$`(X = a_i(X))` in HM
iff
$\forall X \in D_{HM} : (\bigvee_{i=1}^n (X = a\_i))$ in HM
iff
$\forall x \in \Delta_M^I : (\bigvee_{i=1}^n (x = a\_i^I))$ in M
iff
$\forall x \in \Delta_M^I : x \in \{a\_1^I, a\_2^I, ..., a\_n^I\}$ in M
iff
$\forall x \in \Delta_M^I : x \in \{a\_1, a\_2, ..., a\_n\}^I$ in M
iff
$\{a\_1, a\_2, ..., a\_n\}^I = \Delta_M^I = \top^I$ in M
iff
`&thing = [a_1,a_2,...,a_n]` in M $\square$.

Lemma 4.31 Translating $|_{i=1}^n$`r(X,a_i(X))` $\stackrel{saffron}{\longmapsto}$ { `&thing = ?r(_,[a_1,a_2,...,a_n])` }

($n \geq 0, m \geq 0$, and $m + n \geq 2$):

Proof
$|_{i=1}^n$`r(X,a_i(X))` in HM
iff
$\forall X \in D_{HM} : (\bigvee_{i=1}^n r(X, a\_i))$ in HM
iff
$\forall x \in \Delta_M^I : (\bigvee_{i=1}^n (x, a\_i^I) \in r^I)$ in M

iff

$\forall x \in \Delta_M^I : \exists y \in \Delta_M^I : ((x,y) \in r^I \wedge y \in \{a\_1^I, a\_2^I, ..., a\_n^I\})$ in M

iff

$\forall x \in \Delta_M^I : \exists y \in \Delta_M^I : ((x,y) \in r^I \wedge y \in \{a\_1, a\_2, ..., a\_n\}^I)$ in M

iff

$\forall x \in \Delta_M^I : x \in (\exists r.\{a\_1, a\_2, ..., a\_n\})^I$ in M

iff

$(\exists r.\{a\_1, a\_2, ..., a\_n\})^I = \Delta_M^I = \top^I$ in M

iff

`&thing = ?r(_,[a_1,a_2,...,a_n])` in M $\square$.

Lemma 4.32 Translating $|_{i1=1}^{m1}$`p_i1(X) |` $|_{i2=1}^{m2}$ `~q_i2(X) |` $|_{j=1}^{n}$`(X = a_j)` $\overset{saffron}{\longmapsto}$

{`&thing = (` $+\!\!+_{i1=1}^{m1}$`p_i1 ++` $+\!\!+_{i2=1}^{m2}$`-q_i2 ++ [a_1,a_2,...,a_n])`}

where $(m1 + m2) \geq 1$, and $n \geq 1$,

Proof

$|_{i1=1}^{m1}$`p_i1(X) |` $|_{i2=1}^{m2}$ `~q_i2(X) |` $|_{j=1}^{n}$`(X = a_j)` in HM

iff

$\forall X \in D_{HM} : (\bigvee_{i1=1}^{m1} p\_i1(X) \vee \bigvee_{i2=1}^{m2} \sim q\_i2(X) \vee \bigvee_{j=1}^{n} (X = a\_j))$ in HM

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \notin q\_i2^I \vee \bigvee_{j=1}^{n} (x = a\_j^I))$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\Delta_M^I \backslash q\_i2^I) \vee \bigvee_{j=1}^{n} (x = a\_j^I))$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee \bigvee_{j=1}^{n} (x = a\_j^I))$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee x \in \{a\_1, a\_2, ..., a\_n\}^I)$ in M

iff

$\forall x \in \Delta_M^I : x \in (\bigcup_{i1=1}^{n1} p\_i1^I \cup \bigcup_{i2=1}^{n2} (\neg q\_i2)^I \cup \{a\_1, a\_2, ..., a\_n\}^I)$ in M

iff

$\forall x \in \Delta_M^I : x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} (\neg q\_i2) \sqcup \{a\_1, a\_2, ..., a\_n\})^I$ in M

iff

$(\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} (\neg q\_i2) \sqcup \{a\_1, a\_2, ..., a\_n\})^I = \Delta_M^I = \top^I$ in M

iff

`&thing = (` $+\!\!+_{i1=1}^{m1}$`p_i1 ++` $+\!\!+_{i2=1}^{m2}$`-q_i2 ++ [a_1,a_2,...,a_n])` $\square$.

Lemma 4.33 Translating $|_{i1=1}^{m1}$`p_i1(X)` | $|_{i2=1}^{m2}$`~q_i2(X)` | $|_{j1=1}^{n1}$`r_j1(X,a_j1)` | $|_{j2=1}^{n2}$`~r_j2(X,b_j2)` $\stackrel{saffron}{\longmapsto}$

{`&thing = (` $+\!\!+_{i1=1}^{m1}$`p_i1 ++` $+\!\!+_{i2=1}^{m2}$`-q_i2 ++` $+\!\!+_{j1=1}^{n1}$`!r_j1(_,a_j1) ++` $+\!\!+_{j2=1}^{n2}$`-(! r_j2(_,b_j2))` }

where $(m1 + m2) \geq 0$, and $(n1 + n2) \geq 1$

Proof

$|_{i1=1}^{m1}$`p_i1(X)` | $|_{i2=1}^{m2}$`~q_i2(X)` | $|_{j1=1}^{n1}$`r_j1(X,a_j1)` | $|_{j2=1}^{n2}$`~r_j2(X,b_j2)`

iff

$\forall X \in D_{HM} : (\bigvee_{i1=1}^{m1} p\_i1(X) \vee \bigvee_{i2=1}^{m2} \sim q\_i2(X) \vee \bigvee_{j1=1}^{n1} r\_j1(X, a\_j1) \vee \bigvee_{j2=1}^{n2} \sim r\_j2(X, b\_j2))$ in HM

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \notin q\_i2^I \vee \bigvee_{j1=1}^{n}(x, a\_j1^I) \in r\_j1^I \vee \bigvee_{j2=1}^{n}(x, b\_j2^I) \notin r\_j2^I)$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\Delta_M^I \backslash q\_i2^I) \vee \bigvee_{j1=1}^{n}(x, a\_j1^I) \in r\_j1^I \vee \bigvee_{j2=1}^{n}(x, b\_j2^I) \notin r\_j2^I)$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee \bigvee_{j1=1}^{n}(x, a\_j1^I) \in r\_j1^I \vee \bigvee_{j2=1}^{n}(x, b\_j2^I) \notin r\_j2^I)$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee \bigvee_{j1=1}^{n}(x, a\_j1^I) \in r\_j1^I \vee \sim \bigwedge_{j2=1}^{n}(x, b\_j2^I) \in r\_j2^I)$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee \bigvee_{j1=1}^{n} x \in (\forall r\_j1.a\_j1)^I \vee \sim \bigwedge_{j2=1}^{n} x \in (\forall r\_j2.b\_j2)^I)$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee \bigvee_{j1=1}^{n} x \in (\forall r\_j1.a\_j1)^I \vee \bigvee_{j2=1}^{n} x \notin (\forall r\_j2.b\_j2)^I)$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee \bigvee_{j1=1}^{n} x \in (\forall r\_j1.a\_j1)^I \vee \bigvee_{j2=1}^{n} x \in (\Delta_M^I \backslash (\forall r\_j2.b\_j2)^I))$ in M

iff

$\forall x \in \Delta_M^I : (\bigvee_{i1=1}^{n1} x \in p\_i1^I \vee \bigvee_{i2=1}^{n2} x \in (\neg q\_i2)^I \vee \bigvee_{j1=1}^{n} x \in (\forall r\_j1.a\_j1)^I \vee \bigvee_{j2=1}^{n} x \in (\neg \forall r\_j2.b\_j2)^I)$ in M

iff

$\forall x \in \Delta_M^I : x \in (\bigcup_{i1=1}^{n1} p\_i1^I \cup \bigcup_{i2=1}^{n2} (\neg q\_i2)^I \cup \bigcup_{j1=1}^{n} (\forall r\_j1.a\_j1)^I \cup \bigcup_{j2=1}^{n} (\neg \forall r\_j2.b\_j2)^I)$ in M

iff

$\forall x \in \Delta_M^I : x \in (\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} (\neg q\_i2) \sqcup \bigsqcup_{j1=1}^{n} \forall r\_j1.a\_j1 \sqcup \bigsqcup_{j2=1}^{n} (\neg \forall r\_j2.b\_j2))^I$ in M

iff

$(\bigsqcup_{i1=1}^{n1} p\_i1 \sqcup \bigsqcup_{i2=1}^{n2} (\neg q\_i2) \sqcup \bigsqcup_{j1=1}^{n} \forall r\_j1.a\_j1 \sqcup \bigsqcup_{j2=1}^{n} (\neg \forall r\_j2.b\_j2))^I = \Delta_M^I = \top^I$ in M

iff

`&thing = (` $+\!\!+_{i1=1}^{m1}$`p_i1 ++` $+\!\!+_{i2=1}^{m2}$`-q_i2 ++` $+\!\!+_{j1=1}^{n1}$`!r_j1(_,a_j1) ++`
$+\!\!+_{j2=1}^{n2}$`-(!r_j2(_,b_j2))`

$\square$.

Since for each translation rule $c \xmapsto{saffron} saffron(c)$, it is proven that c is `TRUE` in HM iff $saffron(c)$ is `TRUE` in M. Thus, HM and M can be constructed one from another. Thus, the theorem is proven $\square$.

4.5 Testing the Implementation of `Saffron`

In addition to the mathematical proof of the soundness of the translation procedure provided in Section 4.4, an empirical test for the soundness of the implementation of the translation procedure, `Saffron`, has been performed. Initially, `Saffron` was tested over CNF problems created to test different features of the translation, and many problems in the TPTP libarary. Then it was tested over sets of sample problems in the TPTP library.

In order to test the soundness of the implementation of `Saffron`, a CNF problem, *problem*, is translated to DL, then the DL problem, $saffron(problem)$, is attempted using `Konclude`. The CNF problem, *problem*, might be the result of the translation of a problem expressed in a form more expressive than CNF. If a theorem or an unsatisfiable problem expressed in a form more expressive than CNF is translated to

CNF, the resultant CNF problem is unsatisfiable. If a non-theorem or a satisfiable problem expressed in a form more expressive than CNF is translated to CNF, the resultant CNF problem is satisfiable. The status of the translated CNF problem is the basis for the expected status of the translated DL problem.

Table 4.7 shows the status of the CNF problem *problem*, the translation percentage, the possible status of the DL problem $saffron(problem)$ confirmed by `Konclude`, and the soundness evaluation of the implementation of `Saffron` based on these values.

The soundness of `Saffron` was tested over four sets of sample problems from the TPTP library. These four sets are all DL-able problems, CNF **S**oft**W**are **V**erification (SWV) problems that are DL-able after applying the splitting technique, FOF problems with no functions of arity greater than zero, and TF0 **H**are**W**are **V**erification (HWV) problems. The time limit for each stage of the reasoning process was 300 seconds, and the experiment was done on a machine with the following specifications.

- Number of CPUs : 4
- CPU Model: Intel(R) Xeon(TM) CPU 2.80GHz
- RAM per CPU: 756MB
- OS: Linux 2.6.32.26-175.fc12.i686.PAE

Table 4.8 summarizes the results of the testing of the soundness. The column *Sample* indicates the set of sample problems. The column *Status* is either THM, UNS, non-THM, and SAT to categorize the set of sample problems into the theorem, unsatisfiable, non-theorem or satisfiable categories. The column *Translation %* is

Table 4.7: Evaluation of the Soundness of `Saffron`

Status of *problem*	**Translation Percentage**	**Status of** $saffron(problem)$	**Soundness Evaluation**
UNS	100%	UNS	Sound
		SAT	Unsound
	< 100%	UNS	Sound
		SAT	Inconclusive
SAT	100%	UNS	Unsound
		SAT	Sound
	< 100%	UNS	Unsound
		SAT	Inconclusive

the average CNF to DL translation percentage of (translated) CNF problems. The column *Total* is the total number of problems in the corresponding category. The column *Sound*, *Inconclusive*, and *Unsound* shows the number of problems, over which test results suggest soundness, are inconclusive, and suggest unsoundness respectively. None of the test results suggest that `Saffron` is unsound. All the test results either suggest the soundness of `Saffron`, or are inconclusive.

Table 4.8: Result of the Empirical Soundness Test of `Saffron`

Sample	**Status**	**Translation %**	**Total**	**Sound**	**Inconclusive**	**Unsound**
DL-able	UNS	100%	23	12	11	0
	SAT	100%	3	0	3	0
SWV	UNS	100%	40	32	8	0
FOF	THM/UNS	100%	7	7	0	0
		< 100%	128	5	123	0
	non-THM/SAT	100%	8	8	0	0
		< 100%	32	0	32	0
TF0 HWV	THM/UNS	100%	20	0	20	0
		< 100%	31	2	29	0
	non-THM/SAT	100%	6	1	5	0
		< 100%	6	0	6	0

4.5.1 DL-able Problems

The first set of problems was the DL-able problems in the TPTP library. This sample was chosen to test the soundness of `Saffron` without the effects of splitting, partial CNF to DL translation, and the translation from logics more expressive than CNF to CNF. There are 23 unsatisfiable and 3 satisfiable DL-able problems.

The unsatisfiability of 12 unsatisfiable problems was confirmed by `Konclude`. These results suggest that `Saffron` is sound. `Konclude` timed out on the remaining 11 unsatisfiable problems, and on all three satisfiable problems. These results are inconclusive.

4.5.2 CNF Software Verification Problems (SWV) Problems

The second set of problems was all the CNF SWV problems in the TPTP library that are DL-able after applying the splitting technique. This sample was chosen to test the soundness of `Saffron` with splitting, but without the effects of partial CNF to DL translation, and the translation from logics more expressive than CNF to CNF. This experiment also introduces a new industrial application of DL. There are 40 unsatisfiable such SWV problems. They all have only one proposition. The splitting technique produces two unsatisfiable proposition free versions of each problem. The problem is considered solved by translation to DL if the unsatisfiability of the both proposition free versions of the problem is confirmed by `Konclude`.

The unsatisfiability of both proposition free versions of 32 problems out of 40 problems was confirmed by `Konclude`. These results suggest that `Saffron` is sound.

`Konclude` timed out on at least one proposition free version of the remaining 8 problems. Theses results are inconclusive.

4.5.3 FOF Problems with no Functions of Arity Greater than Zero

The third set of problems was the FOF problems with no functions of arity greater than zero in three domains CSR, KRS, and NUM in the TPTP library. This sample was chosen to test the soundness of `Saffron` with the translation from a logic more expressive than CNF to CNF. Initial experiments showed that many translated CNF problem from such FOF problems in three domains CSR, KRS, and NUM are DL-able.

There are 175 FOF problems with no functions of arity greater than zero in three domains CSR, KRS, and NUM in the TPTP library. Sixty-one problems are theorems, seventy-four problems are unsatisfiable, one problem is a non-theorem, and thirty-nine problems are satisfiable. These 175 problem were successfully translated to CNF using the sound FOF to CNF translator, `ECNF`. The resultant 135 unsatisfiable and 40 satisfiable CNF problems are then translated to DL using `Saffron`.

Among the 135 unsatisfiable translated CNF problems, seven problems were DL-able. The unsatisfiability of all the DL-able problems, and five of the partially translated problems was confirmed by `Konclude`. These results suggest that `Saffron` is sound. `Konclude` either timed out on the remaining 128 problems, or confirmed the satisfiability of partially translated problems. These results are inconclusive.

Among the 40 satisfiable translated CNF problems, eight problems were DL-able. The satisfiability of all the DL-able problems was confirmed by `Konclude`. These results suggest that `Saffron` is sound. `Konclude` either timed out on the remaining 32 problems, or confirmed the satisfiability of the partially translated problems. These results are inconclusive.

4.5.4 TF0 Hardware Verification (HWV) Problems

The last set of problems was the TF0 HWV problems in the TPTP library. This sample was chosen to test the soundness of `Saffron` with the translation from logics more expressive than CNF to CNF.

There are 63 TF0 HWV problems. Thirty-one problems are theorem, 20 problems are unsatisfiable, six problems are non-theorem, and six problems are satisfiable. These 63 problem were successfully translated to CNF using the sound TF0 to CNF translator, `Monotonox-2CNF`. The resultant 51 unsatisfiable and 12 satisfiable CNF problems are then translated to DL using `Saffron`.

Among the 51 unsatisfiable translated CNF problems, 20 problems were DL-able. `Konclude` timed out on all 20 DL-able problems. These results are inconclusive. The unsatisfiability of two partially translated problems is confirmed by `Konclude`. These results suggest that `Saffron` is sound. `Konclude` either timed out on the remaining 29 partially translated problems, or confirmed the satisfiability of the partially translated problems. These results are inconclusive.

Among the 12 satisfiable translated CNF problems, six problems were DL-able. The satisfiability of one DL-able problems is confirmed by `Konclude`. This result suggests that `Saffron` is sound. `Konclude` timed out on the remaining 5 DL-able problems. These results are inconclusive. `Konclude` timed out on all partially translated problems. These results are inconclusive.

Chapter 5

Experiments

Different ATP systems and translators can be combined to solve a problem expressed in a given logical form. The related logical forms to this research are Propositional Logic(PL), Description Logic (DL), Effectively Propositional Form (EPR), Conjunctive Normal Form (CNF), First Order Form (FOF), Typed First order form-monomorphic (TF0), Typed First order form-polymorphic (TF1), Typed Higher order form-monomorphic (TH0).

As illustrated in the Figure 5.1, A problem, expressed in a logic, can be either solved using an ATP system for that logic, or translated to a less expressive logic. If it is translated to a less expressive logic, again the same two options of solving using an ATP system, and translating down (if possible) are available. This continues until no further translation is possible. In Figure 5.1, the plain arrows demonstrate the process of available translation, and the dashed arrows demonstrate the process of solving. Table 5.1 lists the translators and ATP systems used in the experiments of

this research, corresponding to the labels on the arrows in Figure 5.1. The selected ATP system for solving CNF problems is Vampire , but if the problem is EPR, the selected ATP system is iProver.

#	Action			Tool
1	TH0	→	Proof	Satallax 2.7
1.3	TH0	→	TF0	Isabelle-2TF0
1.4	TH0	→	FOF	Isabelle-2FOF
2	TF1	→	Proof	Alt-Ergo 0.95.2
2.3	TF1	→	Proof	Why3-TF0 0.85
2.4	TF1	→	Proof	Why3-FOF 0.85
3	TF0	→	Proof	CVC4 TFF-1.5
3.4	TF0	→	FOF	Monotonox-2FOF e3c1636
3.5	TF0	→	CNF	Monotonox-2CNF e3c1636
4	FOF	→	Proof	Vampire 4.0
4.5	FOF	→	CNF	ECNF 1.8
5	CNF	→	Proof	Vampire 4.0 or iProver 1.4
5.6	CNF	→	DL	Saffron 4.5
5.7	EPR	→	PL	EGround 1.8
6	DL	→	Proof	Konclude 0.6.1
7	PL	→	Proof	MiniSAT 2.2.0

Table 5.1: ATP Systems and Translators used in this research

An experiment was designed and carried out to compare the effectiveness of solving problems in different logics (TH0, TF1, TF0, FOF, CNF and EPR) using all possible combinations of available translators and ATP systems. The comparison was done in terms of the number of problems solved and the average CPU time taken for the solved problems, for the whole reasoning process, including translations and solving.

The source of problems for the experiment was the TPTP library. The sample problems in TH0, FOF, CNF and EPR were the TPTP problems selected for the latest CADE ATP System Competitions that have related divisions to each logic. The machine used to run this experiment has the following specifications.

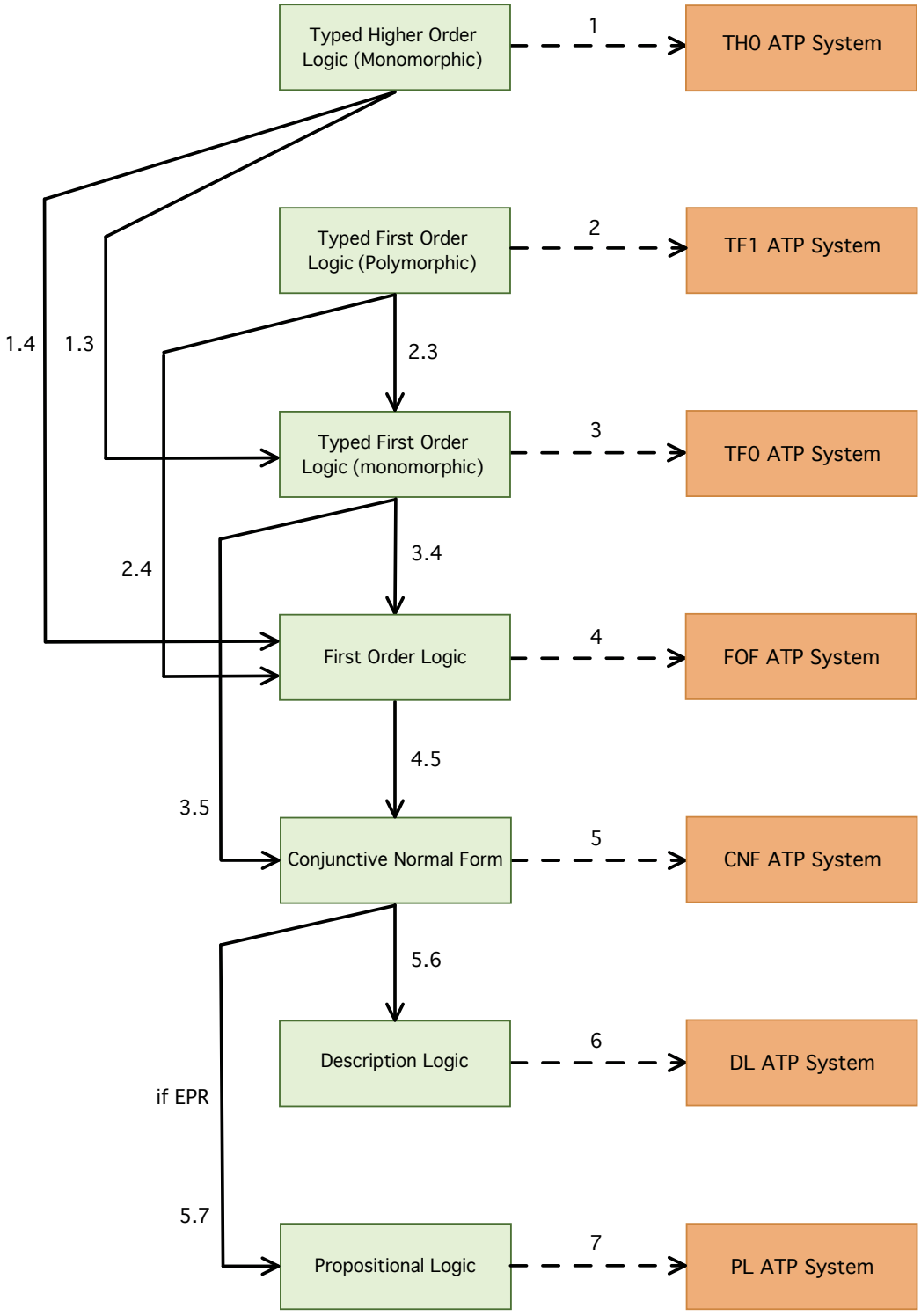

Figure 5.1: Combinations of translators and ATP systems

- Number of CPUs : 4
- CPU Model: Intel(R) Xeon(TM) CPU 2.80GHz
- RAM per CPU: 756MB
- OS: Linux 2.6.32.26-175.fc12.i686.PAE

Notation. A *path* written $logic_1 \cdot logic_2 \cdot \ldots \cdot logic_n$, with a chain of tools (translators and an ATP system), $translator_1 > translator_2 > \ldots > translator_{n-1} > atp_n$, means that the problem is originally in logic $logic_1$, and for $1 \leq i \leq n-1$, problems in the logic $logic_i$ are translated to the logic $logic_{i+1}$ using the translator $translator_i$, and eventually the problem in the logic $logic_n$ is attempted using its ATP system atp_n.

The results of the experiments over each set of problems are discussed in the following sections of this chapter. The results of each set of problems are displayed in result tables, and include *Total Available*, *Success*, *Time*, *Common Time*, *Translation Time*, *Solving Time*, *Timeout* and *Failed* for each path. "Total Available" is the number of problems available for the final translation step (if any) of the path. "Success" is the number of problems solved, and the percentage with respect to "Total Available". "Time" is the average CPU time for the whole reasoning process for solved problems. "Common Time" is the average CPU time for the common problems solved through all paths up to and including the row. "Translation Time" is the average CPU time for the translation steps for solved problems. "Solving Time" is the average CPU time for the ATP system to solve the (translated) problems. "Timeout" is

the number of problems for which the last translation step or the ATP system timed out, and the percentage with respect to "Total Available". "Failed" is the number of problems for which the last translation step or the ATP system failed, and the percentage with respect to "Total Available".

5.1 Effectively Propositional Form

EPR problems can be translated to DL and PL. As explained in Chapter 4, an EPR problem might be only partially translated to DL. However, if the unsatisfiable core of an unsatisfiable EPR problem is translated, the partially translated problem can be used to confirm the unsatisfiability using a DL ATP system.

Three experiments were carried out over three different sets of EPR problems to compare the possible paths for EPR problems. One set is the problems from the CASC-J7 EPR division. The two other sets are the sets of EPR problems that are used in Section 4.5.1 and 4.5.2 in Chapter 4 to test the soundness of the implementation of Saffron.

Table 5.2 shows the results of the experiment on the 200 EPR problems from CASC-J7. As illustrated in this table, iProver can solve more problems than EGround > MiniSAT, which in turn can solve more problems than Saffron > Konclude. The reason that Saffron > Konclude cannot solve more problems than the two other alternatives is that only 20 of the EPR problems are DL-able. An average of 29% of the clauses of each non-DL-able problem were translated to DL.

Table 5.2: Results of Comparing Paths over 200 EPR Problems

Path	Tools	Success	Time	Common Time	Translation Time	Solving Time	Timeout	Failed
EPR	iProver	180 (90%)	22.50	22.50	0.00	22.50	19 (10%)	1 (1%)
EPR.PL	EGround > MiniSAT	56 (28%)	0.26	0.26	10.63	0.07	113 (57%)	31 (16%)
EPR.DL	Saffron > Konclude	4 (2%)	57.26	76.74	8.79	55.42	59 (30%)	137 (69%)

Table 5.3 shows the results of the experiment on the set of 26 DL-able problems used to test the soundness of Saffron in Section 4.5.1 of Chapter 4. Since all of these problems are EPR, iProver is used to solve the problems. As illustrated in this table, iProver can solve more problems than EGround > MiniSAT, which in turn can solve more problems than Saffron > Konclude. The results of the experiment suggest that solving an EPR problem by Saffron > Konclude is an alternative, despite the fact that iProver is a well-tuned and powerful ATP system for EPR problems. The average time for the translation of problems by Saffron is 0.43, but Konclude timed out with the time limit of 300.

Table 5.3: Results of Comparing Paths from EPR over 26 DL-able Problems

Path	Tools	Success	Time	Common Time	Translation Time	Solving Time	Timeout	Failed
EPR	iProver	26 (100%)	34.76	34.76	0.00	34.76	0 (0%)	0 (0%)
EPR.PL	EGround > MiniSAT	13 (50%)	1.40	1.40	0.00	1.40	0 (0%)	13 (50%)
EPR.DL	Saffron > Konclude	12 (46%)	0.54	0.54	0.00	0.54	14 (54%)	0 (0%)

Table 5.4 shows the results of the experiment on the TPTP 40 EPR **S**oft**W**are **V**erification (SWV) problems that are DL-able after removing propositions using the splitting technique. This set was used to test the soundness of Saffron in Sec-

tion 4.5.2 of Chapter 4. All these SWV problems are unsatisfiable and have only one proposition. The splitting technique generates two proposition free versions of these problems. If the unsatisfiability of both versions of each problem is confirmed by Konclude, then the problem is solved through the EPR.DL path. The two versions of the problem can be solved in parallel if more than one CPU is available. In Table 5.4, EPR.DL(serial) and EPR.DL(parallel) show the statistics for solving the two versions of SWV problems in serial and parallel through the path EPR.DL. The success rate of solving by translation to DL is 80%. This result suggests that solving by translation to DL is an alternative way of solving EPR SWV problems.

Table 5.4: Results of Comparing Paths from EPR over 40 EPR SWV Problems

Path	**Tools**	**Success**	**Time**	**Common Time**	**Translation Time**	**Solving Time**	**Timeout**	**Failed**
EPR	iProver	40 (100%)	13.87	13.87	0.00	13.87	0 (0%)	0 (0%)
EPR.DL (parallel)	Saffron > Konclude	32 (80%)	22.00	22.00	2.11	19.89	8 (20%)	0 (0%)
EPR.DL (serial)	Saffron > Konclude	32 (80%)	29.92	29.92	4.21	25.71	8 (20%)	0 (0%)
EPR.PL	EGround > MiniSAT	3 (8%)	0.38	-	0.27	0.12	24 (60%)	13 (33%)

5.2 Conjunctive Normal Form

CNF problems can be translated to DL, and if they are EPR they can be translated to PL. The same as EPR problems, partial translation of CNF problems might be useful.

The problems in CNF are the problems from the CASC-J6 CNF division. Table 5.5 shows the results of the experiment on the set of 199 unsatisfiable CNF problems from CASC-J6. None of these problems are EPR, so none of them can be translated to PL. The translation of all of these CNF problems to DL was partial, with an average of 14% of the clauses of each problem were 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 Konclude.

Table 5.5: Results of Comparing Paths from CNF over 199 Problems from CASC-J6

Path	**Tools**	**Success**	**Time**	**Timeout**	**Failed**
CNF	Vampire	186 (93%)	13.30	13 (7%)	0 (0%)
CNF.DL	Saffron >Konclude	0 (0%)	-	0 (0%)	199 (100%)

5.3 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 PL, as explained in Section 5.2.

The problems in FOF are the problems from the CSC-J7 FOF division. Table 5.6 shows the results of the experiment. As expected from the results of CASC competitions, Vampire is a very powerful ATP system for FOF. This experiment also shows that Vampire solves more problems than any other approach.

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 5.7 shows which path is the best to be

Table 5.6: Results of Comparing Paths from FOF

Path	Tools	Total Available	Success	Time	Common Time	Translation Time	Solving Time	Timeout	Failed
FOF	Vampire	400	376 (94%)	14.01	14.01	0.00	14.01	22 (6%)	2 (1%)
FOF.CNF	ECNF > Vampire or iProver	400	321 (80%)	20.30	19.85	20.26	0.04	63 (16%)	16 (4%)
FOF.CNF.DL	ECNF > Saffron > Konclude	390	12 (3%)	98.75	0.13	95.89	2.86	56 (14%)	322 (83%)
FOF.CNF.PL	ECNF > EGround > MiniSAT	8	0 (0%)	-	-	-	-	4 (50%)	4 (50%)

added to maximize the number of problems solved when increasing the number of CPUs. Out of the 400 sample problems, 379 problems can be solved through all paths together. The two paths shown in Table 5.7 can solve all these problems, so using more than two CPUs will not increase the chance of a problem being solved.

Table 5.7: Paths from FOF in Parallel

CPUs	Path	Exclusive	Success	Time
1	FOF	376	376	14.00
2	FOF.CNF	3	379	10.41

5.4 Typed First-order Form - monomorphic

TF0 problems can be translated to FOF or CNF. If a problem is translated to FOF, then it can be translated to CNF as explained in Section 5.3. If a problem is translated to CNF, then it can be translated to DL or PL, as explained in Section 5.2.

The problems in TF0 are all the TF0 problems without arithmetic in the TPTP. Table 5.8 shows the results of the experiment. As expected from the results of CASC, CVC4 is a very powerful ATP system for TF0. This experiment also shows that CVC4 solves more problems than any other approach. The results of this experiment also

suggests that TF0 problems are very hard because less than half of the problems are solved with the best approach. The two translations to DL solve 4% and 2% of the total available problems.

Table 5.8: Results of Comparing Paths from TF0

Path	Tools	Total Available	Success	Time	Common Time	Translation Time	Solving Time	Timeout	Failed
TF0	CVC4	153	70 (46%)	18.42	18.42	0.00	18.42	83 (54%)	0 (0%)
TF0.FOF	Monotonox-2FOF > Vampire	153	61 (40%)	11.52	8.87	0.13	11.39	55 (36%)	37 (24%)
TF0.FOF.CNF	Monotonox-2FOF > ECNF > Vampire or iProver	128	56 (44%)	21.24	21.64	0.10	21.15	61 (48%)	11 (9%)
TF0.CNF	Monotonox-2CNF > Vampire or iProver	153	52 (34%)	13.66	5.48	0.00	13.65	43 (28%)	58 (38%)
TF0.CNF.DL	Monotonox-2CNF > Saffron > Konclude	98	4 (4%)	14.94	14.94	2.92	12.03	37 (38%)	57 (58%)
TF0.CNF.PL	Monotonox-2CNF > EGround > MiniSAT	45	3 (7%)	0.00	-	0.00	0.00	42 (93%)	0 (0%)
TF0.FOF.CNF.PL	Monotonox-2FOF > ECNF > EGround > MiniSAT	60	3 (5%)	0.00	-	0.00	0.00	52 (87%)	5 (8%)
TF0.FOF.CNF.DL	Monotonox-2FOF > ECNF > Saffron > Konclude	126	2 (2%)	82.11	-	11.47	70.64	40 (32%)	84 (67%)

Table 5.9 shows which path is the best to be added to maximize the number of problems solved when increasing the number of CPUs. Out of the 153 problems, 82 problems can be solved through all paths together. The four paths shown in Table 5.9 can solve all these problems, so using more than four CPUs will not increase the chance of a problem being solved. It shows that using multiple CPUs significantly increases the chance of a problem being solved. The best approach using CVC4 solves 70 problems out of 153. Using the four approaches in Table 5.9 solves 82 problems, which is an over 17% increase in the number of problems solved.

Table 5.9: Paths from TF0 in Parallel

CPUs	Path	Exclusive	Success	Time
1	TF0	70	70	18.42
2	TF0.FOF.CNF	9	79	11.23
3	TF0.CNF	2	81	10.55
4	TF0.FOF	1	82	10.24

5.5 Typed First-order Form - polymorphic

TF1 problems can be translated to TF0 or FOF. If a problem is translated to TF0, then it can be translated to FOF or CNF as explained in Section 5.4. If a problem is translated to FOF, then it can be translated to CNF, as explained in Section 5.3.

The problems in TF1 are all the TF1 problems without arithmetic in the TPTP that were available when this experiment was initiated. Table 5.10 shows the results of the experiment. The results suggest that solving by translation down to FOF or CNF solves more problem than than using the direct ATP system for TF1. The reason is that Vampire, the ATP systems is a very powerful ATP system for FOF and CNF. The TF1 problems are mostly very hard problems, and the best approach can solve 44% of the problems within the time limit of 300 seconds. The best translation to DL approach, through the path TF1.TF0.CNF.DL, can solve 3% of the problems. The other two translation to DL approaches can solve 2% and 1% of the problems. All the translated problems to CNF are partially translated to DL.

Table 5.11 shows which path is the best to be added to maximize the number of problems solved when increasing the number of CPUs. Out of the 538 problems, 300 problems can be solved through all paths together. The five paths shown in Table 5.11 can solve all these problems, so using more than five CPUs will not increase the chance

Table 5.10: Results of Comparing Paths from TF1

Path	Tools	Total Available	Success	Time	Common Time	Translation Time	Solving Time	Timeout	Failed
TF1.FOF	Why3-FOF > Vampire	538	238 (44%)	9.34	9.34	0.08	9.26	288 (54%)	12 (2%)
TF1.TF0.FOF	Why3-TF0 > Monotonox-2FOF > Vampire	538	234 (43%)	2.95	3.77	0.12	2.82	225 (42%)	79 (15%)
TF1.TF0.FOF.CNF	Why3-TF0 > Monotonox-2FOF > ECNF > Vampire or iProver	495	233 (47%)	7.72	8.47	0.19	7.52	225 (45%)	37 (7%)
TF1.FOF.CNF	Why3-FOF > ECNF > Vampire or iProver	538	231 (43%)	7.51	3.70	0.09	7.42	303 (56%)	4 (1%)
TF1.TF0.CNF	Why3-TF0 > Monotonox-2CNF > Vampire or iProver	538	208 (39%)	7.48	5.82	0.14	7.35	257 (48%)	73 (14%)
TF1	Alt-Ergo	538	197 (37%)	0.79	0.25	0.00	0.79	158 (29%)	183 (34%)
TF1.TF0	Why3-TF0 > CVC4	538	184 (34%)	11.00	5.09	0.25	10.75	295 (55%)	59 (11%)
TF1.TF0.CNF.DL	Why3-TF0 > Monotonox-2CNF > Saffron > Konclude	486	14 (3%)	0.68	0.71	0.11	0.56	0 (0%)	472 (97%)
TF1.TF0.FOF.CNF.DL	Why3-TF0 > Monotonox-2FOF > ECNF > Saffron > Konclude	495	8 (2%)	0.71	0.71	0.15	0.56	0 (0%)	487 (98%)
TF1.FOF.CNF.DL	Why3-FOF > ECNF > Saffron > Konclude	538	3 (1%)	0.66	0.67	0.12	0.54	0 (0%)	535 (99%)

of a problem being solved. It shows that using multiple CPUs significantly increases the chance of a problem being solved. The best approach using Why3-FOF > Vampire solves 238 problems out of 538. Using the five approaches in Table 5.11 solves 300 problems, which is an over 26% increase in the number of problems solved. The two translation to DL approaches can exclusively solve 2% of the problems. This result is interesting since the TF1 ATP system cannot solve any problems exclusively.

Table 5.11: Paths from TF1 in Parallel

CPUs	Path	Exclusive	Success	Time
1	TF1.FOF	238	238	9.34
2	TF1.TF0.FOF.CNF	55	293	7.18
3	TF1.TF0.CNF.DL	4	297	7.03
4	TF1.FOF.CNF	2	299	7.47
5	TF1.FOF.CNF.DL	1	300	7.45

5.6 Typed Higher-order From - monomorphic

TH0 problems can be translated to TF0 and FOF. If a problem is translated to TF0 then it can be translated to FOF or CNF, as explained in Section 5.4. If a problem is translated to FOF then it can be translated to CNF, as explained in Section 5.3.

The problems in TH0 are the problems from the CSC-J7 TH0 division. Table 5.12 shows the results of the experiment on this set of TH0 problems. As expected from the results of the CASC competitions, Satallax is a very powerful ATP system for TH0. This experiment also shows that Satallax solves more problems than any other approaches.

Table 5.13 shows which path is the best to be added to maximize the number of problems solved when increasing the number of CPUs. Out of the 400 problems, 367 problems can be solved through all paths together. The four paths shown in Table 5.13 can solve all these problems, so using more than four CPUs will not increase the chance of a problem being solved. It shows that using multiple CPUs significantly increases the chance of a problem being solved. The best approach using Satallax solves 354 problems out of 153. Using the four approaches in Table 5.13 solves 367 problems, which is an almost 4% increase in the number of problems solved.

Table 5.12: Results of Comparing Paths from TH0

Path	Tools	Total Available	Success	Time	Common Time	Translation Time	Solving Time	Timeout	Failed
TH0	Satallax	400	354 (89%)	6.94	6.94	0.00	6.94	46 (12%)	0 (0%)
TH0.FOF	Isabelle-2FOF > Vampire	400	148 (37%)	10.22	10.17	7.07	3.15	111 (28%)	141 (35%)
TH0.FOF.CNF	Isabelle-2FOF > ECNF > Vampire or iProver	400	146 (37%)	13.40	12.10	7.07	6.33	119 (30%)	135 (34%)
TH0.TF0	Isabelle-2TF0 > CVC4	400	136 (34%)	11.30	9.46	6.94	4.36	212 (53%)	52 (13%)
TH0.TF0.FOF.CNF	Isabelle-2TF0 > Monotonox-2FOF > ECNF > Vampire or iProver	400	111 (28%)	12.75	7.27	6.65	6.10	118 (30%)	171 (43%)
TH0.TF0.FOF	Isabelle-2TF0 > Monotonox-2FOF > Vampire	397	108 (27%)	8.46	7.12	6.65	1.81	114 (29%)	175 (44%)
TH0.TF0.CNF	Isabelle-2TF0 > Monotonox-2CNF > Vampire or iProver	397	105 (26%)	8.89	8.30	6.83	2.07	115 (29%)	177 (45%)
TH0.TF0.CNF.PL	Isabelle-2TF0 > Monotonox-2CNF > EGround > MiniSAT	12	12 (100%)	6.75	6.75	6.75	0.00	0 (0%)	0 (0%)
TH0.TF0.FOF.CNF.PL	Isabelle-2TF0 > Monotonox-2FOF > ECNF > EGround > MiniSAT	11	11 (100%)	6.75	6.75	6.75	0.00	0 (0%)	0 (0%)
TH0.FOF.CNF.PL	Isabelle-2FOF > ECNF > EGround > MiniSAT	10	10 (100%)	6.78	6.79	6.78	0.00	0 (0%)	0 (0%)
TH0.TF0.CNF.DL	Isabelle-2TF0 > Monotonox-2CNF > Saffron > Konclude	400	3 (1%)	7.77	7.32	7.23	0.55	1 (0%)	396 (99%)
TH0.FOF.CNF.DL	Isabelle-2FOF > ECNF > Saffron > Konclude	399	1 (0%)	7.31	7.31	6.81	0.50	2 (1%)	396 (99%)
TH0.TF0.FOF.CNF.DL	Isabelle-2TF0 > Monotonox-2FOF > ECNF > Saffron > Konclude	371	1 (0%)	7.34	7.34	6.82	0.52	1 (0%)	369 (99%)

Table 5.13: Paths from TH0 in Parallel

CPUs	Path	Exclusive	Success	Time
1	TH0	354	354	6.94
2	TH0.FOF	10	364	5.60
3	TH0.TF0.FOF.CNF	2	366	5.91
4	TH0.TF0	1	367	5.92

Chapter 6

Conclusion

6.1 Review of the Dissertation

Chapter 1 provides an introduction to the research and related issues, to build up a basis for understanding the dissertation. This includes an introduction to automated theorem proving, a description of the TPTP world, an introduction to related conferences and competitions, and the research goals.

Chapter 2 provides a survey of the logics related to the research. The logics, in the increasing order of expressivity, are Propositional Logic, Description Logic, Effectively Propositional Form, Conjunctive Normal Form, First Order Form, Typed First order form-monomorphic, Typed First order form-polymorphic, Typed Higher order form-monomorphic. For each logic, the syntax and semantics are briefly explained, and common Automated Theorem Proving (ATP) systems, and translators to less expressive logics are introduced.

Chapter 3 describes DLF, a new TPTP syntax for Description Logic. This includes a description of the TPTP problems, and of the DLF annotated formulae (definitions, type formulae, and logic formulae).

Chapter 4 describes a translation procedure from CNF to DL, and its implementation as `Saffron`. This includes a mathematical proof for the soundness of the translation procedure, and empirical tests of the soundness of the implementation of `Saffron`.

Chapter 5 describes an experiment to evaluate different ways of solving problems in target logical forms of this research. This includes analysis and discussion of the results of the experiment over sample test problems in each logical form.

6.2 Contributions and Conclusions

In this research, `Saffron`, a CNF to DL translator has been developed. No such translator was available before. The translation procedure is a mapping of CNF clause structures to DL formulae with equivalent semantics. A proof of the soundness of the translation procedure has been provided. `Saffron` is the implementation of the translation procedure in Prolog. `Saffron` can 100% translate many EPR problems with only constants, unary predicates and binary predicates, many EPR Software Verification problems, many FOF problems with no functions of arity grater than zero translated to CNF, and many TF0 Hardware Verification problems translated to CNF. The translation of EPR Software Verification problems, and CNF Hardware Verification problems has introduced new industrial applications of DL. An empirical

test for the soundness of the implementation has been provided. The empirical test suggests that the implementation is sound.

DLF, a new TPTP syntax for Description Logic, has been designed in this research. DL is more expressive than PL, and less expressive than EPR. DL, as a specific logic for expressing ontologies, was not a part of TPTP world, and there was no TPTP DL syntax before. A BNF definition has been provided to allow parsing DLF problems. Now `Saffron` and the DLF syntax can help embed DL into the TPTP world, so that the DL communities can benefit from the TPTP and TSTP tools.

Experiments have been designed to compare different ways of solving problems by using combinations of translators and ATP systems. This included identifying state-of-the-art ATP systems and translators for all logics, as well as identifying appropriate sets of test problems. A framework was then built for running experiments over all the possible ways of solving the test problems. The execution of the experiments and the analysis of their results revealed that, in general, solving problems by translation to less expressive logics, particularly DL, is an alternative (some times the only) way of solving a problem.

The results of the experiments show that more TF1 problems are solved by translations to less expressive logics than by using a TF1 ATP system. The results also show that solving TF0 problems by translations to less expressive logics is an alternative way of solving problems versus using a TF0 ATP system. Moreover, the results show that solving DL-able and some EPR Software Verification problems by translation to DL is an alternative way of solving problems versus using an EPR ATP system. Almost half of such Software Verification problems are solved by translation to DL

almost as fast as solving by an established and powerful EPR ATP system. Many FOF problems with no functions of arity greater than zero are solved by translation to DL, which offers an alternative way of solving such problems.

When more than one CPU is available, simultaneously solving TF0 and TF1 problems through several paths remarkably increases the chance of the problem being solved, compared to the best approach for solving these problems. Solving TF1 problems by translation to DL with two possible combinations of translators and a DL ATP system can exclusively solve 2% of the TF1 sample problems. This result is interesting since the TF1 ATP system cannot solve any problems exclusively.

6.3 Future Work

In the future, the translation procedure and the implementation of `Saffron` can be extended to translate CNF clauses with wider range of clause structures. Examples of potential clause structures are clauses with unary functions and clauses that specify the domain and the range of binary predicates. Variable instantiation can be added as a preprocessing feature of `Saffron` in a way that the number of variables in a clause is reduced, so that the clause can be translated using the currently available features of `Saffron`.

In the future, DLF and DLF tools can be integrated into the TPTP world. This is possible by developing tools for importing problems in other DL syntaxes into the TPTP in the DLF syntax, and adding features to `TPTP2X` for exporting problems in other DL syntaxes. DL problems can be collected and released in the TPTP problem

library. Solutions for DL problems can be collected and released in the TSTP solution library. Eventually, DL ATP systems and tools can be added to the `SystemOnTPTP`.

Appendix A

The BNF for DLF

This appendix is the BNF for DLF, TPTP syntax for DL SROIQ. Table A.1 describes the separators for different rules reduction [48].

Table A.1: Reduction Separators in TPTP BNF

Rule Type	Separator
Syntactic	::=
Semantic	:==
Lexical	::-
Character-macro	:::

```
%----------------------------------------------------------------------
<dlf_annotated> ::= dlf(<name>,<formula_role>,<dlf_formula><annotations>).
<annotations> ::= ,<source><optional_info> | <null>
%----------------------------------------------------------------------
%----DLF formulae
<dlf_formula> ::= <dlf_typed_atom> | <dlf_definition> |
    <dlf_declaration_atom> | <dlf_indiv_formula> |
    <dlf_role_formula> | ( <dlf_formula> )

%----------------------------------------------------------------------
%----DLF declaration formula which includes:
%----A class, individual or role is equal or not equla to another one.
%----A class or role is subclass(subrole) of or disjoint with another one.
<dlf_declaration_atom> ::= <dlf_block_name> <infix_equality>
    <dlf_class_unitary_term> | <dlf_block_name> <infix_inequality>
```

```
    <dlf_class_unitary_term> | <dlf_block_name> <dlf_disjoint_operator>
    <dlf_class_unitary_term> | <dlf_block_name> <subtype_sign>
    <dlf_class_unitary_term>

<dlf_class_unitary_term> ::= <dlf_block_name> | <dlf_closed_class> |
    <dlf_restriction_term> |
    <dlf_negation_operator> <dlf_class_unitary_term> |
    ( <dlf_class_binary_term> ) | ( <dlf_class_unitary_term> )

%----All dlf binary operators over classes are associative
%----All associative operators are left assosiative by default, unless
% parantesized differently
<dlf_class_binary_term> ::= <dlf_class_union_term> |
    <dlf_class_intersection_term> | <dlf_class_disjoint_union_term>
<dlf_class_union_term> ::= <dlf_class_unitary_term> <dlf_union_operator>
    <dlf_class_unitary_term> | <dlf_class_union_term> <dlf_union_operator>
    <dlf_class_unitary_term>
<dlf_class_intersection_term> ::= <dlf_class_unitary_term>
    <dlf_intersection_operator> <dlf_class_unitary_term> |
    <dlf_class_intersection_term> <dlf_intersection_operator>
    <dlf_class_unitary_term>
<dlf_class_disjoint_union_term> ::= <dlf_class_unitary_term>
    <dlf_disjoint_union_operator> <dlf_class_unitary_term> |
    <dlf_class_disjoint_union_term> <dlf_disjoint_union_operator>
    <dlf_class_unitary_term>

<dlf_closed_class> ::= [<dlf_block_list>]
<dlf_restriction_term> ::= <fol_quantifier>
    <dlf_block_name>(<underscore>,<dlf_class_unitary_term>)

%-----------------------------------------------------------------------
%----DLF individual formula
<dlf_indiv_formula> ::= <dlf_block_name>(<dlf_block_name>,<dlf_block_name>)|
    <dlf_negation_operator><dlf_block_name>(<dlf_block_name>,<dlf_block_name>)

%----DLF individual formula
<dlf_indiv_formula> ::= <dlf_block_name>(<dlf_block_name>,<dlf_block_name>)|
    <dlf_negation_operator><dlf_block_name>(<dlf_block_name>,<dlf_block_name>)

%-----------------------------------------------------------------------
%----DLF role formula
<dlf_role_formula> ::= <dlf_role_characteristic>(<dlf_block_name>) |
    <dlf_role_description_one_on_one>(<dlf_block_name>,<dlf_block_name>) |
    <dlf_block_name> <supertype_sign> <dlf_roles_chain> |
    <dlf_block_name> <dlf_role_infix_one_on_one_operator> <dlf_block_name>

<dlf_role_characteristic> ::= $functional | $inverse_functional |
```

```
    $transitive | $symmetric | $asymmetric | $reflexive | $irreflexive

<dlf_role_description_one_on_one> ::= $equivalentProperty |
    $subPropertyOf | $propertyDisjointWith | $inverseOf

<dlf_role_infix_one_on_one_operator> ::= <dlf_infix_inverse_of>

%----A role is superProperty of a role or a chain of roles
<dlf_roles_chain> ::= <dlf_block_name> |
    <dlf_roles_chain> <dlf_infix_role_chain> <dlf_block_name> |
    ( <dlf_roles_chain> )

%-----------------------------------------------------------------------
%----DLF typed atom
<dlf_typed_atom> ::= <dlf_block_name> :  <dlf_top_level_type>

<dlf_top_level_type> ::= $tType | <dlf_class_unitary_term> |
    $role(<dlf_class_unitary_term>,<dlf_class_unitary_term>) |
    ( <dlf_class_unitary_term> <star> <dlf_class_unitary_term> )
    <arrow> <dlf_oType>

<dlf_definition> ::= <dlf_block_name> <dlf_defn_operator> <dlf_top_level_type>
<dlf_definition> :== <atomic_word> <dlf_defn_operator> <dlf_top_level_type>

%----DLF auxiliary definitions
%----dlf blocks are classes, individuals, and roles
<dlf_block_name> ::= <atomic_word> | <atomic_defined_word> |
    <ampersand_word> | <ampersand_defined_word> |
    <ampersand_word> <dlf_delimiter> <atomic_word> |
    <ampersand_defined_word> <dlf_delimiter> <atomic_word> |
    <ampersand_word> <dlf_delimiter> |
    <ampersand_defined_word> <dlf_delimiter>
<dlf_block_list> ::= <dlf_block_name> | <dlf_block_list>,<dlf_block_name>

%-----------------------------------------------------------------------
%----DLF operators
%--- unary
<dlf_negation_operator> ::= ~

%--- binary and left associative
<dlf_infix_role_chain> ::= @

%--- binary and left associative
<dlf_infix_inverse_of> ::= <minus>=

%--- binary and left associative
<dlf_union_operator> ::= ++
```

```
%--- binary and left associative
<dlf_intersection_operator> ::= <star><star>

%--- binary and left associative
<dlf_disjoint_operator> ::= <less_sign><arrow>

%--- binary and nonassociative
<dlf_defn_operator> ::= <infix_equality><infix_equality>

<dlf_delimiter> ::= ; | <hash_symbol> | /
<dlf_oType> ::= <atomic_defined_word>
<dlf_oType> :== $o
<dlf_disjoint_union_operator> ::= <less_sign><plus><arrow>

%----------------------------------------------------------------------
```

Appendix B

Pseudo Code for Saffron

```
List<Clause> notTranslatedClauses = new ArrayList<Clause>();
List<Characteristic> characteristics = new ArrayList<Characteristic>();
public void translate(String inputFile, String outputFile){
    List<Clause> clauses = readCNFClause(inputFile);
    for(Clause clause: clauses){
        if (!translateOneClause(clause)){
            notTranslatedClauses.add(clause);
        }
    }
    List<DLAxiom> dlAxioms = gather();
    generateOutput(dlAxioms,outputFile);
}
/**
 * reads the TPTP CNF inputFile and returns its list of clauses
 * @param inputFile in
 * @return list of clauses in inputFile
 */
List<Clause> readCNFClause(String inputFile){
   return null;
}
/**
 * Translate the clause into characteristics for an individual,
 * a class or a role, adds the characteristics to the list of
 * characteristics.
 * @param clause
 * @return true if succeeds to translate clause, and false if
 * fails to translate the clause because it is not DL-able.
 */
boolean translateOneClause(Clause clause){
    List<String> positiveIndividuals = new ArrayList<String>();
    List<String> negativeIndividuals = new ArrayList<String>();
    List<String> positiveClasses = new ArrayList<String>();
    List<String> negativeClasses = new ArrayList<String>();
    List<String> positiveRoles = new ArrayList<String>();
    List<String> negativeRoles = new ArrayList<String>();

    extract(clause.literals,INDIVIDUAL,positiveIndividuals,negativeIndividuals);
```

```
        extract(clause.literals,CLASS,positiveClasses,negativeClasses);
        extract(clause.literals,ROLE,positiveRoles,negativeRoles);

        return guessClause(clause.literals, positiveIndividuals,
                negativeIndividuals, positiveClasses, negativeClasses,
                positiveRoles, negativeRoles);
    }
    /**
     * Based on the list of constants, unary predicates, and binary
     * predicates of the clause, and their polarity, and list of
     * variables in the literals of the clause, guesses the translation
     * and if it finds a translation, it will be added to the list of
     * characteristics.
     * @param literals
     * @param positiveIndividuals list of constants appearing in positive
     *                            literals in literals
     * @param negativeIndividuals list of constants appearing in negative
     *                            literals in literals
     * @param positiveClasses list of positive unary predicates appearing
     *                        in literals
     * @param negativeClasses list of negative unary predicates appearing
     *                        in literals
     * @param positiveRoles list of positive binary predicates appearing
     *                      in literals
     * @param negativeRoles list of negative binary predicates appearing
     *                      in literals
     * @return true if a translation for the clause is found, and
     *         false if no translation for the clause is found
     */
    boolean guessClause(List<String> literals,
                    List<String> positiveIndividuals,
                    List<String> negativeIndividuals,
                    List<String> positiveClasses,
                    List<String> negativeClasses,
                    List<String> positiveRoles,
                    List<String> negativeRoles){
        List<String> targets = checkCharacteristics(literals,
                positiveIndividuals, negativeIndividuals, positiveClasses,
                negativeClasses, positiveRoles, negativeRoles);
        if(targets.isEmpty())
            return false;
        for(String individual: positiveIndividuals){
            if(!targets.contains(individual))
                characteristics.add(new
                    Characteristic(INDIVIDUAL,individual,""));
        }
        for(String individual: negativeIndividuals){
            if(!targets.contains(individual))
                characteristics.add(new
                    Characteristic(INDIVIDUAL,individual,""));
        }
        for(String oneClass: positiveClasses){
            if(!targets.contains(oneClass))
                characteristics.add(new Characteristic(CLASS,oneClass,""));
        }
        for(String oneClass: negativeClasses){
            if(!targets.contains(oneClass))
                characteristics.add(new Characteristic(CLASS,oneClass,""));
        }
```

```
    for(String role: positiveRoles){
        if(!targets.contains(role))
            characteristics.add(new Characteristic(ROLE,role,""));
    }
    for(String role: positiveRoles){
        if(!targets.contains(role))
            characteristics.add(new Characteristic(ROLE,role,""));
    }
    return true;
}
/**
 * For each individual, class, and role, gathers all the
 * characteristics that are created during the translation
 * to form one RDF/XML tag for each.
 * @return the list of all RDF/XML tags
 */
List<DLAxiom> gather(){

    Map<String,List<String>> individualsTags = new HashMap<String,
        List<String>>();
    Map<String,List<String>> classesTags = new HashMap<String,
        List<String>>();
    Map<String,List<String>> rolesTags = new HashMap<String,
        List<String>>();
    for(Characteristic characteristic: characteristics){
        List<String> tags;
        if(characteristic.blockType == INDIVIDUAL){
          tags = individualsTags.get(characteristic.name);
          if(tags == null){
              tags = new ArrayList<String>();
          }
          tags.add(characteristic.subTag);
          individualsTags.put(characteristic.name,tags);
        }
        else if(characteristic.blockType == CLASS){
            tags = classesTags.get(characteristic.name);
            if(tags == null){
                tags = new ArrayList<String>();
            }
            tags.add(characteristic.subTag);
            classesTags.put(characteristic.name,tags);
        }
        else {
            tags = rolesTags.get(characteristic.name);
            if(tags == null){
                tags = new ArrayList<String>();
            }
            tags.add(characteristic.subTag);
            rolesTags.put(characteristic.name,tags);
        }
    }
    List<DLAxiom> dlAxioms = new ArrayList<DLAxiom>();
    for(String individual: individualsTags.keySet()){
       dlAxioms.add(new
          DLAxiom(INDIVIDUAL,individual,individualsTags.get(individual)));
    }
    for(String oneClass: classesTags.keySet()){
        dlAxioms.add(new
            DLAxiom(CLASS,oneClass,classesTags.get(oneClass)));
```

```
    }
    for(String role: rolesTags.keySet()){
        dlAxioms.add(new DLAxiom(ROLE,role,rolesTags.get(role)));
    }
    return dlAxioms;
}
/**
 * Writes header, the dl axioms and the footer in outputFile
 * @param dlAxioms the DL axioms gathered after the translation
 * @param outputFile the owl file including the DL translation
 *                   of the input file
 */

void generateOutput(List<DLAxiom> dlAxioms,String outputFile){
  //create output file named outputFile
  //write header of the ontology to outputFile
  //write dlAxioms to outputFile
  //write the number of untranslated CNF as a comment
  //write the untranslated CNF formulas as a comment
  //write footer of including the translation result
}

/**
 * Extracts both lists of positive and negative either
 * constants(Individuals), unary pedicates (classes),
 * or binary predicates (roles) from literals
 * @param literals
 * @param blockType is set to either INDIVIDUAL, CLASS or ROLE
 * @param positiveBlocks will be set to the list of positive constants
 *                      appearing in literals if the blockType is
 *                      INDIVIDUAL, or the list of the name of positive
 *                      unary predicates appearing in literals if the
 *                      blockType is Class, or the list of the name of
 *                      positive binary predicates appearing in literals
 *                      if the blockType is ROLE.
 * @param negativeBlocks will be set to the list of negative constants
 *                      appearing in literals if the blockType is
 *                      INDIVIDUAL, or the list of the name of negative
 *                      unary predicates appearing in literals if the
 *                      blockType is Class, or the list of the name of
 *                      negative binary predicates appearing in literals
 *                      if the blockType is ROLE.
 */
void extract(List<String> literals, int blockType,
            List<String> positiveBlocks, List<String> negativeBlocks){
    ...
}
/**
 * Based of the Tables 4.5 and 4.6  Checks if
 * the clause is translated to characteristics of one or more literals
    in
 * literals or the default class Thing
 * @param literals
 * @param positiveIndividuals list of constants appearing in positive
 *                           literals in literals
 * @param negativeIndividuals list of constants appearing in negative
 *                           literals in literals
 * @param positiveClasses list of positive unary predicates appearing
 *                       in literals
```

```
     * @param negativeClasses list of negative unary predicates appearing
     *                        in literals
     * @param positiveRoles list of positive binary predicates appearing
     *                      in literals
     * @param negativeRoles list of negative binary predicates appearing
     *                      in literals
     * @return the list of characteristics found
     */
    List<String> checkCharacteristics(List<String> literals,
                    List<String> positiveIndividuals, List<String>
                       negativeIndividuals,
                    List<String> positiveClasses, List<String>
                       negativeClasses,
                    List<String> positiveRoles,List<String> negativeRoles){
        ...
    }
```

References

[1] G. Sutcliffe, "The SZS Ontologies for Automated Reasoning Software," in *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, ser. CEUR Workshop Proceedings, G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, Eds., no. 418, 2008, pp. 38–49.

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

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

[4] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *Proceedings of the 23rd International Conference on Computer Aided Verification*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., no. 6806. Springer-Verlag, 2011, pp. 171–177.

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

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

[7] K. Claessen, A. Lillieström, 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. Springer, 2011, pp. 207–221.

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

[9] ——, "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.

[10] ——, "SystemOnTPTP," in *Proceedings of the 17th International Conference on Automated Deduction*, ser. Lecture Notes in Artificial Intelligence, D. McAllester, Ed., no. 1831. Springer-Verlag, 2000, pp. 406–410.

[11] The International Joint Conference on Automated Reasoning. http://www.ijcar.org. [Online]. Available: http://www.ijcar.org

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

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

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

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

[16] DIMACS, "Satisfiability Suggested Format."

[17] 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.

[18] 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.

[19] SAT-Race 2008 Results. http://baldur.iti.uka.de/sat-race-2008/results.html. [Online]. Available: http://baldur.iti.uka.de/sat-race-2008/results.html

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

[21] I. Horrocks, "Applications of description logics: State of the art and research challenges," in *Conceptual Structures: Common Semantics for Sharing Knowledge*, ser. Lecture Notes in Computer Science, F. Dau, M.-L. Mugnier, and G. Stumme, Eds. Springer Berlin Heidelberg, 2005, vol. 3596, pp. 78–90. [Online]. Available: http://dx.doi.org/10.1007/11524564_5

[22] 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.

[23] F. Baader, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003. [Online]. Available: https://books.google.com/books?id=riSeOKw5I6sC

[24] A. Steigmiller, T. Liebig, and B. Glimm, "Konclude: System description," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 0, no. 0, 2014. [Online]. Available: http://www.websemanticsjournal.org/index.php/ps/article/view/366

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

[26] R. Shearer, B. Motik, and I. Horrocks, "HermiT: A Highly-Efficient OWL Reasoner." in *OWLED*, ser. CEUR Workshop Proceedings, C. Dolbear, A. Ruttenberg, and U. Sattler, Eds., vol. 432. CEUR-WS.org, 2008. [Online]. Available: http://dblp.uni-trier.de/db/conf/owled/owled2008.html#ShearerMH08

[27] The OWL Reasoner Evaluation Workshop. http://curation.cs.manchester.ac.uk/ore2013/ore2013.cs.manchester.ac.uk/index.html. [Online]. Available: http://curation.cs.manchester.ac.uk/ore2013/ore2013.cs.manchester.ac.uk/index.html

[28] The OWL Reasoner Evaluation Competition 2014. http://www.easychair.org/smart-program/VSL2014/ORE-competition.html. [Online]. Available: http://www.easychair.org/smart-program/VSL2014/ORE-competition.html

[29] L. Kovacs and A. Voronkov, "First-Order Theorem Proving and Vampire," in *Proceedings of the 25th International Conference on Computer Aided Verification*, ser. Lecture Notes in Artificial Intelligence, N. Sharygina and H. Veith, Eds., no. 8044. Springer-Verlag, 2013, pp. 1–35.

[30] 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.

[31] G. Sutcliffe, "The 7th IJCAR Automated Theorem Proving System Competition - CASC-J7," *AI Communications*, vol. 28, no. 4, pp. 683–692, 2015.

[32] J. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the ACM*, vol. 12, no. 1, pp. 23–41, 1965.

[33] 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*, ser. Lecture Notes in Artificial Intelligence, P. Baumgartner, A. Armando, and D. Gilles, Eds., no. 5195, 2008, pp. 292–298.

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

[35] 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.

[36] 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. Springer-Verlag, 2008, pp. 274–289.

[37] 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*. Springer-Verlag, 1994, pp. 341–355.

[38] J. Blanchette and A. Paskevich, "Tff1: The tptp typed first-order form with rank-1 polymorphism," in *Automated Deduction CADE-24*, ser. Lecture Notes in Computer Science, M. Bonacina, Ed. Springer Berlin Heidelberg, 2013, vol. 7898, pp. 414–420. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38574-2_29

[39] 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*, ser. Lecture Notes in Computer Science, B. Gramlich, D. Miller, and U. Sattler, Eds., vol. 7364. Manchester, UK: Springer, Jun. 2012, pp. 67–81.

[40] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich, "Why3: Shepherd your herd of provers," in *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wrocław, Poland, August 2011, pp. 53–64. [Online]. Available: http://proval.lri.fr/publications/boogie11final.pdf

[41] W. M. Farmer, "The seven virtues of simple type theory," *Journal of Applied Logic*, vol. 6, no. 3, pp. 267 – 286, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157086830700081X

[42] J. Vnnen, "Second-order logic and foundations of mathematics," *Bulletin of Symbolic Logic*, vol. 7, pp. 504–520, 12 2001. [Online]. Available: http://journals.cambridge.org/article_S1079898600005370

[43] L. Henkin, "Completeness in the theory of types," *The Journal of Symbolic Logic*, vol. 15, no. 2, pp. 81–91, 1950. [Online]. Available: http://www.jstor.org/stable/2266967

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

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

[46] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[47] 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.

[48] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder, "Using the TPTP Language for Writing Derivations and Finite Interpretations," in *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, ser. Lecture Notes in Artificial Intelligence, U. Furbach and N. Shankar, Eds., no. 4130, 2006, pp. 67–81.