Faculty of Computer Science, Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
Tel: +39 04710 16000, fax: +39 04710 16009, http://www.inf.unibz.it/krdb/

*KRDB Research Centre Technical Report:*

# Techniques for query rewriting based on interpolation

Nhung Ngo

| | |
|---|---|
| **Affiliation** | KRDB Research Center, FUB |
| **Corresponding author** | Ngo Thi Phuong Nhung maxnhong85@gmail.com |
| **Keywords** | query rewriting, interpolant, defiability, interpolation |
| **Number** | KRDB10-3 |
| **Date** | 04-08-2010 |
| **URL** | http://www.inf.unibz.it/krdb/ |

# Acknowledgments

*First, I would like to give my great thanks to my supervisor Prof. Enrico Franconi at the KRDB Research Center for his kind supervision of this thesis. Without his support and advices, this research would have never been possible.*

*My special gratitude also goes to the EMCL consortium for the two year Erasmus Mundus scholarship which gives me not only a chance to study under an advance education but also a great time to enjoy Europe.*

*I want to thank all of my friends in Vienna and Bolzano, who are always beside me and have made my two years in this master program become one of the most beautiful periods in my life. I would like to thank my colleges in Free University of Bolzano also, in particular Evgeny and Olga for many useful discussions and advices. I wish to send my special thanks as well to my closed friends in Vietnam for keeping our friendships over the Internet.*

*Last but not least, my deepest gratitude goes to my beloved family, my mothers and fathers, my husband, my brothers for always encouraging me, taking care of me and understanding me.*

# Contents

# List of Figures

# List of Tables

# Abstract

Query rewriting is an essential technique for query processing over database systems. In general, query rewriting can be understood as a method to reformulate a query into another form to make the query processing more convenient in the sense of computing costs, giving expected answers and so on. Among many query rewriting approaches, in this master thesis, we will study a recently proposed rewriting framework [26] [4] based on Craig's interpolant [9] and Beth's definability theorem. According to the framework, a query can be rewritten using the language corresponding to a set of database predicates implicitly defining the query. In the other words, from the proof of a valid formula implied by implicitly definability of a query, we can construct a Craig's interpolant which reflects the rewritten query. Related to a fundamental characteristic of a query, the framework requires the domain independence of the rewriting. Based on these researches, we shall first try to get a rewritten query in a syntactic restriction of domain independence named safe range, which is the crucial criteria to transform a first order query to a SQL query. Besides, we will focus on some methods to construct intepolants from proofs generated by automated theorem provers. At the last step, which is also the goal of our work, a query rewriting tool that automatically performs the rewriting process will be implemented on top of a famous theorem prover named Prover9.

Chapter 1

# Introduction

Nowadays, with the help of many database techniques from organizing, storing data to query processing, people have had chances to query over more and more complicated system and get expected answers in a short time. In terms of query answering and query optimization, query rewriting is not only one of the most efficient approaches but also a very challenging one.

Looking at many researches and practical works in database and knowledge representation systems, one can see query rewriting has been applied almost everywhere. In database management system, a query can be rewritten into a new form using less complicated operations, removing duplicated works or using offline-computed views in order to speed up query answering. By means of integrated systems, we need to reformulate a query by a set of views corresponding to the set of data sources. In semantic-driven systems, users can query at semantic level while data is stored in a relational database, then to answer, systems should interpret the query into data level. Therefore, query rewriting is a general technique. In the other words, depending on purposes, people can rewrite a query in various ways.

Considering methods for query rewriting based on the form of rewritten queries, we have two types : syntactic rewriting where the rewriting process changes the syntactic form of the query and language based rewriting where the languages of the original query and rewritten query are different. In this master thesis, we shall do research on a method belonging to the second type. Intuitively, given a query which is written in the language $A$, a language $B$ and a knowledge base over the language $A \cup B$, based on the method, we shall first determine whether one can use the language $B$ to rewrite the query or not. If the answer is 'yes', we show the rewritten query. More precisely, this method contains two following steps :

- Prove the ability of rewriting using implicit definability of the query over the language $B$.
- If the query is implicitly definable, compute the rewritten query based on Craig's interpolant of valid implication derived by implicitly definability.

## 1.1 Motivations

Let us consider the first motivation for the mentioned method with a simple example shown below. Assume we have a knowledge base containing the following axioms :

$$\begin{aligned}\forall x(Student(x) \rightarrow (GradStudent(x) \vee UnderGradStudent(x)))\\ \forall x(GradStudent(x) \rightarrow Student(x))\\ \forall x(UnderGradStudent(x) \rightarrow Student(x))\\ \forall x(GradStudent(x) \rightarrow \neg UnderGradStudent(x))\end{aligned}$$

and in the database, there are 2 tables : *Student* and *UnderGradStudent*. What will happen if a user passes the following query to the database: '*give me all graduate students*'. In fact, from the knowledge base, you can easily see that a graduate student is a student which is not an undergraduate one. Therefore, one just needs to query : '*give me all students which are not undergraduate ones*' in order to have expected answer.

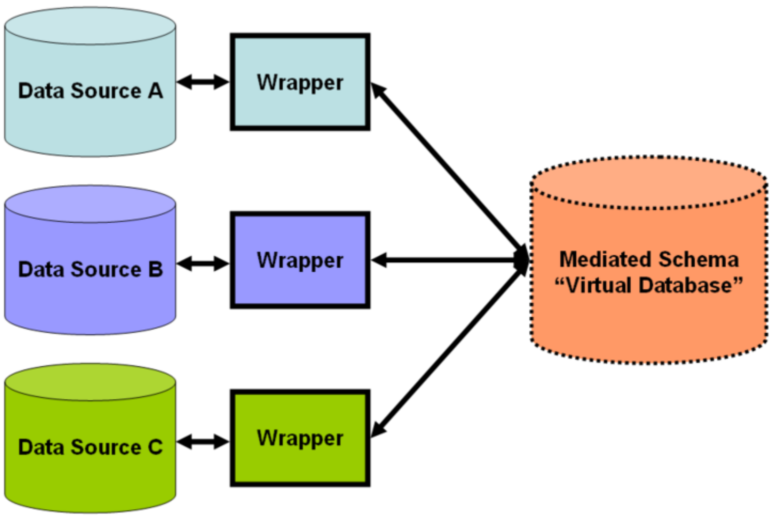


Figure 1.1: Integrated system architecture [28]

The example above shows us multiple perspectives. From the perspective of query optimization, the idea of implicit definability has been implemented purely in DB2 named 'implied predicates' [17]. According to the description of IBM, if in the query you have: A = B, B = C, DB2 will add A = C into the query in order to find the best plan for query answering because possibly doing join B = C and A = C is less expensive than A = B and B = C. Back to our example, if the user wants to ask a query which contains *GradStudent*, we can apply the same technique to rewrite the query. Consequently, the method can be implemented as a module in RDMS where the system knows that accessing to some tables, views is easier than others and then try to use them to rewrite the query.

From another perspective, we can take into account integrated systems, which nowadays are more and more important in the age of information explosion. Consider an architecture of integrated systems in figure 1.1 where a mediated schema is built from local schema. As we have already mentioned above, there is one usual way to answer the query such as rewrite the query in terms of views corresponding to data sources. If we revisit the example and assume that the mediated schema is described by the knowledge base where there are two data sources such as : $UnderGradStudent$ and $Student$ , then the query $GradStudent(x)$ can be answered by rewriting it to $Student(x) \wedge \neg UnderGradStudent(x)$. In general, with this type of systems, if the mediate schema is well designed, all queries should be implicitly definable over views. Therefore, we can apply the method to find the rewritten queries.

Furthermore, let us see the example as in a typical three layer system in Figure 1.2 where the conceptual constraints can be expressed using ontology languages corresponding to the knowledge base and data is in fact stored in above tables: $Student$ and $UnderGradStudent$. This system allows users querying over conceptual language (contains $Student$, $GradStudent$ and $UnderGradStudent$) which is actually richer than the language describing database at logical level (contains $Student$ and $UnderGradStudent$). Consider the query $GradStudent(x)$, applying the method, one can rewrite the query into $Student(x) \wedge \neg UnderGradStudent(x)$, which can be easily transformed to SQL query over relational database and then receive expected answer based on semantics of original query. In fact, from this point of view, this method can support as well the database designing process where designers can check which table is redundant (if its corresponding predicate is implicitly definable from others) and is defined as a view (as a combination of others).

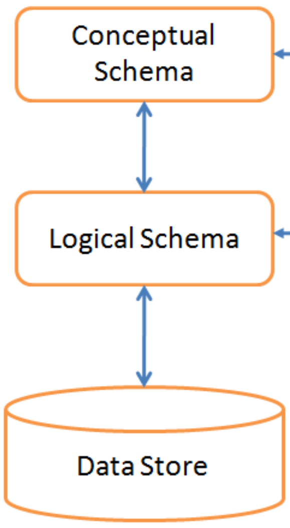


Figure 1.2: Three layers of a system

The second motivation for the method is that it is feasible. As we have already mentioned, this method contains two steps where the first step can be reduced to the validity of a first order formula. In general, this problem is undecidable. However, we know that in most systems, one just considers some decidable fragments of first order logic such as description logic (which is one of the most powerful logic for knowledge representation). Furthermore, with the development of automated theorem proving techniques, one can use some theorem provers to prove the validity of a formula automatically. In the second step, we need to calculate the interpolant of a valid implication. Fortunately, as a consequence of Craig's theorem [9], there is always a method to compute interpolant from tableau based proof or resolution based proof of validity or unsatisfiability.

## 1.2 Previous works

The idea of using Craig's interpolant and Beth's definability for query rewriting were mentioned in several recent researches. In this section, we would like to summary the ideas and results of these researches while the detail framework will be considered later.

In the paper [26], the method is applied for query answering in Description Logic ontologies with DBox while a DBox is syntactically similar to ABox but has different semantics such that the extension of DBox predicates is nothing more than DBox assertions. In the other words, a DBox behaves like a relation database where DBox predicates are tables and Dbox assertions are tuples. Ontologies considered in the paper are general $\mathcal{ALC}$ TBoxes and queries are $\mathcal{ALC}$ concept expressions. According to the paper, one can check whether the answer to a query under an ontology is determined only by DBox or not using normal $\mathcal{ALC}$ reasoning services. If it is the case, a rewriting of the query in terms of DBox predicates will be constructed. In fact, the problem of finding rewritten query in this case coincides with the problem of finding interpolant of $\mathcal{ALC}$ concepts over TBoxes. Therefore, a complete algorithm to compute interpolant based on an adaptation of standard tableaux techniques were introduced as well as its extension for common tableaux optimization techniques. Besides, since rewritten queries are also $\mathcal{ALC}$ concepts, we can easily transform them to SQL queries which can be answered by normal relational database techniques.

In the next paper [4], a query rewriting framework over first order constraints were taken into account with the ultimate goal that one can apply standard relational techniques to answer the rewritten query. In order to fulfill this goal, this paper gave some conditions to reduce the problem of query answering over a knowledge base and a database to the problem of model checking over the database. Besides, the paper also showed important applications of the framework such as: rewriting of a conjunctive query over connected conjunctive views where the problem of rewriting existence is partially decidable; and rewriting of a query over databases which have constraints expressed in the guarded fragment of first order logic.

## 1.3 Contribution of the Thesis

Before turning to the contribution of the thesis, let us summarize the ultimate goal of our efforts. In general, we want to develop a rewriting tool such as :

- The rewritten query can be executed by standard relational engines effectively.
- This tool can be used as a research tool to explore more properties of the rewritten query with the corresponding input.
- This tool is built on top of some theorem provers and provides some services that can be integrated as a rewriting module in application systems.

From foundations of databases [1], it is clear that the problem of checking whether a given query is domain independent or not is undecidable. Therefore, in this thesis we shall focus on a syntatic condition to ensure domain independence named '*safe range*'. It has been agreed that this restriction does not impact the expressiveness of a query such as for all domain independent query, there is a way to construct an equivalent one, which is safe range. Moreover, a first order query can be transformed to a relational algebra query if and only if it is safe range. With this motivation, during this research we shall find conditions of the knowledge base and the original query to have a safe range rewritten query instead of domain independent one. As a negative result, a hypothesis which states that if the input is safe range then so is the output will not hold. However, with these conditions of input, we still have a relaxed property for the output named '*ground domain independent*'. With this property, first we will show that we can have a safe range equivalent query by conjunction with universal relation for each of its free variables. In the next step, we shall point out if we add more constraints to the knowledge base, there will be a less expensive way to change the rewritten query to a safe range one. Unquestionably, these constraints should be feasible in real applications.

In term of tool implementing, we shall describe a rewriting tool which takes a knowledge base, an original query and a set of database predicates as its input, and determines whether the query is rewritable or not. If that is the case, it will return a rewritten query. In order to perform this task, a research on how to compute interpolant of two formulas will be carried out as well and a method based on resolution proof returned by Prover9 - a theorem prover will be implemented.

## 1.4 Organization of the Thesis

This thesis contains 6 chapters : Introduction, Preliminaries, Computing interpolant, The framework, The query rewriting tool and Conclusion. It also has two appendix sections for GUI screenshots and worked out examples. Summary of these chapters are following:

- **Introduction**: This chapter gives readers a brief view of our research throughout motivation, previous works and contribution sections. It contains a global summary of the thesis as well.
- **Preliminaries**: All background knowledge for the thesis such as: first order logic, automated theorem proving techniques, queries, databases, domain independence and description logic will be recalled in this chapter.

- **Interpolant**: This chapter focuses on Craig's interpolant and its properties. Methods to compute interpolant automatically are studied as well in this part.
- **The framework**: This chapter reiterates a general framework based on first order logic mentioned in the paper [4] and its application for description logic [26]. All the results related to safe range property are also mentioned here.
- **The query rewriting tool**: The features of the tool and its services are described here in detail. We mention as well about necessary softwares for the tool in this chapter.
- **Conclusion**: As usual, this chapter sums up our work, its advantages - disadvantages and discusses about future researches.

Chapter 2

# Preliminaries

This chapter will provide the background knowledge related to our topic such as first order logic and its theorem proving techniques. We shall recall as well description logic, a decidable fragment of first order logic where our framework can be applied. Furthermore, since our framework is for query rewriting, some foundations of databases such as: database, query, and domain independence will be reiterated at the last section.

## 2.1 First order logic

Within the thesis, knowledge bases and queries are in first order logic with equality but without functions. However, we shall consider also the resolution method for theorem proving where functional symbols might appear after Skolemization. Therefore, in this section, we are going to look at syntax and semantic of standard first order logic where functional symbols possibly occur.

### 2.1.1 Syntax

All expressions in first order logic can be expressed using the language which is the disjunction of 5 disjoint sets of symbols such as: set of variables $\mathcal{V}$, set of constants $\mathcal{C}$, set of functional symbols $\mathcal{F}$ with associated arities, set of predicate symbols $\mathcal{P}$ with associated arities and the set of logical symbols $\{\forall, \exists, \wedge, \vee, \rightarrow, \neg, =, \leftrightarrow\}$. Basing on these symbols, we have the following concepts:

a) **Term** : A constant in $\mathcal{C}$ is a term. A variable in $\mathcal{V}$ is a term. Moreover, with any $f$ in $\mathcal{F}$, $f(t_1, ..., t_n)$ $(n \geq 1)$ is a term where $t_1, ..., t_n$ are term. The term $f(t_1, ..., t_n)$ can also be called *function*, and the *arity* of this function is $n$

b) **Atomic formula** : An atomic formula is a formula of the form $t_1 = t_2$ where $t_1, t_2$ are terms or $P(t_1, ..., t_n)$ where $P$ is a predicate symbol in $\mathcal{P}$ with arity $n \geq 1$ and $t_1, ..., t_n$ are terms

c) **Formula**:

- Atomic formula is a formula
- If $\phi$ is a formula then so is $\neg\phi$
- If $\phi$ and $\psi$ are formulas then so is $\phi \wedge \psi$
- If $\phi$ is a formula then so is $\exists x \phi$ for any variable $x$
- The formula $\phi \vee \psi$ is a shorten representation of the formula $\neg(\neg\phi \wedge \neg\psi)$
- The formula $\phi \rightarrow \psi$ is an alternative representation of the formula $\neg\phi \vee \psi$
- The formula $\phi \leftrightarrow \phi$ is a shorten formula of $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$

- The formula $\forall x \phi$ is an alternative representation of $\neg \exists x \neg \phi$

Related to variables in a formula, they also define :

d) The set of free variables in formula $\phi$ denoted by $free(\phi)$ is a subset of all variables in $\phi$ such as:
   - If $\phi$ is atomic then $free(\phi)$ is the set of all variables occurring in $\phi$
   - If $\phi = \neg \psi$ then $free(\phi) = free(\psi)$
   - If $\phi = \psi \wedge \theta$ then $free(\phi) = free(\psi) \cup free(\theta)$
   - If $\phi = \exists x \psi$ then $free(\phi) = free(\psi) - \{x\}$

e) A *sentence* or a *closed formula* is a formula $\phi$ such that $free(\phi) = \emptyset$

At general level, we have :

f) A *signature* is the set of constant, functional and predicate symbols with corresponding arities. Given a formula $\phi$, $\sigma_\phi$ is the *signature* of $\phi$ if it is a signature containing only constants, function symbols and predicates symbols in $\phi$. We also can say that a signature is the set of non-logical symbols.

g) Sometimes, we also use '*language*' with the same meaning as signature.

### 2.1.2 Semantic

Semantic of a first order formula is given by the notion of interpretations. In the other words, throughout an interpretation and its corresponding evaluation function, we will know the truth value (**true**, **false**) of a first order formula.

a) **Interpretation** : An interpretation $\mathcal{I}$ is a tuple $(D, S, v_a)$ where
   - Domain $D$ is a non empty set of objects
   - $S$ is a signature mapping that:
     - $S(c) \in D$ for all constants $c \in \mathcal{C}$
     - $S(f)$ is a function from $D^n$ to $D$ where $f \in \mathcal{F}$ is a function with arity $n$
     - $S(P)$ is a relation from $D^n$ to $\{\mathbf{true}, \mathbf{false}\}$ where $P \in \mathcal{P}$ is a predicate with arity $n$
   - Variable assignment $v_a$ maps each variable $v \in \mathcal{V}$ to an element of $D$

b) **Evaluation function** : An evaluation function $e_I$ corresponding to $\mathcal{I}$ evaluates terms and formulas as follows:
   - $e_I(t)$ where $t$ is a term
     - If $t$ is a variable then $e_I(t) = v_a(t)$
     - If $t$ is a constant then $e_I(t) = S(t)$
     - If $t$ if a function $f(t_1, ..., t_n)$ then $e_I(t) = (S(f))(e_I(t_1), ..., e_I(t_n))$
   - $e_I(P(t_1, ..., t_n)) = (S(P))(e_I(t_1), ..., e_I(t_n))$
   - $e_I(t_1 = t_2) = \mathbf{true}$ iff $e_I(t_1) = e_I(t_2)$
   - $e_I(\phi)$
     - If $\phi = \neg \psi$ then $e_I(\phi) = \mathbf{true}$ iff $e_I(\psi) = \mathbf{false}$
     - If $\phi = \psi \wedge \theta$ then $e_I(\phi) = \mathbf{true}$ iff $e_I(\psi) = \mathbf{true}$ and $e_I(\theta) = \mathbf{true}$
     - If $\phi = \exists x \psi$ then $e_I(\phi) = \mathbf{true}$ iff for some $d \in D$, $e_{I'}(\psi) = \mathbf{true}$ where $I' = (D, S, v'_a)$ and $v'_a$ is the same with $v_a$ except that $v'_a(x) = d$.

c) Formula $\phi$ is *satisfiable* if there is an $\mathcal{I}$ such that $e_I(\phi) = \mathbf{true}$. In this case, $\mathcal{I}$ is a *model* of $\phi$ denoted by $I \models \phi$

d) Formula $\phi$ is *valid* iff for all $\mathcal{I}$ $e_I(\phi) = \textbf{true}$

e) Related to sentences, we can define $\mathcal{M} = (D, S)$ and say that $\mathcal{M}$ is a model of a formula $\phi$ iff for all variable assignment $v_a$, $\mathcal{I} = (D, S, v_a)$ is a model of $\phi$. In the other words, the semantic of sentences just depend on the domain and interpretation function. Therefore, for the case of of sentences we can call $\mathcal{I} = (D, S)$ is an interpretation.

f) A set of sentences will be called '*theory*'. A theory is consistent iff there is an interpretation which is a model of all sentences in the theory. Given a theory $\mathcal{T}$ and a formula $\phi$, we say that $\mathcal{T}$ *entails* $\phi$ denoted by $\mathcal{T} \models \phi$ if all models of $\mathcal{T}$ are models of $\phi$.

### 2.1.3 Normal forms

In some applications, before being processed a first order formula has to be transformed to a normal form. In this thesis, we will concern below normal forms

a) **Negation normal form** (NNF) : A formula is in negation normal form iff it does not contain implication or equivalent symbols and the negation symbol appears only in front of its atomic formulas. In order to transform an arbitrary formula to negation normal form, we have to apply the following rules :

$$\begin{aligned}
&(A \to B) \longrightarrow (\neg A \land B)\\
&\neg\neg A \longrightarrow A\\
&\neg(A \land B) \longrightarrow (\neg A \lor \neg B)\\
&\neg(A \lor B) \longrightarrow (\neg A \land \neg B)\\
&\neg(\forall x A) \longrightarrow (\exists x \neg A)\\
&\neg(\exists x A) \longrightarrow (\forall x \neg A)
\end{aligned}$$

b) **Prenex normal form** (PNF) : In prenex normal form, all quantifiers appear at the beginning of the formula. Below are rules to transform a formula to its prenex normal form. Notice that before apply these rules, the variables in formula should be renamed such that neither a variable appears as both free and bounded nor a variable bounded by more than one quantifier.

$$\begin{array}{ll}
(\forall x A) \land B \longrightarrow \forall x(A \land B) & (\forall x A) \lor B \longrightarrow \forall x(A \lor B)\\
(\exists x A) \land B \longrightarrow \exists x(A \land B) & (\exists x A) \lor B \longrightarrow \exists x(A \lor B)\\
\neg \forall x A \longrightarrow \exists x(\neg A) & \neg \exists x A \longrightarrow \forall x(\neg A)
\end{array}$$

c) **Prenex conjunctive normal form** (PCNF) : When a formula is in prenex normal form, we can transform its quantifier-free part to conjunctive normal form as in propositional logic to have prenex conjunctive normal form by first transform it into negation normal form and then apply the Demorgan's law: $A \land (B \lor C) \longrightarrow (A \land B) \lor (A \land C)$

## 2.2 Theorem proving in First order Logic

Theorem proving problem in first order logic can be understand as: given a theory and a first order closed formula, prove that the theory entails the formula. It is easy to see that we can reduce this problem to the problem of checking whether a a first order formula is unsatisfiable or not. Even though in the case of first order logic this problem is undecidable, there are some typical sound and complete structural calculuses that we can follow such as: Natural Deduction or Sequence Calculus. However, in this section, instead of discussing about these calculuses, we shall focus on two important improved techniques for automated theorem proving : Tableaux and Resolution which are also sound and complete.

### 2.2.1 Tableaux

Tableaux technique is one of the most simply techniques to check the satisfiability of a formula since it is close to the formula's semantic. Its basic idea is from the cut-elimination theorem of structural proof theory. This method has been applied widely in some popular fragments of first order logic like description logic or expanded to modal logic and higher order logic.

In order to apply tableaux method, the input formula should be in negation normal form. In general, given a formula, a tableau which basically is a tree reflecting all the possible models of the formula, is built from the following completion rules [12]:

- And-rule : $\dfrac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}}$ means that a model of a conjunction should be the model of each conjunct.

- Or-rule: $\dfrac{\phi \vee \psi}{\phi \quad | \quad \psi}$ means that a model of disjunction should be a model of one disjunct. Therefore, in this case two branches of this tableau are generated.

- Forall-rule: $\dfrac{\forall x.\phi}{\begin{array}{c}\phi\{x/t\} \\ \forall x.\phi\end{array}}$ says that a model of an universal quantified formula is also model of the formula where a quantified variables is substituted with any term which appear in the tableau.

- Exists-rule: $\dfrac{\exists x.\phi}{\begin{array}{c}\phi\{x/a\} \\ \exists x.\phi\end{array}}$ says that a model of an existential quantified formula is also model of the formula where a quantified variable is substituted with a new constant.

- Equal-rule 1: $\dfrac{\phi}{t = t}$ means that a model of a formula should be a model of the equation $t = t$ for every term $t$ occurring in this formula.

- Equal-rule 2: $\dfrac{\begin{array}{c}\phi(t) \\ t = u\end{array}}{\phi(u)}$ says that if an interpretation satisfies a formula and an equation between two terms, it should satisfy the formula where one term is substituted by another one.

In the tableau, a branch is called *closed* iff it contains $\phi$ and $\neg\phi$. A branch is called *completed* iff there is no more applicable rule on it. The tableau construction process will stop when all of its branches are closed or completed. At this time, the corresponding formula is unsatisfiable iff all branches are closed. A completed branch (if have) will reflect a model of the formula. Notice that this construction process does not always terminate, then in this case we cannot decide the satisfiability of the formula.

### 2.2.2 Resolution with unification

In this section, we shall consider another method for automated theorem proving named resolution where the fundamental idea is based on Herbrand's theorem and DavisPutnam algorithm [10]. Using this method, a theorem will be proved by refutation such that its negation will be first transformed to a set of clauses, then using resolution, factoring and paramodulation rules to generate new clauses until an empty clause (a contradiction) is created. Notice that the clauses generating strategy is very important in practical provers since it decides the length of proofs. We can list here some popular strategies such as : breadth first strategy, set of support strategy [30], unit preference strategy [29], linear input form strategy. However, we shall not go into detail for these strategies but only consider theoretical foundation of the method.

a) **Skolemization**
Skolemization is a process of removing existential quantifiers from a prenex normal form formula. Assume that we have a PNF formula :

$$\phi = Q_1 x_1 Q_2 x_2 .... Q_n x_n \psi(x_1, ..., x_n)$$

For each $Q_i$ such as $Q_i = \exists$ and $Q_1, .., Q_{i-1}$ are $\forall s$, we replace $x_i$ by $f(x_1, ..., x_{i-1})$ in $\psi$. In the case $i = 0$, we just need to replace $x_i$ by a fresh constant $c$. The term corresponding to $f(x_1, ..., x_{i-1})$ or $c$ is called *Slokem term.* As we see, the formulas after and before Skolemization are not equivalent. Nevertheless, they are *sat-equivalent*, which means that if one formula is satisfiable, then so is the other.

b) **Clause form**
As we have already mentioned, in order to apply resolution refutation method a formula has to be transformed to a set of clauses. More precisely, each clause has to be the disjunction of literals and does not contain any quantifier. Here are the steps to generate clauses from a formula :

- Convert the formula to prenex conjunctive normal form
- Skolemization to remove existential quantifiers. Formula at this step called Slokem prenex normal form
- Drop all universal quantifiers
- Each clause is a conjunct

c) **Resolution rule**
Since the first time introduced by John Alan Robinson in 1965 [25], resolution rule has been applied broadly in propositional theorem proving. Given two clauses $C \vee L$ and $D \vee \neg L$ where $C, D$ are clauses and $L$ is an atomic formula, we have the following rule : $\dfrac{C \vee L \quad D \vee \neg L}{C \vee D}$ In this case, we say $C \vee D$ is resolvent of $C \vee L$ and $D \vee \neg L$ and $L$ is their pivot literal. It's obvious that this rule is sound since the resolvent is the logic consequence of resolved clauses.

d) **Herbrand interpretation and Herbrand theorem**
Herbrand theorem is a very important result in first order logic because it gives a way to reduce first order unsatisfiability problem to propositional one. For that reason, it also is a crucial theoretical foundation for automated theorem proving. General speaking, the theorem says that only Herbrand interpretation is enough for checking validity of a first order formula. Given a signature $\Sigma$, a Herbrand interpretation over $\Sigma$ is a tuple $\mathcal{I} = (D, S, v_a)$ where:

- $D$ named *Herbrand universe* defined as: every constants in $\Sigma$ is in $D$. If $f$ is a n-ary function in $\Sigma$ then $f(t_1, ..., t_n)$ is in $D$ where $t_1, ..., t_n$ are in $D$.
- $S$ interprets each constant to itself. For each n-ary function $f$ $(S(f))$ $(t_1, .., t_n) = f(S(t_1), ..., S(t_n))$. For each n-ary predicate symbol $P$, $S(P)$ is mapped to a n-ary relation over Herbrand universe.
- $v_a$ assigns each variable to a term in $D$

A ground instance of a clause $C$ is $C$ where all variables are replaced by Herbrand term. With these definitions, the Herbrand theorem can be stated briefly as follow :

**Theorem 2.1 (Herbrand theorem)** *A first order formula $\phi$ is unsatisfiable iff there is a finite set of ground clauses of $\phi$'s clause form that is unsatisfiable in propositional calculus.*

Related to the proof of this theorem, one can refer to [19]. Apply this theorem, we have a method named ground resolution where the proof of an unsatisfiable formula is constructed by resolution from a set of its ground clauses .

e) **Unification**
Unification of two terms is a process of finding substitution such that under this substitution two terms are identical where a substitution is a set of mappings variables to terms. Two terms are called *unifiable* iff exists such substitution and this substitution is called *unifier* In general, there might be more than one unifier between two unifiable terms but people just concerns about the most powerful one named *most general unifier.* In unification theory, the most general unifier of 2 terms $t_1, t_2$ is the unifier $\pi$ such that for all unifiers $\pi'$ of these terms, there is a substitution $\pi_t$ such that : $(t_1)\pi' = ((t_1)\pi)\pi_t = ((t_2)\pi)\pi_t = (t_2)\pi'$ where $(t)\pi$ is the result term of term $t$ after apply the substitution $\pi$. Instead of terms, we also have the same definition for formulas.
In first order logic, the unification problem is decidable. More precisely, there is an algorithm telling us that two given terms are unifiable or not, if the answer is yes, it returns the most general unifier between them. The simplest algorithm was proposed by John Alan Robinson [25] and applied in his first order resolution theorem prover.

f) **General Resolution with factoring and paramodulation**
With the notion of unification, we can generalize the resolution rule as: given two clauses $C \vee L$ and $D \vee \neg L$ where $L$ and $L'$ are unifiable and $\pi$ is their most general unifier, $\dfrac{C \vee L \quad D \vee \neg L}{(C \vee D)\pi}$
In fact, in first order logic without equality, the proof system with only general resolution rules is not complete. In order to have a complete system, we also need *factoring rule* which can be described as follow : given a clause $C \vee L \vee L'$where $L$ and $L'$ are unifiable and $\pi$ is their most general unifier, then $\dfrac{C \vee L \vee L'}{(C \vee L)\pi}$
In first order logic with equality, thing becomes more complicated. Fortunately, thanks to paramodulation, we still have a complete system. Given two clauses $C \vee s = t$ and $L(s) \vee D$ where $L(s)$ is a literal contains s. $s$ and $t$ are unifiable by the most general unifier $\pi$, we have the following *paramodulation rule*: $\dfrac{C \vee s = t \quad L(s) \vee D}{(C \vee L(t) \vee D)\pi}$where $L(t)$ is $L$ with replacement of $s$ by $t$

## 2.3 Description logic

*Description logic* (DL) is a well-known fragment of first order logic together with the development of knowledge representation system because it provides the logical foundation for ontologies and semantic web. In general, basic description logic is decidable and expensive *enough* for information system modeling. DL describes a system throughout *concepts*, *roles*, *individuals* and their relationship in the system where an individual is an object in the system, a concept is a class of objects and a role is a relation between two objects.

### 2.3.1 Syntax

The language of DL in general often contains two disjoint sets of symbols for atomic concepts and atomic roles which corresponding to symbols of unary predicates and binary predicates. It also contains a set of symbols for concept and role construction such as : negation($\neg$), conjunction ($\sqcap$), disjunction ($\sqcup$), existence ($\exists$), universal quantification ($\forall$), cardinality restrictions ($\geq_n, \leq_n$) as well as a set of symbols for the relation between concepts and between roles such as : subsumption ($\sqsubseteq$). Almost description languages also contain $\perp$ and $\top$ symbols as special concepts. Depend on the syntax for building concepts and roles, we have different description logics where the naming convention [3] for each type is following :

- $\mathcal{AL}$ Basic DL with atomic negation, concept intersection, value restriction and limited existential quantification,
- $\mathcal{F}$ Functional properties.
- $\mathcal{E}$ Full existential qualification

- $\mathcal{U}$ Concept union.
- $\mathcal{C}$ Complex concept negation.
- $\mathcal{S}$ An abbreviation for $\mathcal{ALC}$ with transitive roles.
- $\mathcal{H}$ Role hierarchy
- $\mathcal{R}$ Limited complex role inclusion axioms; reflexivity and irreflexivity; role disjointness.
- $\mathcal{O}$ Nominals.
- $\mathcal{I}$ Inverse role properties.
- $\mathcal{N}$ Cardinality restrictions
- $\mathcal{Q}$ Qualified cardinality restrictions
- $^{(\mathcal{D})}$ Use of datatype properties, data values or data types.

Let us now consider the detail syntax of $\mathcal{ALC}$ where our framework is fully applied. Let $N_C$ and $N_R$ be sets of concept and role names. A set of concepts in $\mathcal{ALC}$ is the smallest set $\mathcal{C}$ which fulfills :

- If $C \in N_C \cup \{\bot, \top\}$ then $C \in \mathcal{C}$
- If $C, D \in \mathcal{C}$ then $C \sqcup D$ and $C \sqcap D$ are in $\mathcal{C}$
- If $C \in \mathcal{C}$ then so are $\forall R.C$ and $\exists R.C$ where $R \in N_R$

### 2.3.2 Semantic

With the above syntax, the semantic for $\mathcal{ALC}$ is given by interpretation $I = (\Delta^I, \cdot^I)$ where $\Delta^I$ is a nonempty set and $\cdot^I$ is a function that map a concept to a subset of $\Delta^I$ as in the table 2.1 Under above interpretation $I$, a concept $C$ is *consistent* iff $C^I \neq \emptyset$. An *axiom* in $\mathcal{ALC}$ is written

| Concept | Semantic |
|---|---|
| $A$ | $A^I \subseteq \Delta^I$ |
| $R$ | $R^I \subseteq \Delta^I \times \Delta^I$ |
| $\top$ | $\Delta^I$ |
| $\bot$ | $\emptyset$ |
| $C \sqcap D$ | $C^I \cap D^I$ |
| $C \sqcup D$ | $C^I \cup D^I$ |
| $\neg C$ | $\Delta^I \backslash C^I$ |
| $\exists R.C$ | $\{x \mid \text{exists y such that } R(x,y) \wedge C(y)\}$ |
| $\forall R.C$ | $\{x \mid \text{forall y } R(x,y) \rightarrow C(y)\}$ |

Table 2.1: Semantic of $\mathcal{ALC}$

in the form $C \sqsubseteq D$ named $C$ *subsumes* $D$ or $C \equiv D$ where $C \equiv D$ means $C \sqsubseteq D$ and $D \sqsubseteq C$ . An interpretation $I$ satisfies axiom $C \sqsubseteq D$ iff $C^I \subseteq D^I$ and as a consequence, $I$ satisfies $C \equiv D$ iff $C^I = D^I$.

A *Tbox* is a finite set of axioms. A interpretation $I$ is a *model* of Tbox $T$ iff it satisfies all the axioms in $T$. An *Abox* in $\mathcal{ALC}$ is a set of assertions of the form $C(a)$ or $R(a, b)$ where $C$ is a concept and $R$ is an atomic role. $C(a)$ is satisfied by interpretation $I$ iff $a^I \in C^I$ and $R(a, b)$ is satisfied by interpretation $I$ iff $(a^I, B^I) \in R^I$. A interpretation $I$ is said to be a *model* of Abox $\mathcal{A}$ iff every assertion in $\mathcal{A}$ is satisfied by $I$. A *knowledge base* (KB) is tuple of $(Tbox, ABox)$ and an interpretation is said to be a *model* of KB iff it is a model of both *Tbox* and *Abox*. A KB is satisfiable iff it has a model. Given a KB $\Sigma$ and an axiom or an assertion $\phi$, we say $\Sigma \models \phi$ iff all models of $\Sigma$ are models of $\phi$.

### 2.3.3 Reasoning in Description Logic

There are several type of reasoning services in DL such as :

- Concept satisfiability : given a knowledge base $\Sigma$ and a concept $C$, check $\Sigma \models C \equiv \bot$
- Subsumption : given a knowledge base $\Sigma$ and 2 concepts $C, D$, check $\Sigma \models C \sqsubseteq D$
- Satisfiability: given a knowledge base $\Sigma$, check whether $\Sigma$ is satisfiable or not
- Instance checking: Check whether $\Sigma \models C(a)$ or not where $\Sigma$ is a knowledge base
- Retrieval: Find $a$ such that $\Sigma \models C(a)$ where $\Sigma$ is a knowledge base

As we know, these reasoning services can be reduced to the satisfiability problem and this problem is decidable in DL. In most of DL reasoners, they often apply tableaux method mentioned generally in the previous section. Now we shall look at the specific algorithm for $\mathcal{ALC}$. Notice that before apply the algorithm, all concepts have to be transformed to negation normal form (NNF) such that negation appears only in front of atomic concepts. We can transform an arbitrary concept to a NNF by following rules:

$$\begin{aligned}
&\neg\neg C \longrightarrow C\\
&\neg(C \sqcup D) \longrightarrow (\neg C \sqcup \neg D)\\
&\neg(C \sqcap D) \longrightarrow (\neg C \sqcap \neg D)\\
&\neg(\exists R.C) \longrightarrow (\forall R.\neg C)\\
&\neg(\forall R.C) \longrightarrow (\exists R.\neg C)
\end{aligned}$$

After transform all concepts to NNF, an empty tableau $S$ will be initialized. Each branch of the tableau during completion process contains expression of the form $x : C$ or $xRy$ where $C$ is a concept and $R$ is a role. A branch is *closed* if it contains a *clash* $\{x : A, x : \neg A\}$. The following completion rules are applied until all the branches of the tableau are closed.

- $S \longrightarrow_{\sqcap} \{x : C, x : D\} \cup S$ if $x : C \sqcap D$ is in $S$ and $x : C$, $x : D$ are not both in $S$
- $S \longrightarrow_{\sqcup} \{x : C\} \cup S$ or $S \longrightarrow_{\sqcup} \{x : D\} \cup S$ if $x : C \sqcup D$ is in $S$ and neither $x : C$ nor $x : D$ in $S$
- $S \longrightarrow_{\exists} \{xRy, y : C\} \cup S$ if $x : \exists R.C$ is in $S$, $y$ is a new variable, there is no $z$ such that both $xRz$ and $z : C$ are in $S$
- $S \longrightarrow_{\forall} \{y : C\} \cup S$ if $x : \exists R.C$ is in $S$, $xRy$ is in $S$, and $y : C$ is not in $S$

In some practical reasoners, several optimization techniques for above algorithm have been applied. We can list here some common techniques [3] such as :

- Non-atomic closure: In this technique, the definition of *clash* is relaxed to non-atomic concepts such as $\{x : C, x : \neg C\}$ is a clash even when $C$ is a complex concept. If a branch contains this clash, we will stop applying completion rules on it.
- Semantic branching: In semantic branching method, the semantic of disjunction is considered such that : if $x : C \sqcup D$ is true then $x : C$ is true or $x : \neg C, x : D$ are true. Therefore, the rule for $\sqcup$ is slightly modified to: $S \longrightarrow_{\sqcup} \{x : C\} \cup S$ or $S \longrightarrow_{\sqcup} \{x : D, x : \neg C\} \cup S$ if $x : C \sqcup D$ is in $S$ and neither $x : C$ nor $x : D$ in $S$
- Lazy unfolding and absorption: Regarding to this optimization method, first the TBox $T$ will be changed to a new Tbox $T' = T_G \cup T_U$ throughout absorption technique such that $T_U$ contains only axioms in the form of $A \sqsubseteq C$ or $A \equiv C$ where $A$ is an atomic concept. After that, all the axioms in $T_U$ are considered in new completion rules by an intuitively idea such that: if $x : A$ is in $S$ and $A \sqsubseteq C$ in $T_U$ then $x : C$ should be in $S$; if $x : A$ is in $S$ and $A \equiv C$ in $T_U$ then $x : C$ is in $S$; if $x : \neg A$ is in $S$ and $A \equiv C$ in $T_U$ then $x : \neg C$ is in $S$

## 2.4 Database and Query

In this section, we shall recall some important definitions in database theory such as : database instance, database constraints, query, domain independence, safe range in the language and semantics of first order logic without functions.

### 2.4.1 Database and constraints

Let $\Sigma$ is a signature containing an (infinite) set of *constants* $\mathcal{C}$ called *database constants* and a set of predicates $\mathcal{P}$. A *database instance* (or *database* in a shorten way) $DB$ over $\Sigma$ is a set of assertions in the form of $P(c_1, ..., c_n)$ where $P$ is a n-ary predicate in $\mathcal{P}$ and $c_1, ..., c_n$ are constants in $\mathcal{C}$. We also call a set of constants appearing in the database $C_{DB}$ is *active domain* of $DB$ and a set of predicates appearing in $DB$ is the set of database predicates denoted by $P_{DB}$.

A database *constraint* of $DB$ is an arbitrary first order sentence over $\Sigma$. A knowledge base $KB$ is a set of constraints. In relational database, all popular constraints are in of the form : $\forall x_1, ..., x_k A_1 \wedge ... \wedge A_n \rightarrow \exists y_1, ..., y_l B_1 \wedge ... \wedge B_m$ [8] where $x_1, .., x_k, y_1, ..., y_l$ are variables and $A_1, ..., A_n, B_1, ..., B_m$ are database predicates or equalities. In detail, a constraint is :

- *Full dependencies* if there is no existential variables (l = 0)
- *Tuple-generating dependencies* (TGDs) if there is no equality atoms
- *Equality-generating dependencies* (EGDs) if $m = 1$ and $B_1$ is an equality atom
- *Functional dependencies* (FDs) if $n = 2$ and it is an EGDs
- *Join dependencies* (JDs) if it is a TGDs with the left hand side conjunct is a multiway join
- *Denial constraints* if $l = 0$ and $m = 0$
- *Inclusion dependencies* (INDs) if $n = m = 1$ and no equality atoms

In term of semantic, let us consider an interpretation $I = (\Delta^I, \cdot^I)$ where $\Delta^I$ is a domain and $\cdot^I$ is an interpretation function. $I$ is said that it *embeds* [4] database $DB$ (written $I_{DB}$) if for every database constants $c \in C_{DB}$ $c^I = c$ and for every n-ary database predicate $P \in \mathcal{P}_{DB}$ we have $(c_1, ..., c_n) \in P^I$ if and only if $P(c_1, ..., c_n)$ is in $DB$. Given database $DB$ and knowledge base $KB$, we say that $DB$ satisfies $KB$ iff there is an interpretation $I_{DB}$ which is a model of $KB$. From this definition, we imply that the domain of $I_{DB}$ always contains $C_{DB}$.

To generalize the characteristic of database constants, we have the notion of '*standard name assumption*' as follows:

**Definition 2.2 (Standard name assumption)** *A set of constants $\mathcal{A}$ satisfies standard name assumption if for every $c \in \mathcal{A}$, for every interpretation $I$, $c^I = c$*

Using this notion, we can say that the set of database constants $C_{DB}$ satisfies standard name assumption. In fact, in relational database, the assumption holds for the set of all constants.

### 2.4.2 Query

In general, a query is a first order formula. If the formula is closed, we say it is a boolean query. Intuitively, the answer of a boolean query is a boolean value where the answer of non-boolean query is a set of possible tuples of constants assigned to its free variables.

Given a query $Q$ and $X = \{x_0, ..., x_n\}$ is the set of its free variables, we will write it as $Q_{[X]}$ or $Q(x_0, .., x_n)$. We also define a substitution $\Theta_X^{\mathcal{C}}$ is a total assignment which assigns each variable in $X$ to a constant in $\mathcal{C}$. This definition holds as well for the case $X = \emptyset$. The answers of $Q_{[X]}$ over the database $DB$ and the knowledge base $KB$ is the set

$$A = \{\Theta_X^{\mathcal{C}} | \text{for every } I_{DB} \text{ is a model of } KB : I_{DB} \models Q_{[X/\Theta_X^{\mathcal{C}}]}\}$$

In the case of boolean query, if $A \neq \emptyset$ the answer is *yes* and $A = \emptyset$ the answer is *no*.

### 2.4.3 Domain Independence

Since a query can be an arbitrary first order formula, its answer can be infinite (since the domain is not restricted to be finite) or depend on the domain. For example, the query $Q(x) = -Student(x)$ over the database $Student(A), Student(B)$, with domain $\{A, B, C\}$ we have the answer $\{x = C\}$ and with domain $\{A, B, C, D\}$ we have the answer $\{x = C, x = D\}$ and if we change the domain to an infinite one, we will get an infinite answer. Therefore, in real applications, they often discard this kind of query. In the other words, we are just interested in *domain independent* queries where their answers do not depend in the domain.

The formal definition [4] of a domain independent query is given as follow :

**Definition 2.3 (Domain independent query)** *A query is domain independent iff for every pair of first order interpretation $I$ and $J$ which agree on the interpretation function ($\cdot^I = \cdot^J$) and for every substitution $\Theta_X^{\Delta^I \cup \Delta^J}$ :*

$$act - range(\Theta_X^{\Delta^I \cup \Delta^J}) \subseteq \Delta^I \textit{ and } I, \Theta_X^{\Delta^I \cup \Delta^J} \models Q_{[X]}$$

$$\textit{iff}$$

$$act - range(\Theta_X^{\Delta^I \cup \Delta^J}) \subseteq \Delta^J \textit{ and } J, \Theta_X^{\Delta^I \cup \Delta^J} \models Q_{[X]}$$

*where $act - range(\Theta_X^C)$ is the active image of the mapping $\Theta_X^C$*

Intuitively, since the answer of the query does not depend on the domain we can evaluate the query over the smallest domain. In the case of relational database, the smallest domain is the union of active domain $C_{DB}$ and $C_Q$ where $C_Q$ is the set of constants appearing in query $Q$.

In this thesis, we also need a weaker version of domain independence named '*ground domain independence*'

**Definition 2.4** *A query $Q_{[X]}$ is ground domain independent iff for all substitution $\Theta_X^C$, $Q_{[X/\Theta_X^C]}$ is domain independent.*

Obviously, domain independence implies ground domain independence while the opposite direction does not hold.

**Example 2.1** *The query $Student(x)$ is both domain independence and ground domain independence while the query $\neg Student(x)$ is just ground domain independent. The query $\forall x$ $Teaching(x, y)$ is neither domain independent nor ground domain independent.*

### 2.4.4 Safe range

Domain independence queries make the query evaluation problem be less complicated. Unfortunately, given a query, the problem of checking whether the query is domain independent or not is undecidable [1]. Therefore, in every query language, they often define a syntax restricted subset of domain independent queries which is decidable and maintains the expression power.

In term of first order queries, *safe range* is a famous syntax restriction introduced by Codd. Intuitively, a query is safe range if and only if its variables are bounded by positive predicates or equalities. The following formal definitions [1] will explain what is a safe range query and how to check whether a query is safe range :

**Definition 2.5 (Safe range normal form)** *denoted by SRNF*
*A first order formula can be transformed to SRNF by following steps :*

- *Variable substitution: no distinct pair of quantifiers may employ same variable.*
- *Remove universal quantifiers*
- *Remove implications*
- *Push negation*
- *Flatten and/or*

**Definition 2.6 (Range restriction of a formula)** *denoted by rr*
*Input : a formula $\varphi$ in SRNF*
*Output : a subset of $free(\varphi)$ or $\bot$*
*Case $\varphi$ of*

- $R(e_1, ..., e_n)$ *: $rr(\varphi)$ = set of variables in $e_1, ..., e_n$*
- $x = a$ *or* $a = x$ *:* $rr(\varphi) = \{x\}$
- $\varphi_1 \wedge \varphi_2$ *:* $rr(\varphi) = rr(\varphi_1) \cup rr(\varphi_2)$
- $\varphi_1 \vee \varphi_2$*:* $rr(\varphi) = rr(\varphi_1) \cap rr(\varphi_2)$
- $\varphi_1 \wedge x = y$ *:* $rr(\varphi) = rr(\varphi_1)$ *if* $\{x, y\} \cap rr(\varphi_1) = \emptyset$*;* $rr(\varphi) = rr(\varphi_1) \cup \{x, y\}$ *otherwise*
- $\neg\varphi_1$*:* $rr(\phi) = \emptyset$
- $\exists\vec{x}\varphi_1$ *:* $rr(\varphi) = rr(\varphi_1)\backslash\vec{x}$ *if* $\vec{x} \subseteq rr(\varphi_1)$*;* $rr(\varphi) = \bot$ *otherwise*

*Note :* $\bot \cup Z = \bot \cap Z = \bot\backslash Z = Z\backslash\bot = \bot$

**Definition 2.7 (Safe range)** *A formula $\varphi$ is safe range iff $rr(SRNF(\varphi)) = free(\varphi)$*

**Example 2.2** *$Q(y) = \forall x Student(x, y)$ is not safe range while $Q(y) = \exists x Student(x, y)$ is safe range.*

Related to expression power of safe range queries, below theorems hold :

**Theorem 2.8** *Safe range queries are domain independent*

**Proof**
Sketch. Instead of proving directly, one can show that every first order safe range query can be transformed to a relational algebra query (Codd's theorem) and all the relational algebra queries are domain independent. ■

**Theorem 2.9** *Every domain independent query $Q_{[X]}$ over a database $DB$ has an corresponding safe range query giving the same answer.*

**Proof** Sketch. Let $D$ is an unary predicate such as $D(x)$ means $x$ is a database constant of $DB$. Let $Q'_{[X]}$ is the same with $Q_{[X]}$ except for each subformula of $Q_{[X]}$ we will replace $\forall y\psi(y)$ by $\forall y(D(y) \rightarrow \psi(y))$, $\exists x\psi(x)$ by $\exists x(\psi(x) \wedge D(x))$ in $Q'_{[X]}$. Therefore $Q'_{[X]} \wedge D(x_0) \wedge D(x_2) \wedge ... \wedge D(x_n)$ $(X = \{x0, ..., x_n\})$ is a safe range query. Of course, it also have the same answer with $Q_{[X]}$ because for every constant $c$ in the domain $\mathcal{C}$ of $DB$, $D(c)$ is true. ■

Related to relational algebra - theoretical foundation of the most popular query language SQL, there is following famous theorem introduced by Codd:

**Theorem 2.10 (Codd's theorem)** *A first order query is definable in relational algebra if and only if it is safe range.*

Chapter 3

# Interpolant

In this chapter, we shall discover deeply one of the most famous theorems in logic named Interpolation theorem invented by William Craig in 1957 [9]. This theorem has a strong impact in both theoretical and practical aspects of mathematical logic and computer science. In the first section, we will give a formal theorem and its proof for propositional and first order case. We also will list some strengthen versions as well as some remarkable applications of the theorem. In the second one, constructive proofs based on resolution and tableau for the theorem will be taken into account.

## 3.1 Craig's interpolant

Interpolation theorem is a theorem about the relationship between syntax and semantic aspect of a logic. Roughly speaking, given two sentences $\phi$ and $\psi$ such that $\phi \to \psi$ is valid then there is a sentence $\iota$ called intepolant where all non logical symbols occurring in $\iota$ also occur in both $\phi$ and $\psi$ and $\phi \to \iota$; $\iota \to \psi$. Craig proved the theorem for first order logic and it has been generalized to many other logics such as equational logic or modal logic. Beside compactness theorem, interpolation theorem is a crucial property had to be checked for any new logic. Let us look at some applications to explain why the theorem is so important.

In term of theory consistency [13], given two theories $\Delta$ and $\Gamma$ which agree on their common language, one can ask whether we can put them together to have a consistent theory $\Delta \cup \Gamma$. In first order logic, the answer is 'yes' and known as Robinson's consistent theorem. It is not difficult to see this theorem can be proved by interpolation theorem because if $\Delta \cup \Gamma$ is inconsistent then $\Delta \to \neg\Gamma$ is valid. Therefore, there is an $I$ in the common language of $\Delta$ and $\Gamma$ such as $\Delta \to I$ and $I \to \neg\Gamma$ and it is contradicted to the statement that $\Delta$ and $\Gamma$ agree in common language. In general, theory consistency theorem holds in any kind of logic which has interpolant property.

In term of proof theory, interpolants can be constructed from proofs of validity. Hence, if the proof system is algorithmitic then there is also an algorithm to compute interpolant. In the other side, from the (non)existence of interpolant property of a logic, we also can say whether the logic has a good proof system or not [14].

Now, let us consider the application of interpolation in model checking - one of great applications of logic in computer science and software engineering. In model checking, a software or hardware system is modeled as a state transition system which contains a set of states, the transtion relation between 2 states, the set of initial states and the labelling function for each state. A property will be 'safe' in this system if it is not satisfiable at any reachable state of initial states. In bounded model checking, they consider only the set of states that are reachable from initial states in at most $k$ steps. In fact, bounded model checking problem can be reduced to satisfiabilty problem in propositional logic. For each state $s$, the conjunction of the formula corresponding to safe property and the formula $\phi$ implying $s$ is reachable is unsatisifiable. In general, compute $\phi$ is quite expensive. Fortunately, as shown in [21], toghether with properties of Craig's interpolant, an approximation of $\phi$ can be computed quickly. That is a reason why computing interpolant module for propositional logic is implemented in most SAT-based bounded model checking systems.

Last but not least, we shall look at the interpolant theorem from definability perspective where our query rewriting framework follows. Roughly stated, in first order logic, a knowledge base *implicitly defines* a predicate $P$ if in two arbitrary models which have the same domain and extensions for other predicates, the extensions of $P$ are the same. We also say the knowledge base *explicitly defines* $P$ if there is a formula which does not contain $P$ equivalent to $P$ under the knowledge base. About the relationship between implicit and explicit definability, Beth proved that implicit definabilty implies explicit definabilty. In the other words, Beth's theorem is a completeness theorem for definability. Interpolant theorem in this case is a way to prove the Beth's theorem. Moreover, it shows what an explicit definition is. To generalize the relationship between Beth's definability and interpolant property, one can show if a logic has interpolant property then it also has Beth's theorem. Notice that the opposite direction is not true as in some logics, Beth's theorem holds but interpolant theorem does not hold.

### 3.1.1 The theorem

In this section, the formal theorem given by Craig is taken into account. In [9], Craig stated that :

**Theorem 3.1 (Craig's interpolant theorem)** *If $A \rightarrow A'$ and if $A$ and $A'$ have a predicate parameter in common, then there is an 'intermediate' formula $B$ such that $A \rightarrow B$ and $B \rightarrow A'$, and all parameters of $B$ are parameters of both $A$ and $A'$. Also, if $A \rightarrow A'$ and if $A$ and $A'$ have no predicate parameter in common then either $\neg A$ is valid or $A'$ is a valid.*

Notice that the notion of 'predicate parameter' in the theorem is the same with the notion of language. A simple example of Craig's interpolant is following :

**Example 3.1** *$\phi = A \wedge B$, $\psi = A \vee C$. We have: $\phi \rightarrow \psi$ is valid. In this case $A$ is a Craig's interpolant of $\phi$ and $\psi$ because $\phi \rightarrow A$ and $A \rightarrow \psi$ and $A$ occurs in both $\phi$ and $\psi$*

There is also another way to express the theorem using inconsistency of theories. Assume $\Delta$ and $\Gamma$ are two inconsistent theories. Call $\phi$ is the conjunction of all sentences in $\Delta$, $\psi$ is the conjunction of all sentences in $\Gamma$. We have, $\Delta \cup \Gamma$ is inconsistent iff $\phi \rightarrow \neg\psi$ is valid. Therefore, the following theorem is also considered as Craig's interpolant theorem (for convenience, we named it Modified Craig's interpolant theorem) :

**Theorem 3.2 (Modified Craig's interpolant theorem)** *Let $\Delta$ and $\Gamma$ are two inconsistent first order theories which have some common non logical symbols. There is a sentence $\theta$ called Craig interpolant such as $\theta$ is true in $\Delta$ and false in $\Gamma$ and every non logical symbol occurring in $\theta$ occurs in both $\Delta$ and $\Gamma$*

A logic where the interpolation theorem holds is said that has *interpolant property.* Regarding to fragment of first order logic, propositional logic and first order logic with one variable $\mathcal{L}_1$ have interpolant property while the Craig's interpolant theorem fails in $\mathcal{L}_n$ with $1 < n$ ($\mathcal{L}_n$ is first order logic with $n$ variables) and in guarded fragment. A summary [14] on interpolant property of some first order fragments is given in the table 3.1

### 3.1.2 The proof

In general, there are several ways to prove the Craig's interpolant property in a logic such as: model-theoretic approach and proof-theoretic approach. One tries to construct some models having certain expected properties in the former while in the latter, an interpolant is found inductively from the proof of validity or unsatisfiability. In the other words, the first approach is the semantic one and the second approach is the syntactic one. In some proofs, they combine both approaches [14].

| Fragment | Interpolant property |
|---|---|
| First order logic | Yes |
| Propositional logic | Yes |
| $\mathcal{L}_1$ | Yes |
| $\mathcal{L}_n$ $(1 < n < \omega)$ | No |
| Guarded fragment | No |

Table 3.1: Interpolant property in some FO fragments

In this section, first we shall give a proof for propositional logic and then will consider a proof for first order logic. These proofs follow the first approach while several methods to construct an interpolant using second approach will be consider in the next section. Notice that the proof for the propositional logic considered below is also a constructive one even it does not base on the proof of validity.

a) Propositional logic
Assume $\phi \to \psi$ is valid in propositional logic. $A(\phi)$ and $A(\psi)$ are the set of propositional variables in $\phi$ and $\psi$ respectively. We will show there is a propositional formula $\iota$ which is an interpolant of $\phi$ and $\psi$ by induction in the number $n$ of propositional variables which occur in $\phi$ but do not occur in $\psi$.
*Base case*: $n = 0$ means that all propositional variables in $\phi$ is also in $\psi$. Therefore one can choose $\iota = \phi$ because $\phi \to \phi$ and $\phi \to \psi$ are valid. Moreover, $A(\iota) = A(\phi) = A(\phi) \cap A(\psi)$
*Induction hypothesis*: Assume the statement holds for $n = k$, which means that for all $\phi'$ and $\psi'$ such that if the number of propositional variables occur in $\phi'$ but do not occur in $\psi'$ is $k$ and $\phi' \to \psi'$ then there is an interpolant of $\phi'$ and $\psi'$.
*Induction step*: Now we have to prove that the statement holds for the case $n = k{+}1$. Assume $p$ is an variable such that $p \in A(\phi)$ but $p \notin A(\psi)$. Consider the formula $\phi' = \phi[\top/p] \vee \phi[\bot/p]$ where $\phi[\top/p]$ ($\phi[\bot/p]$) is $\phi$ with every occurrence of $p$ replaced by $\top$ ($\bot$). Obviously, $\phi'$ is equivalent to $\phi$ since for every $M$ which is model of $\phi$, if $M \models p$ then $M \models \phi[\top/p]$, if $M \not\models p$ then $M \models \phi[\bot/p]$. Then the following is true :

- (3.1.1)$\phi \to \phi'$
- (3.1.2)$\phi' \to \psi$
- (3.1.3)The number of variables in $\phi'$ but not in $\psi$ is $k$

Apply the induction assumption for (3.1.2)and (3.1.3) we imply there is an interpolant $\iota$ of $\phi'$ and $\psi$. Hence :

- (3.1.4)$\phi' \to \iota$
- (3.1.5)$\iota \to \psi$
- (3.1.6)$A(\iota) \subseteq A(\phi') \cap A(\psi)$

From (3.1.1) and (3.1.4) $\phi \to \iota$ is true while from (3.1.6) $A(\iota) \subseteq A(\phi) \cap A(\psi)$ is true as well. Therefore $\iota$ is an interpolant of $\phi$ and $\psi$

b) First order logic
This proof is a detail version of a proof from [7]. Let $\phi$ and $\psi$ are first order sentences and $\phi \to \psi$ is valid. We will show there is an interpolant of $\phi$ and $\psi$ by refutation. In the other words, we will prove if there is no interpolant then $\phi \wedge \neg\psi$ is consistent by showing a theory containing $\phi$ and $\psi$ is consistent.
Let $L_1$, $L_2$ are languages corresponding to $\phi$ and $\psi$ plus a set $C$ of infinite fresh constants. $L = L_1 \cap L_2$. $T_1$ and $T_2$ are theories in $L_1$ and $L_2$ respectively. We say $T_1$ and $T_2$ are inseparable iff there is no $\phi_0$ in $L$ such that $T_1 \models \phi_0$ and $T_2 \models \neg\phi_0$. Therefore $\{\phi\}$ and $\{\neg\psi\}$ are inseparable because if not, $\phi \models \phi_0(c_1, .., c_n)$, $\phi_0(c_1, .., c_n) \models \psi$ and $c_1, .., c_n \in C$. Consequently, $\forall x_1, ..., x_n \phi_0(x_1, ..., x_n)$ where $x_1, ..., x_n$ do not occurs in $\phi_0$ is an interpolant of $\phi$ and $\psi$, then we have a contradiction with our assumption.

Given a first order language, the set of all formulas in this language is enumerable. Therefore, let $\sigma_0, \sigma_1, ...$ enumerates all sentences in $L_1$ and $\theta_0, \theta_1, ....$ enumerates all sentences in $L_2$. Now we will construct sequences of theories $T_0, T_1, ...$ and $U_0, U_0, ...$ inductively as follows.

- $T_0 = \{\phi\}$; $U_0 = \{\neg\psi\}$
- At each step $i$, construct first :

$$X = \begin{cases} T_i \cup \{\sigma_i\} & \text{if } T_i \cup \{\sigma_i\} \text{ is inseparable from } U_i \\ T_i, & \text{otherwise} \end{cases}$$

$$Y = \begin{cases} U_i \cup \{\theta_i\} & \text{if } U_i \cup \{\theta_i\} \text{ is inseparable from } T_{i+1} \\ U_i, & \text{otherwise} \end{cases}$$

  Then $U_{i+1}$ and $T_{i+1}$ are constructed as :

$$T_{i+1} = \begin{cases} X \cup \{\alpha(c)\} & \text{for some fresh constant } c \in C \text{, if } \sigma_i = \exists x \alpha(x) \in X \\ X, & \text{otherwise} \end{cases}$$

$$U_{i+1} = \begin{cases} Y \cup \{\lambda(c)\} & \text{for some fresh constant } c \in C \text{, if } \theta_i = \exists x \lambda(x) \in Y \\ Y, & \text{otherwise} \end{cases}$$

- $T_\omega = \bigcup_{i<\omega} T_i$ and $U_\omega = \bigcup_{i<\omega} U_i$

Then the following statements are true:

- *$T_i$ and $U_i$ are inseparable for all $i < \omega$*
  We can prove this statement by induction in $i$. At the base case $i = 0$, the statement is obvious true because $\{\phi\}$ and $\{\neg\psi\}$ are inseparable. At induction hypothesis, $T_i$ and $U_i$ are inseparable. Assume that $T_i \cup \{\sigma(c)\}$ and $U_i$ are separable by a formula $\theta$ where $c$ is a new constant and $\exists x\sigma(x) \in T_i$. If $\theta$ does not contain $c$ then $\theta$ also separates $T_i$ and $U_i$, which is a contradiction. If $\theta$ contains $c$ then $\sigma(c) \models \theta(c)$. Then $U_i \models \neg\sigma(c)$ as well. Since $c$ is an arbitrary constant and $c$ is not in $U_i$, $U_i \models \neg\exists x\sigma(x)$. As a consequence, $U_i$ and $T_i$ now are separable by $\exists x\sigma(x)$, which also is a contradiction. The same statement holds when we change the role of $T$ and $U$. Therefore, $T_{i+1}$ and $U_{i+1}$ are inseparable.
- *$T_\omega$ and $U_\omega$ are inseparable*
  Assume that $T_\omega$ and $U_\omega$ are not inseparable. Hence, there is $\theta$ such as $T_\omega \models \theta$ and $U_\omega \models \neg\theta$. Apply the compactness theorem, there is $i < \omega$ such that $T_i \models \theta$ and $U_i \models \neg\theta$. But $T_i$ and $U_i$ are inseparable then it is a contradiction.
- *$T_\omega$ and $U_\omega$ are consistent*
  If $T_\omega$ is inconsistent then $T_\omega \models \neg(x = x)$ while $U_\omega \models x = x$. Then $T_\omega$ and $U_\omega$ are not inseparable. The same argument can be applied for $U_\omega$
- *$T_\omega$ and $U_\omega$ are complete*
  Notice that a theory $T$ is complete in a language if for every formula $\theta$ in this language, $T \models \theta$ or $T \models \neg\theta$. Assume there is $\theta$ in $L_1$ such that neither $T_\omega \models \theta$ nor $T_\omega \models \neg\theta$. We can call $i$ is index of $\theta$ and $j$ is index of $\neg\theta$ in the enumerable above. Then $T_i \cup \{\sigma_i\}$ and $U_i$ are separated by $\theta_1$; $T_j \cup \{\sigma_j\}$ and $U_j$ are separated by $\theta_2$. As a consequence, $T_\omega \models \theta_1 \vee \theta_2$ where $U_\omega \models \neg\theta_1 \wedge \neg\theta_2$. Therefore $T_\omega$ and $U_\omega$ are not inseparable, which is a contradiction for above statement. Analogously, we also have $U_\omega$ are complete
- *$T_\omega \cap U_\omega$ is complete*
  Let $\theta$ be a sentence in $L$. Since both $T_\omega$ and $U_\omega$ are complete, either $T_\omega \models \theta$ or $T_\omega \models \neg\theta$ and either $U_\omega \models \theta$ or $U_\omega \models \neg\theta$. In the other side, since $T_\omega$ and $U_\omega$ are inseparable, if $T_\omega \models \theta$ then $U_\omega \not\models \neg\theta$ and vice versa. Therefore, either $T_\omega \cap U_\omega \models \theta$ or $T_\omega \cap U_\omega \models \neg\theta$ .

- *$T_\omega \cup U_\omega$ is consistent*
  Since $T_\omega$ and $U_\omega$ are consistent, there are $M_1 = (D_1, S_1)$ and $M_2 = (D_2, S_2)$ as their models. Because for each existential sentence, we added one witness in $C$ for it then there always is a case that $D_1 = D_2 = C$. Let $A_1$ and $A_2$ are the signature mapping corresponding to $S_1$ and $S_2$ but just for non-logical symbols in $L$. Since $T_\omega \cap U_\omega$ is complete then we can map $A_1$ to $A_2$ [7] without loss of satisfiability. Therefore we can apply this mapping in order to have a model $M$ of both $T_\omega$ and $U_\omega$. Because $\phi$ is in $T_\omega$ and $\neg\psi$ is in $U_\omega$, $T_\omega \cup U_\omega$ is the theory that we need.

### 3.1.3 Related theorems

From the Craig's interpolant theorem we know that the interpolant exists. Nevertheless, the theorem just says about the language of interpolant and does not mention more about its syntax restriction. Therefore, the problem of finding syntax restriction of the interpolant w.r.t the syntax of input formula is also interesting and plausible in some applications.

Let us consider first a famous theorem introduced by Lyndon

**Theorem 3.3 (Lyndon's interpolation theorem)** *Given $S$ and $T$ are first order theory such as $S \cup T$ is unsatisfiable, then there is a interpolating sentence $\phi$ in the language of $S \cup T$ that is true in all models of $S$ and false in all models of $T$. Moreover, $\phi$ has the stronger property that every relation symbol that has a positive occurrence in $P$ has a positive occurrence in some formula of $S$ and a negative occurrence in some formula of $T$, and every relation symbol with a negative occurrence in $\phi$ has a negative occurrence in some formula of $S$ and a positive occurrence in some formula of $T$*

For a detail proof, readers can refer to the original paper of Lyndon [20]. In the next section, where we consider a constructive method to compute interpolant based on Tableaux, we will also see that it is actually a constructive proof for the theorem.

In some applications, one also tries to find the syntax restriction of interpolant based on the syntax restriction of input theories (formulas). For example, in [23] they show that if $S$ and $T$ are theories such that $S \to T$ then if $S$ is an arbitrary first order theory and $T$ contains only sentences built with $\vee, \wedge, \forall, \exists$ then there is an interpolant built with $\vee, \wedge, \forall, \exists$ as well. Related to this kind of interpolant theorems, [23] is recommended while a summary of results can be referred to 3.2

| S | T | Interpolant | Note |
|---|---|---|---|
| FO | Pos | Pos | FO - first order sentences |
| Pos* | FO | Pos | Pos - Positive sentences (built with $\vee, \wedge, \forall, \exists$) |
| FO | $\forall$ | $\forall$ | $\forall$ - Universal sentences |
| $\forall$* | FO | $\forall$ | * - All function symbols of $S$ also occur in $T$ |
| $\forall$* | Pos | $\forall\vee$ | $\forall\vee$ - Universally quantified disjunctions of atoms |
| FO | $\exists$** | $\exists$ | $\exists$ - Existential sentences |
| $\exists$ | FO | $\exists$ | ** All function symbols of $T$ also occur in $S$ |

Table 3.2: Some interpolant results from [23]

## 3.2 Constructing Craig's interpolant

Our main discussion in this section is concerned with some methods to compute interpolants from proofs based on automated theorem proving techniques such as tableaux and resolution.

### 3.2.1 Tableaux based method

A method to compute interpolant from tableaux proof is proposed by Fitting in [11]. However, Fitting considered only first order logic without equality. In [4], they considered as well some equality rules which are original based on equality rules for constructing interpolants in quantified hybrid logic. We shall follow [4] in this part and prove the correctness of those rules.

Assume we have $\phi \to \psi$ is valid, therefore $\phi \wedge \psi$ is unsatisfiable. From chapter 2, we all know that there is a closed tableau corresponding to $\phi \wedge \psi$. In order to compute an interpolant from this tableau we need to modify it to *biased tableau.*

**Definition 3.4 (Biased tableau)** *A biased tableau for formulas $\phi$ and $\psi$ is a tree $T = (V, E)$ where :*

- *$V$ is a set of nodes, each node is a set of biased formulas. Each biased formula is an expression in the form of $L(\varphi)$ or $R(\varphi)$ where $\varphi$ is a formula. For each node $n$, we call $S(n)$ is the set of biased formulas in $n$*
- *The root of the tree is corresponding to $\{L(\phi), R(\neg\psi)\}$*
- *$E$ is a set of edges. Given 2 nodes $n_1, n_2$, $(n_1, n_2) \in E$ iff there is a biased completion rule from $n_1$ to $n_2$. We say there is a biased completion rule from $n_1$ to $n_2$ if we apply a rule for a biased formula $X(\varphi)$ in $n_1$ and receive $Y(\mu)$ then $S(n_2) = (S(n_1) - \{X(\varphi)\}) \cup \{Y(\mu)\}$.*

The biased completion rules are quite similar to the completion rules in chapter 2 but formulas now will be biased formulas. Call $C$ is the set of all constants in input formulas of the tableau. $C^{par}$ extends $C$ with an infinite set of new constants. A constant is new if it does not occur anywhere in the tableau. We also use $X$ and $Y$ as replacements of $L$ or $R$. With these notations, we have the following rules :

- Propositional rules

| Negation rules | | | $\alpha-$rule | $\beta-$rule |
|---|---|---|---|---|
| $\dfrac{X(\neg\neg\varphi)}{X(\varphi)}$ | $\dfrac{X(\neg\top)}{X(\bot)}$ | $\dfrac{X(\neg\bot)}{X(\top)}$ | $\dfrac{X(\varphi \wedge \psi)}{\begin{array}{c}X(\varphi)\\X(\psi)\end{array}}$ | $\dfrac{X(\neg(\neg\varphi \wedge \neg\psi))}{X(\varphi) \quad \mid \quad X(\psi)}$ |

- First order rules

| $\gamma-$rule | $\sigma-$rule |
|---|---|
| $\dfrac{X(\forall x.\varphi)}{X(\varphi(t))}$ for any $t \in C^{par}$ | $\dfrac{X(\exists x.\varphi)}{X(\varphi(c))}$ for a new constant $c$ |

- Equality rules

| refexivity rule | replacement rule |
|---|---|
| $\dfrac{X(\varphi)}{X(t = t)}$ $t \in C^{par}$ occurs in $\varphi$ | $\dfrac{\begin{array}{c}X(t = u)\\Y(\varphi(t))\end{array}}{Y(\varphi(u))}$ |

A node in the tableau is *closed* if it contains $X(\varphi)$ and $Y(\neg\varphi)$. If a node is closed, we will not apply any rule on it. In the other words, it becomes a leaf of the tree. A branch is closed if it contains a closed node and a tableau is closed if all of its branches are closed. Of course, if the normal tableau is closed then so is the biased tableau and vice versa. Therefore if $\phi \to \psi$ is valid then the biased tableau of $\phi \wedge \neg\psi$ is closed.

Now we are ready to compute interpolant of $\phi$ and $\psi$ using this tableau. For interpolant rules, we use notion $S \xrightarrow{int} I$ where $S = \{L(\phi_1), L(\phi_2), ..., L(\phi_n), R(\psi_1), R(\psi_2), ..., R(\psi_m)\}$ and $I$ is a formula. Given a closed tableau, the interpolant is computed bottom-up based on the following rules :

- Rules for closed branches

r1. $S \cup \{L(\varphi), L(\neg\varphi)\} \xrightarrow{int} \bot$ r2. $S \cup \{R(\varphi), R(\neg\varphi)\} \xrightarrow{int} \top$

r3. $S \cup \{L(\bot)\} \xrightarrow{int} \bot$ r4. $S \cup \{R(\bot)\} \xrightarrow{int} \top$

r5. $S \cup \{L(\varphi), R(\neg\varphi)\} \xrightarrow{int} \varphi$ r6. $S \cup \{R(\varphi), L(\neg\varphi)\} \xrightarrow{int} \neg\varphi$

- Rules for propositional cases

p1. $\dfrac{S \cup \{X(\varphi)\} \xrightarrow{int} I}{S \cup \{X(\neg\neg\varphi)\} \xrightarrow{int} I}$ p2. $\dfrac{S \cup \{X(\top)\} \xrightarrow{int} I}{S \cup \{X(\neg\bot)\} \xrightarrow{int} I}$ p3. $\dfrac{S \cup \{X(\bot)\} \xrightarrow{int} I}{S \cup \{X(\neg\top)\} \xrightarrow{int} I}$

p4. $\dfrac{S \cup \{X(\varphi_1), X(\varphi_2)\} \xrightarrow{int} I}{S \cup \{X(\varphi_1 \wedge \varphi_2)\} \xrightarrow{int} I}$ p5. $\dfrac{S \cup \{L(\varphi_1)\} \xrightarrow{int} I_1 \quad S \cup \{L(\varphi_2)\} \xrightarrow{int} I_2}{S \cup \{L(\neg(\neg\varphi_1 \wedge \neg\varphi_2))\} \xrightarrow{int} I_1 \vee I_2}$

p6. $\dfrac{S \cup \{R(\varphi_1)\} \xrightarrow{int} I_1 \quad S \cup \{R(\varphi_2)\} \xrightarrow{int} I_2}{S \cup \{R(\neg(\neg\varphi_1 \wedge \neg\varphi_2))\} \xrightarrow{int} I_1 \wedge I_2}$

- Rules for first order cases :

f1. $\dfrac{S \cup \{X(\varphi(p))\} \xrightarrow{int} I}{S \cup \{X(\exists x.\varphi(x))\} \xrightarrow{int} I}$ where $p$ is a parameter that does not occur in $S$ or $\varphi$

f2. $\dfrac{S \cup \{L(\varphi(c))\} \xrightarrow{int} I}{S \cup \{L(\forall x.\varphi(x))\} \xrightarrow{int} I}$ if $c$ occurs in $\{\varphi_1, ..., \varphi_n\}$

f3. $\dfrac{S \cup \{R(\varphi(c))\} \xrightarrow{int} I}{S \cup \{R(\forall x.\varphi(x))\} \xrightarrow{int} I}$ if $c$ occurs in $\{\psi_1, ..., \psi_m\}$

f4. $\dfrac{S \cup \{L(\varphi(c))\} \xrightarrow{int} I}{S \cup \{L(\forall x.\varphi(x))\} \xrightarrow{int} \forall x.I[c/x]}$ if $c$ does not occur in $\{\varphi_1, ..., \varphi_n\}$

f5. $\dfrac{S \cup \{R(\varphi(c))\} \xrightarrow{int} I}{S \cup \{R(\forall x.\varphi(x))\} \xrightarrow{int} \exists x.I[c/x]}$ if $c$ does not occur in $\{\psi_1, ..., \psi_m\}$

- Rules for equality cases

e1. $\dfrac{S \cup \{X(\varphi(p)), X(t = t)\} \xrightarrow{int} I}{S \cup \{X(\varphi(p))\} \xrightarrow{int} I}$ e2. $\dfrac{S \cup \{X(\varphi(u)), X(t = u)\} \xrightarrow{int} I}{S \cup \{X(\varphi(t)), X(t = u)\} \xrightarrow{int} I}$

e3. $\dfrac{S \cup \{L(\varphi(u)), R(t = u)\} \xrightarrow{int} I}{S \cup \{L(\varphi(t)), R(t = u)\} \xrightarrow{int} t = u \to I}$ if $u$ occurs in $\varphi(t), \psi_1, ..., \psi_m$

e4. $\dfrac{S \cup \{R(\varphi(u)), L(t = u)\} \xrightarrow{int} I}{S \cup \{R(\varphi(t)), L(t = u)\} \xrightarrow{int} t = u \wedge I}$ if $u$ occurs in $\varphi(t), \psi_1, ..., \psi_m$

e5. $\dfrac{S \cup \{L(\varphi(u)), R(t = u)\} \xrightarrow{int} I}{S \cup \{L(\varphi(t)), R(t = u)\} \xrightarrow{int} I[u/t]}$ if $u$ does not occur in $\varphi(t), \psi_1, ..., \psi_m$

e6. $\dfrac{S \cup \{R(\varphi(u)), L(t = u)\} \xrightarrow{int} I}{S \cup \{R(\varphi(t)), L(t = u)\} \xrightarrow{int} I[u/t]}$ if $u$ does not occur in $\varphi(t), \psi_1, ..., \psi_m$

Let $\phi$ and $\psi$ as above and $T$ is their biased tableau. Since $\phi \wedge \neg\psi$ is unsatifiable then the biased tableau $T$ is closed. Call $S$ is the root of the tableau, we have the following theorem :

**Theorem 3.5 (Correctness of interpolant rules)** *If $S \xrightarrow{int} I$ then $I$ is an interpolant of $\phi$ and $\psi$*

**Proof**
We will prove the theorem by induction on the shape of the biased tableau. In the other words, we will show that at each node correponding to $S' = \{L(\phi_1), ..., L(\phi_n), R(\psi_1), ..., R(\psi_m)\}$ and $S' \xrightarrow{int} I$ then $I$ is an interpolant of $L(S') = \phi_1 \wedge ... \wedge \phi_n$ and $R(S') = \neg\psi_1 \vee ....\neg\psi_m$
First, let us check base cases. For r1 rule, since $S'$ contains $L(\varphi)$ and $L(\neg\varphi)$ then $L(S') \equiv \bot$ and of course $\bot$ is an interpolant of $\bot$ and an arbitrary formula. r2, r3, r4 can be proved analogously. For r5 rule, we have $L(S') = \varphi \wedge ...$ and $R(S') = \varphi \vee ....$ Then, $L(S') \to \varphi$, $\varphi \to R(S')$ and $\varphi$ is in both left and right side. Therefore, $\varphi$ is an interpolant of $L(S')$ and $R(S')$. The rule r6 can be proved similarly.
Now let's move to propositional cases:

- For p1 rule, since $I$ is an interpolant of $L(S \cup \{X(\varphi)\})$ and $R(S \cup \{X(\varphi)\})$ then $I$ is an interpolant of $L(S \cup \{X(\neg\neg\varphi)\})$ and $R(S \cup \{X(\neg\neg\varphi)\})$ because $\varphi \equiv \neg\neg\varphi$. Analogously p2, p3 hold.

- For p4 rule, if $X$ is $L$, since $L(S) \wedge \varphi_1 \wedge \varphi_2 \equiv L(S) \wedge (\varphi_1 \wedge \varphi_2)$ then the interpolant is preserved. If $X$ is $R$, we also have $R(S) \vee \neg\varphi_1 \vee \neg\varphi_2 \equiv R(S) \vee \neg(\varphi_1 \wedge \varphi_2)$ then the interpolant is preserved as well.

- For the rule p5, $I_1$ is interpolant of $L(S) \wedge \varphi_1$ and $R(S)$ and $I_2$ is interpolant of $L(S) \wedge \varphi_2$ and $R(S)$. Then $L(S) \wedge (\varphi_1 \vee \varphi_2) \to I_1 \vee I_2$ and $I_1 \vee I_2 \to R(S)$. Since constants, functions and relations symbol in $I_1$ are in $L(S) \wedge \varphi_1$, constants, functions and relations symbol in $I_2$ are in $L(S) \wedge \varphi_2$, then all constants, functions and relations symbol in $I_1 \vee I_2$ should be in $L(S) \wedge (\varphi_1 \vee \varphi_2)$. Therefore $I_1 \vee I_2$ is an interpolant of $L(S) \wedge (\varphi_1 \vee \varphi_2)$ and $R(S)$

- For the rule p6, $I_1$ is interpolant of $L(S)$ and $R(S) \vee \neg\varphi_1$, $I_2$ is interpolant of $L(S)$ and $R(S) \vee \neg\varphi_2$. Then $L(S) \to I_1 \wedge I_2$, $(I_1 \wedge I_2) \to (R(S) \vee \neg\varphi_1) \wedge (R(S) \vee \neg\varphi_2)$ which is equivalent to $(I_1 \wedge I_2) \to R(S) \vee \neg(\varphi_1 \vee \varphi_2)$. Then $I_1 \wedge I_2$ is an interpolant of $L(S)$ and $R(S) \vee \neg(\varphi_1 \vee \varphi_2)$ because the language of $I_1 \wedge I_2$ is in the languages of both $L(S)$ and $R(S) \vee \neg(\varphi_1 \vee \varphi_2)$.

Related to first order cases, we have :

- For f1 rule, if $X = L$ then suppose $I$ is an interpolant of $L(S) \wedge \varphi(p)$ and $R(S)$ where $p$ is not in $\varphi$ and $S$. Assume $M$ is a model of $L(S) \wedge \exists x\varphi(x)$ but $M$ is not a model of $I$ and under $M$, $x$ is assigned to $a$ in the domain of $M$. Since $p$ is not in $I$ so the extension $M'$ of $M$ such that $p$ is assigned to $a$ is not a model of $I$ either. But $M'$ is a model of $L(S) \wedge \varphi(p)$ then we have the contradiction. Moreover, it is trivial to see that the language $I$ is in both $L(S) \wedge \exists x\varphi(x)$ and $R(S)$. Therefore $I$ is an interpolant of $L(S) \wedge \exists x\varphi(x)$ and $R(S)$. If $X = R$, we have $I$ is an interpolant of $L(S)$ and $R(S) \wedge \neg\varphi(p)$. Since $p$ is an arbitrary constant that is not in $S$ or $\varphi$ then $I \to R(S) \wedge \forall x \neg\varphi(x)$ as well. Then $I \to R(S) \wedge \neg\exists x\varphi(x)$. Analogously, $I$ is an interpolant of $L(S)$ and $R(S) \wedge \neg\exists x\varphi(x)$.

- The f2 rule is quite trivial because we always have $\forall x\varphi(x) \to \varphi(c)$

- For f3 rule, suppose $I$ is an interpolant of $L(S)$ and $R(S) \vee \neg\varphi(c)$. Since $\forall x\varphi(x) \to \varphi(c)$ then $\neg\varphi(c) \to \neg\forall x\varphi(x)$. Consequently $I \to (R(S) \vee \neg\forall x\varphi(x))$. Therefore $I$ and interpolant of $L(S)$ and $R(S) \vee \neg\forall x\varphi(x)$ because even if $I$ contains $c$ then $c$ is in the common language between $L(S)$ and $R(S) \vee \neg\forall x\varphi(x)$ (because $c$ occurs in $\{\varphi_1, ..., \varphi_n\}$)

- For f4 rule, assume $I$ is an interpolant of $L(S) \wedge \varphi(c)$ and $R(S)$, we have to prove $\forall x I[c/x]$ is an interpolant of $L(S) \wedge \forall x\varphi(x)$ and $R(S)$. First, $\forall x I[c/x] \to I$, then $\forall x I[c/x] \to R(S)$ (1). Second, since $L(S) \wedge \varphi(c) \to I$ then $\neg I \to \neg(L(S) \wedge \varphi(c))$. Consequently, $\exists x(\neg I[c/x]) \to \exists x(\neg(L(S) \wedge \varphi(c)))$. Since $c$ is not in $R(S)$ then: $\neg\forall x I[c/x] \to \neg(L(S) \wedge \forall x\varphi(x))$. Therefore $L(S) \wedge \forall x\varphi(x) \to \forall x I[c/x]$ (2). From (1) and (2) we have $\forall x I[c/x]$ is an interpolant of $L(S) \wedge \forall x\varphi(x)$ and $R(S)$ because it is trivial to see the languague of $\forall x I[c/x]$ is in the common language of $L(S) \wedge \forall x\varphi(x)$ and $R(S)$.

- Rule f5 can be proved analogously as rule f4.

For equality rules we have :

- The rule e1, e2 are trivial
- For rule e3, suppose $I$ an interpolant of $L(S) \wedge \varphi(u)$ and $R(S) \vee \neg(t = u)$, we have to prove that $t = u \to I$ is an interpolant of $L(S) \wedge \varphi(t)$ and $R(S) \vee \neg(t = u)$ where $u$ occurs in $\varphi(t)$ or $L(S)$. First, call $M$ is a model of $L(S) \wedge \varphi(t)$. If $t = u$ is true under $M$, $M$ is also a model of $L(S) \wedge \varphi(u)$, then $M \models I$ and finally $M \models t = u \to I$. If $t = u$ is false under $M$ then $M \models t = u \to I$. Therefore $M$ always is a model of $t = u \to I$. As a consequence, we have $L(S) \wedge \varphi(t) \to t = u \to I$ is valid (3). Second, call $M'$ is a model of $t = u \to I$. If $M' \models t = u$ then $M' \models I$. But $I \to R(S) \vee \neg(t = u)$ is valid, so $M'$ is a model of $R(S) \vee \neg(t = u)$. If $t = u$ is false in $M'$ then $M'$ is also a model of $R(S) \vee \neg(t = u)$ (4). Moreover, since the language of $I$ is in the common language of $L(S) \wedge \varphi(u)$ and $R(S) \vee \neg(t = u)$ and $u$ occurs in $\varphi(t)$ or $L(S)$ then the languague of $t = u \to I$ is in the common language of $L(S) \wedge \varphi(t)$ and $R(S) \vee \neg(t = u)$(5). From (3)(4)(5), we have $t = u \to I$ is an interpolant of $L(S) \wedge \varphi(t)$ and $R(S) \vee \neg(t = u)$.
- Rule e4 can be proved analogously as rule e3.
- For rule e5, suppose $I$ an interpolant of $L(S) \wedge \varphi(u)$ and $R(S) \vee \neg(t = u)$, we have to prove $I[u/t]$ is an interpolant of $L(S) \wedge \varphi(t)$ and $R(S) \vee \neg(t = u)$ where $u$ does not occur in $\varphi(t)$ or $L(S)$. First, since $u$ does not occur in $\varphi(t)$ or $L(S)$ if $L(S) \wedge \varphi(u) \to I$ is valid, $L(S) \wedge \varphi(t) \to I[u/t]$ is valid as well(6). Second, call $M$ is an model of $I[u/t]$, if $u = t$ is true under $M$ then so is $I$. Since $I \to R(S) \vee \neg(t = u)$ valid then $R(S) \vee \neg(t = u)$ is true under $M$ as well. If $u = t$ is false under $M$ then $R(S) \vee \neg(t = u)$ is still true under $M$. Therefore $I[u/t] \to R(S) \vee \neg(t = u)$ is valid(7). Since $t$ is in both $L(S) \wedge \varphi(t)$ and $R(S) \vee \neg(t = u)$ then the language of $I[u/t]$ is in the common language of $L(S) \wedge \varphi(t)$ and $R(S) \vee \neg(t = u)$ (8). From (6)(7)(8), we have $I[u/t]$ is an interpolant of $L(S) \wedge \varphi(t)$ and $R(S) \vee \neg(t = u)$
- Rule e6 can be proved analogously as rule e5.

■

### 3.2.2 Interpolation for $\mathcal{ALC}$ concepts

In this section, we shall look at a tableau based method to compute interpolant of two $\mathcal{ALC}$ concepts $C$ and $D$ such as $C \sqsubseteq D$ is valid under a $TBox$ $\mathcal{T}$ where the interpolant is defined as follows :

**Definition 3.6 (Interpolant for $\mathcal{ALC}$ with Tboxes)** *Given a Tbox $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ and two concepts $H$ and $J$ such that $H \sqsubseteq_{\mathcal{T}} J$. A concept $I$ is called interpolant of $H$ and $J$ under $\mathcal{T}$ iff $H \sqsubseteq_{\mathcal{T}} I$, $I \sqsubseteq_{\mathcal{T}} J$ and $\Sigma(I) \subseteq \Sigma(H, \mathcal{T}_1) \cap \Sigma(J, \mathcal{T}_2)$ where $\Sigma(I)$ is the set of all atomic concept names and atomic role names in $I$ and $\Sigma(H, \mathcal{T}_1)$ ($\Sigma(J, \mathcal{T}_2)$) is the set of all atomic concept names and atomic role names in $H$ ($J$) or $\mathcal{T}_1$ ($\mathcal{T}_2$)*

In order to show the interpolant always exists, we shall describe a method proposed in [26] where the similar idea of improving normal tableau to biased tableau is applied. According to this method, first all axioms $C \sqsubseteq D$ in Tbox $\mathcal{T}$ are transformed to $\top \sqsubseteq \neg C \sqcup D$ and then we can consider $\mathcal{T}$ as a set of universal concepts. Given an infinite set of variables names $N_V$ and $<$ is a well-order relation on $N_V$, we define:

- A *labeled concept* is an expression of the form $C^\lambda$ where $\lambda \in \{l, r\}$ and $C$ is a concept.
- An expression of the form $(x : D)^\lambda$ is a *biased contraint* where $D$ is a concept, $x \in N_V$ and $\lambda \in \{l, r\}$.
- A finite, non-empty set $S$ of biased constraints is an a *biased constraint system.* A varibale $x \in N_V$ is fresh for $S$ if $x$ does not occur in $S$ and for all $y$ in $S$, $y < x$. If a variable $x$ occurs in $S$ then $S$ also contains $(x : \top)^l$ and $(x : \top)^r$

- For any $x$ in $S$, the set of $x - constraints$ of $S$ (denoted by $x : C \in S$) is the set of all constrains in $S$ in the form of $(x : D)^\lambda$
- For a role name $R$, $succ(x, R, S) = \{C^\lambda | (x : \forall R.C)^\lambda \in S\}$
- If $X$ is a set of labeled concept then $x : X$ is a shorter way to express the set of biased constraints $\{(x : C)^\lambda | C^\lambda \in X\}$
- A biased contrains system $S$ is a *clash* if there are variable $x$ and concept name $A$, $\{(x : A)^\lambda, (x : \neg A)^\kappa\} \in S$.

| | |
|---|---|
| $R_\sqcap$ rule | |
| Condition : | $(x : C_1 \sqcap C_2)^\lambda \in L(g)$ and $\{(x : C_1)^\lambda, (x : C_2)^\lambda\} \not\subseteq L(g)$ |
| Effect : | $L(g') = L(g) \cup \{(x : C_1)^\lambda, (x : C_2)^\lambda\}$ |
| $R_\sqcup$ rule | |
| Condition : | $(x : C_1 \sqcup C_2)^\lambda \in L(g)$ and $\{(x : C_1)^\lambda, (x : C_2)^\lambda\} \cap L(g) = \emptyset$ |
| Effect : | $L(g') = L(g) \cup \{(x : C_1)^\lambda\}$ and $L(g'') = L(g) \cup \{(x : C_2)^\lambda\}$ |
| $R_\exists$ rule | |
| Condition : | $(x : \exists R.C)^\lambda \in L(g)$ and there is no variable $y$ in $L(g)$ such that $y : \{C^\lambda\} \cup succ(x, R, S) \subseteq L(g)$ |
| Effect : | $L(g') = L(g) \cup \{(x : C_1)^\lambda\} \cup y : \{C^\lambda\} \cup succ(x, R, S) \cup \mathcal{T}$ where $y$ fresh for $S$ |

Table 3.3: Biased completion rules for $\mathcal{ALC}$

Given two concept $H$ and $J$ and a tbox $\mathcal{T}$ is the set of univeral concepts, $\mathcal{T}^l$ is the set of labeled $l$ concepts in $\mathcal{T}$ and $\mathcal{T}^r$ is the set of labeled $l$ concepts in $\mathcal{T}$. A biased tableau for $\langle H \sqcap \neg J, \mathcal{T}\rangle$ is a finite tree $(V, E)$ with labelling function $L$ such as :

- L maps each node $v \in V$ into a biased constrain system $L(v)$
- $L(v_0)$ is corresponding with the biased constrain system $x_0 : \{H^l, (\neg J)^r\} \cup T^l \cup T^r$ where $v_0$ is the root and $x_0$ is the initial variable.
- L maps each $(v_1, v_2) \in E$ to a biased completion rule from table 3.3

$$\text{na1.}\frac{(x:A)^l, (x:\neg A)^l \in L(g)}{int(g)(x) := \bot}$$
$$\text{na2.}\frac{(x:A)^r, (x:\neg A)^r \in L(g)}{int(g)(x) := \top}$$
$$\text{na3.}\frac{(x:A)^l, (x:\neg A)^r \in L(g)}{int(g)(x) := A}$$
$$\text{na4.}\frac{(x:A)^r, (x:\neg A)^l \in L(g)}{int(g)(x) := \neg A}$$

Table 3.4: Closed branch rules for $\mathcal{ALC}$

As we see, given a root node, the tableau will be built by applying these rules repeatedly. In this case, we just consider the building strategy such as the priority of the rules is $R_\sqcap > R_\sqcup > R_\exists$. For each node $v$, the bulding process will finish if there is no more rule can be applied or if $L(v)$ is a clash. The tableau is said closed if all its branches are closed. In the case that $H$ and $J$ fulfill the condition in the above definition where $\mathcal{T}^l = \{C^l | C \in \mathcal{T}_1\}$ and $\mathcal{T}^r = \{C^r | C \in \mathcal{T}_2\}$, the tableau is closed because $H \sqcap \neg J$ is unsatifiable under $\mathcal{T}$.

In the next step, we shall construct the interpolant in the definition based on the closed tableau. Given a node $g$ in the tableau, let $\{(x : C_0)^l, ..., (x : C_n)^l, (x : D_0)^r, ..., (x : D_m)^r\}$ is $x - constraints$ in $L(g)$, we have :

$$\text{pa1.}\ \frac{\begin{array}{c}(x : C_1 \sqcap C_2)^\lambda \in L(g)\\ L(g') = L(g) \cup \{(x : C_1)^\lambda, (x : C_2)^\lambda\}\end{array}}{int(g) := int(g')}$$

$$\text{pa2.}\ \frac{\begin{array}{c}(x : C_1 \sqcup C_2)^l \in L(g)\\ L(g') = L(g) \cup \{(x : C_1)^l\}\\ L(g'') = L(g) \cup \{(x : C_2)^l\}\end{array}}{int(g) := int(g') \sqcup int(g'')}$$

$$\text{pa3.}\ \frac{\begin{array}{c}(x : C_1 \sqcup C_2)^r \in L(g)\\ L(g') = L(g) \cup \{(x : C_1)^r\}\\ L(g'') = L(g) \cup \{(x : C_2)^r\}\end{array}}{\begin{array}{c}int(g)(y) := int(g')(y) \sqcup int(g'')(y)(y \neq x)\\ int(g)(x) = int(g')(x) \sqcap int(g'')(x)\end{array}}$$

Table 3.5: Propositional rules for $\mathcal{ALC}$

- A concept $I$ is an interpolant of this $x - constraints$ if $I$ is an interpolant of $C_0 \sqcap ... \sqcap C_n$ ($\top$ if $n = 0$) and $\neg D_0 \sqcup .... \sqcup D_m$ ($\bot$ if $m = 0$) under $\mathcal{T}$.
- $int(g)$ is the set of all interpolants corresonding to all variables in $g$. $int(g)(x)$ is the interpolant corresponding to $x$.
- $int(g) = int(g_1) \sqcup int(g_2)$ ($int(g) = int(g_1) \sqcap int(g_2)$) means that for every variable $x$ in $g$, $int(g)(x) = int(g_1)(x) \sqcup int(g_2)(x)(int(g)(x) = int(g_1)(x) \sqcap int(g_2)(x))$. Notice that the relation $\sqcup$ and $\sqcap$ between two concepts can be applied here for a special concept $null$ where $C \sqcup null = null \sqcup C = C$ and $C \sqcap null = null \sqcap C = C$.

$$\text{fa1.}\ \frac{\begin{array}{c}(x : \forall R.C)^l \in L(g)\\ L(g') = L(g) \cup (y : \{C^l\} \cup X)\end{array}}{\begin{array}{lr}int(g)(z) := int(g')(z) & z \neq x, z \neq y\\ int(g)(y) := null & I := int(g')(y)\\ int(g)(x) := int(g')(x) & \text{if } I = null\\ int(g)(x) := int(g')(x) \sqcup \bot & \text{if } I = \bot\\ int(g)(x) := int(g')(x) \sqcup \exists R.I & \text{otherwise}\end{array}}$$

$$\text{fa2.}\ \frac{\begin{array}{c}(x : \forall R.C)^r \in L(g)\\ L(g') = L(g) \cup (y : \{C^r\} \cup X)\end{array}}{\begin{array}{lr}int(g)(z) := int(g')(z) & z \neq x, z \neq y\\ int(g)(y) := null & I := int(g')(y)\\ int(g)(x) := int(g')(x) & \text{if } I = null\\ int(g)(x) := \top & \text{if } I = \top\\ int(g)(x) := int(g')(x) \sqcup \forall R.I & \text{otherwise}\end{array}}$$

Table 3.6: First order rules for $\mathcal{ALC}$

Then for each node $g$ in the tableau, $int(g)$ is computed based on if-then rules in tables: 3.4, 3.5, 3.6. Notice that for a variable $x$, if these rules do not assign a value for $int(g)(x)$ then $int(g)(x)$ is assumed to be $null$.

Related to the soundness of these rules, we have the following lemma

**Lemma 3.7** *For every node $g$ in the tableau, $x$ is a variable in $L(g)$, if $int(g)(x)$ is not null, it is an interpolant for the $x - constraints$ of $L(g)$.*

**Proof**
Sketch. In order to prove the theorem, one can use induction and then verify the soundness

$$\text{nc1.}\ \frac{(x:C)^l, (x:\neg C)^l \in L(g)}{int(g)(x) := \bot}$$
$$\text{nc2.}\ \frac{(x:C)^r, (x:\neg C)^r \in L(g)}{int(g)(x) := \top}$$
$$\text{nc3.}\ \frac{(x:C)^l, (x:\neg C)^r \in L(g)}{int(g)(x) := C}$$

Table 3.7: Non-atomic closed branch rules for $\mathcal{ALC}$

of each rule. In [26], they showed the soundness of fa2. Now we will show the soundness of pa3. Let us consider the case $y \neq x$ first. Assume $int(g')(y)$ and $int(g'')(y)$ are interpolants of $y - constraints$ in $L(g')$ and $L(g'')$. Since there is no difference among $y - constraints$ of $L(g)$, $L(g')$ and $L(g'')$ then $int(g)(y) = int(g')(y) \sqcup int(g'')(y) = int(g')(y) = int(g'')(y)$ is interpolant of $y - constraints$ in $L(g)$. Second, for the $x$ case, assume $I_1 = int(g')(x)$ and $I_2 = int(g'')(x)$ are interpolants of $x - constraints$ in $L(g')$ and $L(g'')$. Let $\{(x : D_0)^l, ..., (x : D_n)^l, (x : E_0)^r, ..., (x : E_m)^r\}$ is $x - constraints$ in $L(g)$. We have $I_1$ is an interpolant of $D_0 \sqcap ... \sqcap D_n$ and $\neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg C_1$ and $I_2$ is an interpolant of $D_0 \sqcap ... \sqcap D_n$ and $\neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg C_2$. Then $(D_0 \sqcap ... \sqcap D_n) \sqsubseteq_{\mathcal{T}} I_1 \sqcap I_2$ (1) and $I_1 \sqcap I_2 \sqsubseteq_{\mathcal{T}} (\neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg C_1) \sqcap (\neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg C_2)$. Since $(\neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg C_1) \sqcap (\neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg C_2) \equiv \neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg(C_1 \sqcup C_2)$ then $I_1 \sqcap I_2 \sqsubseteq_{\mathcal{T}} \neg E_0 \sqcup ... \sqcup \neg E_m \sqcup \neg(C_1 \sqcup C_2)$ (2). Moreover, since $L(g)$ contains both $C_1$ and $C_2$ then $\Sigma(I_1 \sqcap I_2)$ fulfills the condition for interpolant(3). From (1)(2)(3) we have expected statement. ■

| | |
|---|---|
| $R_{\sqcap 1}$ rule | |
| Condition : | $(x : A)^\lambda \in L(g)$, $A \equiv C$ is in $\mathcal{T}_U$ (or $\mathcal{T}'_U$) and $(x : C)^l \notin L(g)$ (or $(x : C)^r \notin L(g)$ ) |
| Effect : | $L(g') = L(g) \cup \{(x : C)^l\}$ (or $\{(x : C)^r\}$ |
| $R_{\sqcup 1}$ rule | |
| Condition : | $(x : \neg A)^\lambda \in L(g)$, $A \equiv C$ is in $\mathcal{T}_U$ (or $\mathcal{T}'_U$) and $(x : \neg C)^l \notin L(g)$ (or $(x : \neg C)^r \notin L(g)$ ) |
| Effect : | $L(g') = L(g) \cup \{(x : \neg C)^l\}$ (or $\{(x : \neg C)^r\}$ |
| $R_{\exists 1}$ rule | |
| Condition : | $(x : \exists R.C)^\lambda \in L(g)$ and $A \sqsubseteq C$ is in $\mathcal{T}_U$ (or $\mathcal{T}'_U$), and $(x : C)^l \notin L(g)$ (or $(x : C)^r \notin L(g)$ ) |
| Effect : | $L(g') = L(g) \cup \{(x : C)^l\}$ (or $\{(x : C)^r\}$ ) |

Table 3.8: Lazy unfolding completion rules for $\mathcal{ALC}$

For the correctness of the method we have :

**Lemma 3.8** *Given $H, J$ and $\mathcal{T}$ as in the definition 3.6, $g_0$ is the root of closed tableau corresponding to them and $x_0$ is the initial variable then $int(g_0)(x_0)$ is not null.*

**Proof**

In order to prove it, first we will show for two variable $x$ and $y$ in the tableau, if $x < y$, $int(g')(y) \neq null$ and $g'$ is an descendent of $g$ then $int(g)(x) \neq null$ by induction on the number $k$ of variables between $x$ and $y$. For the base case $k = 0$ then $g'$ is a child of $g$ where in $L(g)$ there is $(x : \exists R.C)^\kappa$. Then $int(g)(x)$ is computed based on one of 2 rules : fa1 and fa2. Since $int(g')(y) \neq null$, base on the conditions of these rules we have $int(g)(x) \neq null$. At induction step, assume that the statement holds for $k = n$, we have to prove that it holds for $k = n + 1$. Call $x_1 > ... > x_n > x_{n+1}$ are variables between $y$ and $x$. Then $int(g_n)(x_n)$ is not $null$. Based on the same argument as in the base case, $int(g)(x) \neq null$. For the case $x = y$, the statement trivialy holds. Under the construction we have $x_0$ is the smallest variable and $g_0$ is the biggest ancestor in the tableau then $int(g_0)(x_0)$ is not null because for all variable $y$ which causes the clash in $g$ then $int(g)(y) \neq null$ ■

Related to some optimization techniques mentioned in section 2.3, a set of corresponding rules for non-atomic closed branch and lazy unfolding is introduced in the paper [26] as well. In the case of non-atomic closed branch, the biased completion rules are the same but only the interpolant calculation rules for clashes need to be motified as in 3.7. In the case of lazy unfolding, the biased completion rules need to expand with the rules mentioned in 3.8, where the interpolant calculation rules correponding to them are introduced in 3.9

$$\text{la1.}\ \frac{\begin{array}{c}(x:A)^{\lambda} \in L(g)\\ A \equiv C \in \mathcal{T}_U(or\mathcal{T}'_U)\\ L(g') = L(g) \cup \{(x:C)^{\kappa}\}\end{array}}{int(g) := int(g')}$$

$$\text{la2.}\ \frac{\begin{array}{c}(x:\neg A)^{\lambda} \in L(g)\\ A \equiv C \in \mathcal{T}_U(or\mathcal{T}'_U)\\ L(g') = L(g) \cup \{(x:\neg C)^{\kappa}\}\end{array}}{int(g) := int(g')}$$

$$\text{la3.}\ \frac{\begin{array}{c}(x:\neg A)^{\lambda} \in L(g)\\ A \sqsubseteq C \in \mathcal{T}_U(or\mathcal{T}'_U)\\ L(g') = L(g) \cup \{(x:C)^{\kappa}\}\end{array}}{int(g) := int(g')}$$

Table 3.9: Lazy unfolding interpolant rules for $\mathcal{ALC}$

### 3.2.3 Resolution based methods

In order to apply interpolation for model checking, several methods to compute interpolant for propositional logic from the resolution proof has been proposed in [21] [24]. In this section, instead of considering these specific methods, we shall follow a general method introduced by Huang [16] where Craig interpolant formulas are constructed from completeness resolution for first order logic. However, in Huang's paper, the difference between the clause form of a sentence and the sentence ifseft is not considered and makes readers confused. Hence, we shall clarify the difference in each definition or theorem as well

From section 2.2, one can see that given two sentences $\phi$ and $\psi$ such as $\phi \wedge \psi$ is unsatifiable, $\Sigma$ is the set of clauses of $\phi$'s clause form, $\Pi$ is the set of clauses of $\psi$'s clause form, then there is a way to apply resolution rule, factoring rule and paramodulation rule repeatly in order to generate the empty clause from $\Sigma$ and $\Pi$. We name it as 'resolution proof'.

**Definition 3.9 (Resolution proof)** *Given two set of clauses $\Sigma$ and $\Pi$ where $\Sigma \cup \Pi$ is unsastifiable, a resolution proof $P$ corresponding to $\Sigma$ and $\Pi$ is a directed acyclic graph $(V, E)$ with labelling function $L$ where :*

- *$V$ is the set of nodes where each node is corresponding to a clause*
- *For each node $v \in V$:*
  - *$v \in \Sigma$ or $v \in \Pi$, $v$ is called a leaf. $L(v)$ is corresponding to $\Sigma$ or $\Pi$*
  - *$v$ is result of applying : resolution rule, paramodulation for $v_1$ or $v_2$ or factoring rule for $v_1$ where $v_1, v_2$ is in $V$. $L(v)$ is corresponding to the names of these rules.*
- *$(v_1, v)$ and $(v_2, v)$ is in $E$ if $v$ is the resolvent or paramodulant of $v_1$ and $v_2$. $(v_1, v) \in E$ if $v$ is the factor of $v_1$.*
- *The empty clause is the unique root of the graph*

For each literal $L$ in the resolution proof $P$, we say $L$ is *from* $\Sigma(\Pi)$ if its relation symbols occurs in $\Sigma(\Pi)$, $L$ is from $\Sigma(\Pi)$ *alone* if $L$ is from $\Sigma(\Pi)$ but not from $\Pi(\Sigma)$. For a term $t$ in $P$, we say $t$ is *noncommon term* if $t$ begins with a symbol which is not in the common language of $\Sigma$ and $\Pi$. A noncommon term is called $\Sigma - term$ ($\Pi - term$) if its initial symbol is from $\Sigma$ ($\Pi$). An occurrence of a $\Sigma - term$ ($\Pi - term$) is said *maximal* if it is not a subterm of a $\Sigma - term$ ($\Pi - term$).

We need to compute the interpolant (according to Modified Interpolant Theorem) of $\phi$ and $\psi$ such as $\phi \wedge \psi$ is unsatifiable base on resolution proof of $\Sigma$ and $\Pi$ where $\Sigma$ and $\Pi$ are their clause forms.

For each vertex $v \in V$, we call $s(v)$ is *local interpolant* of $v$ and $s(v)$ is computed as follows :

- Input cases
  If $v$ is a leaf and $L(v) = \Sigma$ then (i1.)$s(v) = \bot$. In the case $L(v) = \Pi$ then (i2.)$s(v) = \top$
- Resolvent cases
  If $v$ is the resolvent of $v_1 = L \vee C$ and $v_2 = D \vee \neg L'$ and $\pi$ is m.g.u of $L$ and $L'$ then :
  - (res1.)$s(v) = (s(v_1) \vee s(v_2))\pi$ if $L$ is from $\Sigma$ alone
  - (res2.)$s(v) = (s(v_1) \wedge s(v_2))\pi$ if $L$ is from $\Pi$ alone
  - (res3.)$s(v) = ((\neg L' \wedge s(v_1)) \vee (L \wedge s(v_2)))\pi$ otherwise
- Factor cases
  If $v$ is the factor of $v_1 = L \vee L' \vee C$ and $\pi$ is m.g.u of $L$ and $L'$ then (fac1)$s(v) = (s(v_1))\pi$
- Paramodulant cases
  If $v$ is the paramodulant of $v_1 = C(r)$ and $v_2 = (s = t \vee D)$ and $\pi$ is m.g.u of $r$ and $s$ then :
  - (para1.)$s(v) = ((s(v_1) \wedge s = t) \vee (s(v_2) \wedge s \neq t))\pi \vee (s = t \wedge h(s) \neq h(t))\pi$ if $r$ occurs in $C(r)$ as a subterm of a maximal $\Pi - term$ $h(r)$ and there is more than one occurence of $h(r)$ in $C(r) \vee s(v_1)$
  - (para2.)$s(v) = ((s(v_1) \wedge s = t) \vee (s(v_2) \wedge s \neq t))\pi \vee (s \neq t \wedge h(s) = h(t))\pi$ if $r$ occurs in $C(r)$ as a subterm of a maximal $\Sigma - term$ $h(r)$ and there is more than one occurence of $h(r)$ in $C(r) \vee s(v_1)$
  - (para3.)$s(v) = (s(v_1) \wedge s = t) \vee (s(v_2) \wedge s \neq t))\pi$ otherwise.

We have the following theorem :

**Theorem 3.10** *For each node $v$ in resolution proof $P$ of $\Sigma$ and $\Pi$ , $s(v)$ fulfills the following conditions :*

1. *All relational symbols of $s(v)$ are in the common language of $\Sigma$ and $\Pi$ where $\vec{x}$ are free variables in $\theta$*
2. $\Sigma \models^* s(v) \vee v$
3. $\Pi \models^* \neg s(v) \vee v$

**Proof**
Sketch. One can prove this theorem base on induction in the shape of $P$ where the correctness of each rule is proved detailly in Huang's paper. Notice that $\Sigma, \Pi, s(v), v$ here may be contain free variables so actually the relation $\models^*$ here is slightly different from normal $\models$ relation such as $A \models^* B$ means $\forall \vec{x} A(x) \models \forall \vec{y} B(y)$ where $\vec{x}, \vec{y}$ are free variables in $A, B$. ■

**Theorem 3.11** *Given $v_0$ is the root node of resolution proof $P$ of $\Sigma$ and $\Pi$ and $s(v_0) = \phi(t_1, ..., t_n)$ where $t_i$ is the maximal $\Sigma - term(\Pi - term)$ in $\phi$ then $\theta = Q_1 x_1 ... Q_n x_n \phi(x_1, ..., x_n)$ where $Q_i = \exists$ if $t_i$ is $\Sigma - term$, $Q_i = \forall$ if $t_i$ is $\Pi - term$ fulfills the following conditions :*

- *The language of $\theta$ is in the common language of $\Sigma$ and $\Pi$*
- $\Sigma \models^* \theta$
- $\Pi \models^* \neg\theta$

In order to prove this theorem, Huang first used lifting theorem to transform the resolution proof to the propositional one and then prove two last conditions by induction on the shape of propositional proof.

Now consider standard Craig interpolant between $\phi$ and $\psi$, we have the following theorem

**Theorem 3.12** *Given $\theta$ as in theorem 3.11, $\forall\vec{x}\theta$ is the interpolant of $\phi$ and $\psi$*

**Proof** First, one can see that $\forall\vec{x}\theta$ does not contain any slokem term because it is not a common term of $\Sigma$ and $\Pi$. Therefore, the language of $\forall\vec{x}\theta$ is in the common language of $\phi$ and $\psi$. Second, assume $M$ is a model of $\phi$ but $M$ is not a model of $\forall\vec{x}\theta$, use the theorem 3.11 we can extend $M$ to $M'$ such as $M'$ is not a model of $\forall\vec{y}\Sigma$ ($\vec{y}$ are free variable in $\Sigma$). It is contradiction to the fact that $\phi \models \forall\vec{y}\Sigma$. The condition related to $\psi$ and $\Pi$ can be verified similarly. ■

Chapter 4

# Query rewriting framework

Following all formal definitions, theorems presented in previous chapters, we are now ready to define a query rewriting framework in first order logic which:

- Reduces the problem of deciding the existence of a rewriting of a query over a set of predicates and a knowledge base to validity of a first order formula.
- Provides an effective way to construct the rewriting if it exists.
- Points out conditions to transform query answering problem over database and knowledge base to model checking over database.
- Gives general properties of knowledge bases and original queries that make sure rewritten queries fulfill not only those conditions above but also constraints which guarantee these rewritings can be executed using relational database techniques.
- Proves that description logic $\mathcal{ACL}$ and guarded fragment logic are possible applications of the framework.

## 4.1 General framework

In the paper [4], a general framework for reformulating a query using a set of predicates named *database predicates* was described unambiguously. Therefore, in two first parts of this section, we shall follow that paper.

First, let us consider a set of notions and assumptions that will be used:

- $\mathcal{C}$ is the set of constants.
- $KB$ is a set of constraints, which actually is a set of first order sentences
- $M(KB)$ is the set of all models of $KB$
- Given a database $DB$, $\mathcal{P}_{DB}$ is a set of predicates occurring in $DB$
- Given a database $DB$, $\mathcal{C}_{DB}$ is a set of database constants in $DB$ and $\mathcal{C}_{DB} \subseteq \mathcal{C}$
- $\Sigma_{KB}$ is the signature of $KB$. $\Sigma_Q$ is the signature of formula $Q$
- $\sigma(Q)$ is the set of predicates in $Q$
- Given an interpretation $I$ and a set of predicates $\mathcal{P}$, $I|\mathcal{P}$ is an interpretation as $I$ in terms of domain $\Delta^I$ and interpretation function $\cdot^I$ but defined only for predicates in $\mathcal{P}$
- Given $KB$ and $\mathcal{P}_{DB}$, $\widetilde{KB}$ is a set of constraints as in $KB$ where every predicate $P \in \Sigma_{KB}$ such as $P \notin \mathcal{P}_{DB}$ is renamed to $\widetilde{P}$.

- Given a formula $Q$ and $\mathcal{P}_{DB}$, $\widetilde{Q}$ is defined similarly as $\widetilde{KB}$
- $I_{DB}$ means the interpretation $I$ embeds database $DB$ as described in section 2.4. $I^{\Delta}_{DB}$ where $\Delta$ is a domain is an interpretation embedding $DB$ with $\Delta$ is its domain.

### 4.1.1 Definability of a query

In order to rewrite a query using a set of predicates under a knowledge base, the definability of the query is taken into account. Revised from the notion of implicit definability of a predicate given by Beth [11], we have a similar definition for a query as follow:

**Definition 4.1 (Implicit definability: semantic)** *Let $I$ and $J$ be two models of the constrains $KB$. A query $Q_{[X]}$ is implicitly definable from the database predicates $\mathcal{P}_{DB}$ under constraints $KB$ iff $I|\mathcal{P}_{DB} = J|\mathcal{P}_{DB}$ and $\Delta^I = \Delta^J$ implies that for every substitution $\Theta^{\Delta^I}_X$ : $I, \Theta^{\Delta^I}_X \models Q_{[X]}$ iff $J, \Theta^{\Delta^I}_X \models Q_{[X]}$*

In fact, this definition means that given a set of constraints $KB$, the query is implicitly definable from $\mathcal{P}_{DB}$ if under arbitrary two $KB$'s models which have the same domain and the same interpretation function on predicates in $\mathcal{P}_{DB}$, the truth values of the query are indistinguishable. In the other words, the answer of an implicitly definable query depends only on the extension of database predicates. Therefore, there is another way to express implicit definiablity of a query as follows:

**Definition 4.2 (Implicit definability: syntactic)** *A query $Q_{[X]}$ is implicitly definable from the database predicate $\mathcal{P}_{DB}$ under the constraints $KB$ if :*

$$KB \cup \widetilde{KB} \models \forall X.Q_{[X]} \leftrightarrow \widetilde{Q_{[X]}}$$

One can verify that these definitions actually are coincident by the following theorem :

**Theorem 4.3** *Syntactic definition and semantic definition of implicit definability of a query are equivalent.*

**Proof**

- Semantic definition implies syntactic definition
  Let $M$ is a model of $KB \cup \widetilde{KB}$. We will prove that $M$ is also a model of $\forall X.Q_{[X]} \leftrightarrow \widetilde{Q_{[X]}}$. Based on the renaming rule, we can say $M = I \cup J'$ where $\Delta^I = \Delta^{J'} = \Delta^M$, $I|\mathcal{P}_{DB} = J'|\mathcal{P}_{DB}$, $I$ interprets un-renamed predicates while $J'$ interprets renamed predicates and $I$ is a model of $KB$, $J'$ is a model of $\widetilde{KB}$. Since $J'$ is a model of $\widetilde{KB}$ then $J$ is also a model of $KB$ (where $J$ is an interpretation such that: $\Delta^J = \Delta^{J'}$, $J|\mathcal{P}_{DB} = J'|\mathcal{P}_{DB}$ and for a predicate $P \in Sigma(KB)$ but $P \notin \mathcal{P}_{DB}$, $J$ interprets $P$ as $J'$ interprets $\widetilde{P}$ ). Apply the condition of semantic definition, we have for all $\Theta^{\Delta^I}_X$, if $I, \Theta^{\Delta^I}_X \models Q_{[X]}$ then $J, \Theta^{\Delta^I}_X \models Q_{[X]}$ (4). Actually, (4) is equivalent to if $I, \Theta^{\Delta^I}_X \models Q_{[X]}$ then $J', \Theta^{\Delta^I}_X \models \widetilde{Q}_{[X]}$. In the other words, $I \cup J'$ is a model of $\forall X.Q_{[X]} \rightarrow \widetilde{Q}_{[X]}$. Another direction can be proved analogously.
- Syntactic definition implies semantic definition
  Let $I, J$ be models of $KB$ fulfill the conditions that $\Delta^I = \Delta^J$, $I|\mathcal{P}_{DB} = J|\mathcal{P}_{DB}$. Let $\Theta^{\Delta^I}_X$ is the substitution such as $I, \Theta^{\Delta^I}_X \models Q_{[X]}$. We shall prove that $J, \Theta^{\Delta^I}_X \models Q_{[X]}$.
  From $J$ construct an interpretation $J'$ such that $\Delta^J = \Delta^{J'}$, $J|\mathcal{P}_{DB} = J'|\mathcal{P}_{DB}$ and for a predicate $P \in \Sigma(KB)$ but $P \notin \mathcal{P}_{DB}$, $J'$ interprets $\widetilde{P}$ as $J$ interprets $P$. Since $J$ is a model of $KB$ then $J'$ is a model of $\widetilde{KB}$. From the condition of syntactic implicit definability, we have $M = I \cup J'$ ($M$ is defined as above) is a model of $\forall X.Q_{[X]} \leftrightarrow \widetilde{Q}_{[X]}$ as well. Therefore, for all substitution $\Theta^{\Delta^I}_X$, if $I \cup J', \Theta^{\Delta^I}_X \models Q_{[X]}$ then $I \cup J', \Theta^{\Delta^I}_X \models \widetilde{Q}_{[X]}$ (1). Moreover, $Q$ does not contain renamed predicates and $\widetilde{Q}$ does not contain un-renamed predicates then

(2) is equivalent to for all substitution $\Theta_X^{\Delta^I}$, if $I, \Theta_X^{\Delta^I} \models Q_{[X]}$ then $J', \Theta_X^{\Delta^I} \models \widetilde{Q}_{[X]}$ (3). Based on the description above of $J'$, we have (3) is equivalent to if $I, \Theta_X^{\Delta^I} \models Q_{[X]}$ then $J, \Theta_X^{\Delta^I} \models Q_{[X]}$, which is our expected statement

■

Based on these definitions, one can see that the problem of checking whether a query is implicitly definable or not can be reduced to the problem of checking the validity of a first order formula. Obviously, it can be done automatically by using a prover.

In general, implicit definability means that it is possible to define a query using only a set of database predicates . In order to know what is the exact definition of this query based on database predicates we need a notion of explicit definability:

**Definition 4.4 (Explicit definability)** *A query $Q_{[X]}$ is explicitly definable from the database predicates $P_{DB}$ under constraints $KB$ iff there is some formula $\widehat{Q}_{[X]}$ in first order logic such as $KB \models \forall Q_{[X]} \leftrightarrow \widehat{Q}_{[X]}$ and $\sigma(\widehat{Q}) \subseteq P_{DB}$*

Related to the relationship between implicit definablility and explicit definability we have the following theorem :

**Theorem 4.5 (Projective Beth definability)** *If a query $Q$ is implicitly definable from the database predicates $P_{DB}$ under constraints $KB$, then it is explicitly definable as a formula $\widehat{Q}$ in first order logic with $\sigma(\widehat{Q}) \in P_{DB}$ under constraints $KB$*

This theorem can be proved by showing what $\widehat{Q}$ is in the next subsection.

### 4.1.2 Query rewriting

As we have already mentioned in the previous part, if the query is implicitly definable then there exists an formula as its explicit definition. In this subsection we shall prove that there is a constructive way to find the formula thanks to Craig's interpolant.

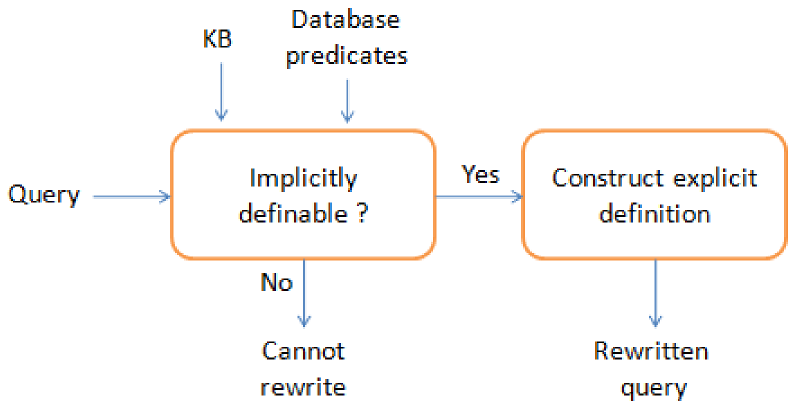


Figure 4.1: Query rewriting process

In general, the first steps of query rewriting framework are in the figure 4.1 where the '*Construct explicit definition*' module works based on this theorem :

**Theorem 4.6 (Interpolants as definitions)** *Let $Q$ be a query (with $n \geq 0$ free variables) implicitly defined from the database predicates $P_{DB}$ under the constraints $KB$. Then, the closed formula with $c_0, ..., c_{n-1}$ distinct constant symbols in $C$ not appearing in $KB$ or $Q$:*

$$((\bigwedge KB) \wedge Q_{[X/c_0,...,c_{n-1}]}) \rightarrow ((\bigwedge \widetilde{KB}) \rightarrow \widetilde{Q}_{[X/c_0,...,c_{n-1}]}) \tag{4.1}$$

*is valid, and its interpolant $\widehat{Q}_{[c_0,\ldots,c_{n-1}/X]}$ defines the query $Q$ explicitly from the database predicates $P_{DB}$ under the constraints $KB$.*

**Proof**
First we will prove that if $Q$ is implicitly definable then the formula 4.1 is valid. Applying syntactic definition of implicit definability: $KB \cup \widetilde{KB} \models \forall X.Q_{[X]} \leftrightarrow \widetilde{Q_{[X]}}$. Therefore, when we replace $X$ by a set of constants $c_0, \ldots, c_{n-1}$, the following formula is valid $(\bigwedge KB \wedge \bigwedge \widetilde{KB}) \rightarrow (Q_{[X/c_0,\ldots,c_{n-1}]} \leftrightarrow \widetilde{Q}_{[X/c_0,\ldots,c_{n-1}]})$. As a consequence, 4.1 is valid.
Next, we have to prove $KB \models (Q_{[X]} \leftrightarrow \widehat{Q}_{[c_0,\ldots,c_{n-1}/X]})$ where $\widehat{Q}_{[X/c_0,\ldots,c_{n-1}]}$ is a Craig interpolant of 4.1. Since $\widehat{Q}_{[X/c_0,\ldots,c_{n-1}]}$ is an interpolant:

1. $((\bigwedge KB) \wedge Q_{[X/c_0,\ldots,c_{n-1}]}) \rightarrow \widehat{Q}_{[X/c_0,\ldots,c_{n-1}]}$
   Then : $KB \models (Q_{[X/c_0,\ldots,c_{n-1}]} \rightarrow \widehat{Q}_{[X/c_0,\ldots,c_{n-1}]})$
2. $\widehat{Q}_{[X/c_0,\ldots,c_{n-1}]} \rightarrow ((\bigwedge \widetilde{KB}) \rightarrow \widetilde{Q}_{[X/c_0,\ldots,c_{n-1}]})$
   Then : $\widetilde{KB} \models (\widehat{Q}_{[X/c_0,\ldots,c_{n-1}]} \rightarrow \widetilde{Q}_{[X/c_0,\ldots,c_{n-1}]})$.
   Since $\sigma(\widehat{Q}) \subseteq \mathcal{P}_{DB}$, the relation $KB \models (\widehat{Q}_{[X/c_0,\ldots,c_{n-1}]} \rightarrow Q_{[X/c_0,\ldots,c_{n-1}]})$ holds as well

From(1)(2) we have expected statement.
Last but not least, since $\sigma(\widehat{Q}_{[X/c_0,\ldots,c_{n-1}]}) \subseteq \mathcal{P}_{DB}$ then $\sigma(\widehat{Q}_{[c_0,\ldots,c_{n-1}/X]}) \subseteq \mathcal{P}_{DB}$.
With above statements, $\widehat{Q}_{[c_0,\ldots,c_{n-1}/X]}$ is really an explicit definition of $Q$ ■

### 4.1.3 Query answering

Now we shall consider the answer of original query using the answer of its rewriting. Since the rewritten query contains only database predicates, we will mention as well a set of conditions to reduce rewriting answering problem into model checking one. More precisely, we have the following theorem:

**Theorem 4.7 (Domain independent rewriting)** *Let $DB$ be a database satisfying a set of constraints $KB$, and $Q_{[X]}$ be implicitly definable from $P_{DB}$ under $KB$ and $\widehat{Q}$ is an explicit definition $Q$. If the rewritten query $\widehat{Q}_{[X]}$ is domain independent and $\mathcal{C}$ satisfies standard name assumption, then:*

$$\{\Theta_X^{\mathcal{C}} |\ \textit{for every } I_{DB} \in M(KB) : I_{DB} \models Q_{[X/\Theta_X^{\mathcal{C}}]}\} =$$
$$\{\Theta_X^{\mathcal{C}_{DB} \cup \mathcal{C}_{\widehat{Q}}} | I_{(DB)}^{(\mathcal{C}_{DB} \cup \mathcal{C}_{\widehat{Q}})} | \mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}_{DB} \cup \mathcal{C}_{\widehat{Q}}}]}\}$$

*where $\mathcal{C}_{\widehat{Q}}$ is set of constants in $\widehat{Q}$*

Intuitively, this theorem states that if the rewriting is domain independent and standard name assumption holds for all constants in $\mathcal{C}$ then the problem of answering $Q$ coincides with model checking problem. In the other words, it shows a way to get rid of knowledge base. Notice that to make this theorem holds, we need standard name for all constants in $\mathcal{C}$ as in relational databases. By using this assumption, we loose the expressiveness of first order constants but reduce the complexity of query answering. From theoretical point of view, this assumption makes sense since in relational database, constants in queries have standard name, therefore, the set of constants in $\widehat{Q}$ which may contain both constants in $Q$ and $KB$ should satisfy this assumption. From practical point of view, in order to evaluate queries, the universal relation which contains all constants in $\mathcal{C}$ might be used as a table in the database, then these constants should have standard names as well.

A proof of this theorem can be implied from the following lemmas:

**Lemma 4.8** *Let $DB$ be a database satisfying a set of constraints $KB$, and $Q_{[X]}$ be implicitly definable from $P_{DB}$ under $KB$ and $\widehat{Q}$ is an explicit definition $Q$. If the rewritten query $\widehat{Q}_{[X]}$ is* *ground domain independent and $\mathcal{C}$ satisfies standard name assumption, then:*

$$\{\Theta_X^{\mathcal{C}} |\ \textit{for every}\ I_{DB} \in M(KB) : I_{DB} \models Q_{[X/\Theta_X^{\mathcal{C}}]}\} =$$
$$\{\Theta_X^{\mathcal{C}} |\ \textit{for every}\ \Delta^I \supseteq \mathcal{C} : I_{DB}^{\Delta^I}|\mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}}]}\}$$

**Proof**
Let $S_1 = \{\Theta_X^{\mathcal{C}} |$ for every $I_{DB} \in M(KB) : I_{DB} \models Q_{[X/\Theta_X^{\mathcal{C}}]}\}$
and $S_2 = \{\Theta_X^{\mathcal{C}} |$ for every $\Delta^I \supseteq \mathcal{C} : I_{DB}^{\Delta^I}|\mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}}]}\}$.
First, we show $S_2 \subseteq S_1$. Let $s = \Theta_X^{\mathcal{C}} \in S_2$. Assume that there is $I_{DB} \in M(KB)$ such that $I_{DB} \not\models Q_{[X/\Theta_X^{\mathcal{C}}]}$. Since $Q$ and $\widehat{Q}$ are equivalent under $KB$ then $I_{DB} \not\models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}}]}$. Because $\widehat{Q}$ contains only database predicates, $I_{DB}|\mathcal{P}_{DB} \not\models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}}]}$. Therefore there is an $\Delta^I$ is domain of $I_{DB}$ such that: $I_{DB}^{\Delta^I}|\mathcal{P}_{DB} \not\models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}}]}$, which is a contradiction because $s \in S_2$.
Now we prove $S_1 \subseteq S_2$ by contradiction. Assume that there is $s = \Theta_X^{\mathcal{C}} \in S_1$ but $s \notin S_2$. Therefore, there is a domain $\Delta^I$ such that: $I_{DB}^{\Delta^I}|\mathcal{P}_{DB} \not\models \widehat{Q}[X/s]$. Call $J_{DB}$ is a model of $KB$. We have: $\cdot^{I_{DB}^{\Delta^I}|\mathcal{P}_{DB}} = \cdot^{J_{DB}|\mathcal{P}_{DB}}$ because $DB$ contains only $\mathcal{P}_{DB}$. Then, since $\widehat{Q}$ is ground domain independent, $I_{DB}^{\Delta^J}|\mathcal{P}_{DB} \not\models \widehat{Q}[X/s]$ (1). Furthermore, because we have standard name assumption for $\mathcal{C}$ then for every constant $c \in \mathcal{C}$: $c^I = c^J$. Therefore we can extend the interpretation function of $I_{DB}^{\Delta^I}|\mathcal{P}_{DB}$ to interpretation function of $J$ (2). From (1) and (2), we have $J \not\models \widehat{Q}_{[X/s]}$. But $J$ is a model of $KB$ and $Q_{[X/s]}$ and $\widehat{Q}_{[X/s]}$ are equivalent under $KB$ then $J \not\models Q_{[X/s]}$. In the other words, $s \notin S_1$ - which is a contradiction. ■

**Lemma 4.9** *If the rewritten query $\widehat{Q}_{[X]}$ is domain independent and $\mathcal{C}$ satisfies standard name assumption then:*

$$\{\Theta_X^{C} |\ \textit{for every}\ \Delta^I \supseteq \mathcal{C} : I_{DB}^{\Delta^I}|\mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{C}]}\} =$$
$$\{\Theta_X^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}} | I_{DB}^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}}|\mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}}]}\}$$

**Proof**
Let $S_3 = \{\Theta_X^{C} |$ for every $\Delta^I \supseteq \mathcal{C} : I_{DB}^{\Delta^I}|\mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{C}]}\}$
and $S_4 = \{\Theta_X^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}} | I_{DB}^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}}|\mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}}]}\}$.
Let $s$ be a substitution such that $s \in S_4$. We have: $I_{DB}^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}}|P_{DB} \models \widehat{Q}_{[X/s]}$. Since $\widehat{Q}$ is domain independent then for any domain $\Delta^I$ such that: $\Delta^I \supseteq \mathcal{C} \supseteq \mathcal{C}_{DB} \cup \mathcal{C}_{\widehat{Q}}$, $I_{DB}^{\Delta^I}|\mathcal{P}_{DB} \models \widehat{Q}_{[X,s]}$. Therefore $s \in S_3(3)$.
Let $s'$ be a substitution such that $s' \in S_3$. Consider an arbitrary domain $\Delta \supseteq \mathcal{C}$. Let $I$ is a first order interpretation such that $\Delta^I = \Delta$ and $\cdot^I = \cdot^{I_{DB}^{\Delta}|\mathcal{P}_{DB}}$.
We have: $act - range(s') \subseteq \Delta^I$ and $I, s' \models \widehat{Q}$ because $s' \in S_3$
Let $J$ is a first order interpretation such that $\Delta^J = \mathcal{C}_{DB} \cup \mathcal{C}_{\widehat{Q}}$ and $\cdot^J = \cdot^{I_{DB}^{\Delta}|\mathcal{P}_{DB}} = \Delta^I$.
Since the query $\widehat{Q}$ is domain independent and $\cdot^J = \cdot^I$, we have: $act - range(s') \in J$ and $J, s' \models \widehat{Q}$. In the other words, $s' \in S_4(4)$.
From (3) and (4), $S_3 = S_4$. ■

Domain independence of rewritings not only reduces the complexity of query answering to complexity of model checking as in theorem 4.7 but also is needed criteria to transform the rewritings to queries in relational languages. Unfortunately, it is still a open question about general constraints of knowledge bases and original queries which guarantee the domain independence of rewritten queries. However, as we will show in the next section, together with some conditions of knowledge bases and queries, there is a method to have domain independent rewritings.

## 4.2 Safe range rewriting

In this section, we shall consider safe range knowledge base and safe range original queries as input of our framework and show that this input will guarantee domain independence of output if we use tableau method. From practical point of view, the assumption about safe range properties of $KB$ and $Q$ is feasible because the most important typical constraints (as in the section 2.4) considered in the database literature are safe range. Furthermore, it is also an efficient way to guarantee the domain independence of input.

First, as a negative result, we do not have domain independent/safe range output in general if the input is safe range. A counterexample is as followed:

**Example 4.1** *Let $KB$ be:*

$$\forall x, y(V_1(x,y) \leftrightarrow \exists z, v(R(z,x) \land R(z,v) \land R(v,y)))$$
$$\forall x, y(V_2(x,y) \leftrightarrow \exists z(R(x,z) \land R(z,y)))$$
$$\forall x, y(V_3(x,y) \leftrightarrow \exists z, v(R(x,z) \land R(z,v) \land R(v,y)))$$

*and $Q(x,y)$ be $\exists z, v, u(R(z,x) \land R(z,v) \land R(v,u) \land R(u,y))$ as in [22].*

*Applying our rewriting framework, we have $\widehat{Q}(x,y) = \exists z(V_1(x,z) \land \forall v(V_2(v,z) \rightarrow V_3(v,y)))$ is an rewriting of $Q$. It is easy to see that all sentences in $KB$ and $Q$ are safe range while $\widehat{Q}$ is not safe range/domain independent. However, if we replace all free variables of $\widehat{Q}$ by constants then the result will be safe range.*

Motivated by this intuition, within the first subsection, we will introduce the notion of '*ground safe range*' and its role to have domain independent rewriting. In the second one, we shall prove if input is safe range then output computed by tableau based method is ground safe range. Last but not least, a less expensive way to transform the ground safe range rewriting to safe range one will be taken into account.

### 4.2.1 Ground safe range

**Definition 4.10 (Ground safe range)** *A query $Q_{[X]}$ is ground safe range iff $Q_{X/\Theta_X^C}$ is safe range for every substitution $\Theta_X^C$*

In the next theorem, we will see the role of ground safe range in query answering problem. More precisely, ground safe range queries can be transformed to the safe range ones (which are also domain independent) without changing the answers.

**Theorem 4.11 (Ground safe range rewriting)** *Let us extend the database $DB$ with a new unary relation $T_{\mathcal{C}}$ (named universal predicate) containing all the constants in $\mathcal{C}$. If the rewritten query $\widehat{Q}_{[X]}$ is a ground safe range query, then:*

$$\{\Theta_X^{\mathcal{C}}|\ \textit{for every}\ I_{DB} \in M(KB) : I_{DB} \models Q[X/\Theta_X^{\mathcal{C}}]\} =$$
$$\{\Theta_X^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}} | I_{DB}^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}} | P_{DB} \models (\widehat{Q}_{[X]} \land T_{\mathcal{C}}(x_1)) \land ... \land T_{\mathcal{C}}(x_n))_{[X/\Theta_X^{\mathcal{C}_{DB}\cup\mathcal{C}_{\widehat{Q}}}]}\}$$

**Proof** Sketch. Since ground safe range implies ground domain independent then from lemma 4.8 we have :

$$\{\Theta_X^{\mathcal{C}}|\ \text{for every}\ I_{DB} \in M(KB) : I_{DB} \models Q_{[X/\Theta_X^{\mathcal{C}}]}\} =$$
$$\{\Theta_X^{\mathcal{C}}|\ \text{for every}\ \Delta^I \supseteq \mathcal{C}_{DB} : I_{(DB)}^{(\Delta^I)} | \mathcal{P}_{DB} \models \widehat{Q}_{[X/\Theta_X^{\mathcal{C}}]}\}$$

Since we just consider all substitutions over $\mathcal{C}$ then then adding $T_{\mathcal{C}}$ restriction for $\widehat{Q}$'s free variables as above does not affect the meaning of the query. Moreover, it's easy to see that $\widehat{Q}_{[X]} \land T_C(x_1)) \land ... \land T_C(x_n))_{[X/\Theta_X^{\mathcal{C}_{DB}}]}$ is a safe range formula (because $\widehat{Q}_{[X]}$ is ground safe range), then it is domain independence. Therefore, we can apply the lemma 4.9 for this query in order to receive the desirable answer. ■

### 4.2.2 Tableau based rewriting

In this subsection, we show that if the interpolant in theorem 4.6 is computed by tableau method considered in 3.2.1 then the rewritings of safe range queries over safe range knowledge bases are ground safe range. In order to prove it, we need following propositions:

**Proposition 4.12** *$\varphi_1 \wedge \varphi_2$ is safe range and closed iff $\varphi_1$ and $\varphi_2$ are safe range and closed.*

**Proof**
We have :

- $rr(\varphi_1 \wedge \varphi_2) = rr(\varphi_1) \cup rr(\varphi_2)$
- $free(\varphi_1 \wedge \varphi_2) = free(\varphi_1) \cup free(\varphi_2)$
- $rr(\varphi_1) = \perp$ or $rr(\varphi_1) \subseteq free(\varphi_1)$
- $rr(\varphi_2) = \perp$ or $rr(\varphi_2) \subseteq free(\varphi_2)$
- $\varphi_1 \wedge \varphi_2$ is closed iff $free(\varphi_1) = free(\varphi_2) = \emptyset$
- $\varphi_1$ is closed iff $free(\varphi_1) = \emptyset$
- $\varphi_2$ is closed iff $free(\varphi_2) = \emptyset$
- $\varphi_1 \wedge \varphi_2$ is safe range iff $rr(\varphi_1 \wedge \varphi_2) = free(\varphi_1 \wedge \varphi_2)$
- $\varphi_1$ is safe range iff $rr(\varphi_1) = free(\varphi_1)$
- $\varphi_1$ is safe range iff $rr(\varphi_2) = free(\varphi_2)$

Therefore :

- $\varphi_1 \wedge \varphi_2$ is closed iff $\varphi_1$ and $\varphi_2$ are closed
- $\varphi_1 \wedge \varphi_2$ is closed, safe range iff $\varphi_1$ and $\varphi_2$ are closed, safe range

■

**Proposition 4.13** *$\varphi_1 \vee \varphi_2$ is safe range and closed iff $\varphi_1$ and $\varphi_2$ are safe range and closed.*

**Proof**
We have :

- $rr(\varphi_1 \vee \varphi_2) = rr(\varphi_1) \cap rr(\varphi_2)$
- $free(\varphi_1 \vee \varphi_2) = free(\varphi_1) \cup free(\varphi_2)$
- $rr(\varphi_1) = \perp$ or $rr(\varphi_1) \subseteq free(\varphi_1)$
- $rr(\varphi_2) = \perp$ or $rr(\varphi_2) \subseteq free(\varphi_2)$
- $\varphi_1 \vee \varphi_2$ is closed iff $free(\varphi_1) = free(\varphi_2) = \emptyset$
- $\varphi_1$ is closed iff $free(\varphi_1) = \emptyset$
- $\varphi_2$ is closed iff $free(\varphi_2) = \emptyset$
- $\varphi_1 \vee \varphi_2$ is safe range iff $rr(\varphi_1 \vee \varphi_2) = free(\varphi_1 \vee \varphi_2)$
- $\varphi_1$ is safe range iff $rr(\varphi_1) = free(\varphi_1)$
- $\varphi_1$ is safe range iff $rr(\varphi_2) = free(\varphi_2)$

Therefore :

- $\varphi_1 \vee \varphi_2$ is closed iff $\varphi_1$ and $\varphi_2$ are closed
- $\varphi_1 \vee \varphi_2$ is closed, safe range iff $\varphi_1$ and $\varphi_2$ are closed, safe range

■

**Proposition 4.14** *$\forall\vec{x}\varphi(\vec{x})$ is closed and safe range then $\varphi(\vec{t})$ is closed and safe range where $\vec{t}$ are constants.*

**Proof**
Obviously, if $\forall\vec{x}\varphi(\vec{x})$ is closed then $\varphi(\vec{t})$ is closed.
Assume that $\varphi(\vec{t})$ is not safe range. Since it's closed then $rr(SRNF(\varphi(\vec{t}))) = \bot$
$\Rightarrow SRNF(\varphi(\vec{t}))$ must contain a subformula which is in the form $\exists\vec{z}\varphi'(\vec{t},\vec{z})$
where $\vec{z} \not\subseteq rr(SRNF(\varphi'(\vec{t},\vec{z})))$
$\Rightarrow SRNF(\varphi(\vec{x}))$ must contain a subformula which is in the form $\exists\vec{z}\varphi'(\vec{x},\vec{z})$
where $\vec{z} \not\subseteq rr(SRNF(\varphi'(\vec{x},\vec{z})))$
$\Rightarrow SRNF(\neg\varphi(\vec{x}))$ must contain a subformula which is in the form $\exists\vec{z}\varphi'(\vec{x},\vec{z})$
where $\vec{z} \not\subseteq rr(SRNF(\varphi'(\vec{x},\vec{z})))$ because pushing negation does not effect the formula under $\exists$
$\Rightarrow rr(SRNF(\neg\varphi(\vec{x}))) = \bot$
$\Rightarrow rr(SRNF(\neg\exists\vec{x}\neg\varphi(\vec{x}))) = \bot$
$\Rightarrow rr(SRNF(\forall\vec{x}\varphi(\vec{x}))) = \bot$
$\forall\vec{x}\varphi(\vec{x})$ is not safe range
$\Rightarrow$ *Contradiction.* ■

**Proposition 4.15** *$\exists\vec{x}\varphi(\vec{x})$ is closed and safe range then $\varphi(\vec{t})$ is closed and safe range where $\vec{t}$ are constants.*

**Proof**
Of course, if $\exists\vec{x}\varphi(\vec{x})$ is closed then $\varphi(\vec{t})$ is closed.
Assume that $\varphi(\vec{t})$ is not safe range. Since it's closed then $rr(SRNF(\varphi(\vec{t}))) = \bot$
$\Rightarrow SRNF(\varphi(\vec{t}))$ must contain a subformula which is in the form $\exists\vec{z}\varphi'(\vec{t},\vec{z})$
where $\vec{z} \not\subseteq rr(SRNF(\varphi'(\vec{t},\vec{z})))$
$\Rightarrow SRNF(\varphi(\vec{x}))$ must contain a subformula which is in the form $\exists\vec{z}\varphi'(\vec{x},\vec{z})$
where $\vec{z} \not\subseteq rr(SRNF(\varphi'(\vec{x},\vec{z})))$
$\Rightarrow rr(SRNF(\varphi(\vec{x}))) = \bot$
$\Rightarrow rr(SRNF(\exists\vec{x}\varphi(\vec{x}))) = \bot$
$\Rightarrow \exists\vec{x}\varphi(\vec{x})$ is not safe range
$\Rightarrow$ *Contradiction.* ■

Based on these propositions, we have the following theorem :

**Theorem 4.16 (Ground safe range)** *Given a set of constraints $KB$, a query $Q$ which is implicitly definable from $\mathcal{P}_{DB}$, if $KB$ and $Q$ are safe range then there is a rewritten $\widehat{Q}$ of $Q$ which is ground safe range.*

**Proof**
First we will show that if $\phi$ and $\psi$ are closed and safe range and $\phi \to \psi$ is valid then so is their interpolant. Assume $T$ is a biased tableau of of $\phi \wedge \neg\psi$. Therefore the root node if $T$ is $S_) = \{L(\phi), R(\neg\psi)\}$. Based on all the tableau expansion rules and above propositions we have at every expansion step where $S = \{L(\varphi_1), ..., L(\varphi_n), R(\psi_1), ..., R(\psi_m)\}$, $\varphi_1, ..., \varphi_n$ and $\psi_1, ..., \psi_m$ are safe range and closed(*) .
Now we have to prove that the interpolant at each step is safe range and closed (**) by induction on the shape of proof and the set of rules in 3.2.1.

- Rules for closed branches: It's trivial because $\varphi$ and $\neg\varphi$ are safe range and closed because of (*)

- Rules for propositional case :
  For the rule (p1)(p2)(p3)(p4) nothing changes, so we don't have to prove
  For the rule (p5), apply the proposition 4.19 then (**) holds
  For the rule (p6), apply the proposition 4.20 then (**) holds

- Rules for first order case :
  For the rule (f1) (f2) (f3) nothing changes, so we don't have to prove
  For the rule (f4), since $c$ does not occur in $\{\varphi_1, ..., \varphi_n\}$ then the only case to have $c$ in $I$ is that $S$ contains $R(\neg\varphi(c))$. Therefore $S \cup \{L(\varphi(c))\} \overset{int}{\to} I = \varphi(c)$. Since $\forall x.\varphi(x)$ is safe range (due to (*)) then $\forall x.I[c/x]$ is safe range too
  For the rule (f5), since $c$ does not occur in $\{\psi_1, ..., \psi_m\}$ then the only case to have $c$ in $I$ is that $S$ contains $L(\neg\varphi(c))$. Therefore $S \cup \{R(\varphi(c))\} \overset{int}{\to} I = \neg\varphi(c)$. Since $\forall x.\varphi(x)$ is safe range (due to (*)) then $\exists x.\neg I[c/x]$ is safe range too

- Rules for equality : Since all the input formulas are closed and do not contain function symbols then all equations are ground. Therefore, they do not influence the safe range property of interpolant in each step.

As a consequence, since $\mathcal{Q}(\vec{c})$, $\mathcal{KB}$, $\mathcal{KB}'$,$\neg\mathcal{Q}(\vec{c})$ are closed and safe range then so is the interpolant $\widehat{Q}(\vec{c})$ of $\mathcal{KB} \wedge \mathcal{Q}(\vec{c})$ and $\mathcal{KB}' \to \mathcal{Q}'(\vec{c})$. ∎

### 4.2.3 From ground safe range to safe range

As we see in above subsection, ground safe range property of rewritings is enough to transform them to a domain independent one. In the other words, if rewritten queries are ground safe range then we can reduce the original query answering problem to model checking. Now, let us consider conditions of rewritings to use standard relational algebra technique to solve the model checking problem. Based on the Codd's theorem (theorem 2.10) we comprehend that in order to transform a first order query to a query in relational algebra (and then to SQL query), the first order query should be safe range. From the theorem 4.11, it is obvious that if a query is ground safe range, then we can convert it to a safe range one by conjunction with universal predicate for each free variable. However, the safe range query is expensive in term of query answering because we have to check substitutions of each variable for each constant in the database.

Motivated by above reason, we shall do one more step in order to transform the ground safe range rewritten query to a less expensive safe range one by the following assumption on the set of constraints $KB$ and $P_{DB}$:

**Assumption 4.17** *Let extend the signature of $KB$ and $P_{DB}$ by a set of finite unary predicate $UP$. For each predicate $P \in \Sigma(KB)$ add the following safe range axiom to $KB$:*

$$\forall x_1, ..., x_n(P(x_1, ..., x_n) \to P_1(x_1) \wedge ... \wedge P_n(x_n))$$

*where $P_i \in UP$*

It is obvious that these axioms are feasible in practical databases because they mention each constant should belong to one category. For example, given a predicate $Student(x, y)$ where $x$ is his name and $y$ is his age then we might have $Student(x, y) \to Name(x) \wedge Age(y)$ or $Student(x, y) \to String(x) \wedge Int(y)$. Notice that in the worst case, all predicates in $UP$ are universal predicate.

Basing on above assumption, we have the following theorem:

**Theorem 4.18 (Safe range rewriting)** *Given a safe range knowledge base $KB$, a safe range query $Q_{[X]}$ and $\widehat{Q}_{[X]}$ is its ground safe range rewriting over $KB$, there is a safe range query which has the same answer as to $\widehat{Q}_{[X]}$ and does not contain universal predicate.*

**Proof**
We will prove this theorem by a constructive proof. Firstly, since $Q$ is a safe range query then for all $x_i \in X$, there will be a positive predicate $P$ taking $x_i$ as its $j-th$ argument. Secondly, let $\phi := \widehat{Q}_{[X]}$, for each $x_i \in X$ which is not range restricted in $\widehat{Q}$ ($x_i \in X \backslash rr(\widehat{Q})$), $\phi := \phi \wedge P_j(x_i)$. Finally, after this process we have $\phi$ is safe range and $\phi$ and $\widehat{Q}$ have the same answer.
Moreover, call $SF(\widehat{Q})$ is the standard transformation of $\widehat{Q}$ to safe range as in the theorem 4.11. It is easy to see that in general answering $SF(\widehat{Q})$ is harder than answering $\phi$ because for every $P \in UP$, $P \subseteq T_{\mathcal{C}}$ where $T_{\mathcal{C}}$ is the universal predicate . ■

## 4.3 Applied framework for $\mathcal{ALC}$

In this section, we shall consider an application of previous framework for $\mathcal{ALC}$ where database constraints here are $\mathcal{ALC}$'s axioms, queries are $\mathcal{ACL}$ concepts and databases are expressed via *DBoxes* as in the paper [26].

### 4.3.1 DBox

Let us formalize the notion of DBox in this subsection. Let $N_C$ and $N_R$ be countably infinite sets of concept and role names respectively and $N_I$ be a countably infinite set of individual names. The syntax of DBox is similar to syntax of normal $\mathcal{ALC}$ ABox such that:

- $\sigma_D$ is a signature where $\sigma_D(C) \subseteq N_C$ is the set of DBox concept names, $\sigma_D(R) \subseteq N_R$ is the set of DBox roles, $\sigma_D(I) \subseteq N_I$ is the set of DBox individuals.
- $A(a)$ and $R(a,b)$ are Dbox assertions where $A \in \sigma_D(C)$, $R \in \sigma_D(R)$ and $a, b \in \sigma_D(I)$
- $\sigma_D(P) = \sigma_D(C) \cup \sigma_D(R)$ is the set of DBox predicates.

In term of semantic, an interpretation $I = (\Delta^I, \cdot^I)$ is a model of a DBox $\mathcal{D}$ (written $I \models \mathcal{D}$) if :

- $a^I = a$ for every DBox individual $a \in \sigma_D(I)$
- For every $A \in \sigma_D(C)$ and $u \in \sigma_D(I)$, $u \in A^I$ iff $A(u) \in \mathcal{D}$
- For every $R \in \sigma_D(R)$ and $u, v \in \sigma_D(I)$, $(u,v) \in R^I$ iff $R(u,v) \in \mathcal{D}$

From above semantic, one can see it is different from the semantic of ABox since the set of '**true**' assertions is bounded by DBox and nothing else. We call the set of individuals in $\sigma_D(I)$ appearing in $\mathcal{D}$ is *active domain* of $\mathcal{D}$.

In the combination with a TBox $\mathcal{T}$: $\mathcal{D}$ is satisfiable with respect to $\mathcal{T}$ iff there is a model of $\mathcal{D}$ is also a model of $\mathcal{T}$. Given an arbitrary concept $C$ and an individual $a$, $(\mathcal{T}, \mathcal{D}) \models C(a)$ iff for every model $I$ of $\mathcal{T}$ and $\mathcal{D}$, $I$ is also a model of $C(a)$. Similarly, $\mathcal{D} \models C(a)$ iff $(\emptyset, \mathcal{D}) \models C(a)$ where $\emptyset$ is the empty TBox.

### 4.3.2 The framework

The query rewriting problem in this case can be stated as: given a TBox $\mathcal{T}$, a DBox $\mathcal{D}$ which satisfies $\mathcal{T}$ and a query which is actually an $\mathcal{ALC}$ concept $Q$, check the implicit definability of $Q$ over a set of DBox predicates $P$ of $\mathcal{D}$. If it is the case then find an explicit definition.

Before going to detail definitions and theorems, let us call $\sigma(Q, \mathcal{T}) = \{B_1, ..., B_m, D_1, ..., D_m\}$ as a set of role names and concept names occurring in $Q$ or $\mathcal{T}$ where $\{D_1, ..., D_m\} \subseteq \sigma_D(P)$ and $\{B_1, ..., B_m\} \cap \sigma_D(P) = \emptyset$.

With these notions we have :

**Definition 4.19 (Implicit definability in $\mathcal{ALC}$)** *Let a concept $\phi'$ be like $\phi$ but with all occurrences of $B_1, ..., B_m$ replaced by distinct occurrences of $B'_1, ..., B'_m \notin \sigma(Q, \mathcal{T}) \cup \sigma_D(P)$. $\mathcal{T}'$ and $Q'$ are defined similarly. Then $Q$ is implicitly definable from $D_1, ..., D_m$ in $T$ iff $Q \equiv_{\mathcal{T} \cup \mathcal{T}'} Q'$.*

**Definition 4.20 (Explicit definability in $\mathcal{ALC}$)** *$Q$ is explicitly definable from $D_1, ..., D_n$ in $\mathcal{T}$ iff there is some concept $C$ such that $Q \equiv_{\mathcal{T}} C$ and $\sigma(C) \subseteq \{D_1, ..., D_n\}$*

For the relationship between implicit definability and explicit definability, similar to the general framework, the following theorem holds:

**Theorem 4.21 (Beth definability for $\mathcal{ALC}$ with TBox)** *If $Q$ is implicitly definable from $D_1, ..., D_n$ in $\mathcal{T}$ then $Q$ is explicitly definable from $D_1, ..., D_n$*

**Proof** From the definition of implicit definability we have $Q \equiv_{\mathcal{T} \cup \mathcal{T}'} Q'$ (*). From previous chapter we also know that the interpolant theorem holds in the case of $\mathcal{ALC}$ with TBox. Therefore, from (*) there is an interpolant $C$ which is an $\mathcal{ALC}$ concept between $Q$ and $Q'$ such as :

1. $Q \sqsubseteq_{\mathcal{T} \cup \mathcal{T}'} C$
2. $C \sqsubseteq_{\mathcal{T} \cup \mathcal{T}'} Q'$
3. $\sigma(C) \subseteq \sigma(Q, \mathcal{T}) \cap \sigma(Q', \mathcal{T}')$. I

Since we also have $Q' \sqsubseteq Q$ then combine with (2), we have $Q \equiv_{\mathcal{T} \cup \mathcal{T}'} C$ (**). In the other side, $\sigma(Q, \mathcal{T}) \cap \sigma(Q', \mathcal{T}') \subseteq \{D_1, ..., D_m\}$ therefore $\sigma(C) \subseteq \{D_1, ..., D_m\}$ (***). From both (**) and (***), we have $C$ is exact explicit definition of $Q$ in $\mathcal{T}$.
Notice that $C$ is constructive because there is a constructive proof of interpolant theorem for $\mathcal{ALC}$ with TBox as we already mentioned in 3.2.2. ■

### 4.3.3 Ground safe range theorem

In order to apply the theorem 4.11 to reduce query answering problem to model checking in DBox, we need ground safe range of the rewritten query where a concept is ground safe range if its equivalent first order formula is ground safe range. Fortunately, we have the following theorem :

**Theorem 4.22 (Ground safe range theorem for $\mathcal{ALC}$ concept)** *Every concept in $\mathcal{ALC}$ is ground safe range.*

**Proof**
We will prove this theorem by induction on the structure of concept $C$. Without loss of generality, we can assume that $C$ is in negation normal form. For a concept $C$, we call $f(C)$ is the first order formula corresponding to $C$

- Base cases : For atomic concept $A$ where $f(A)(x) = A(x)$, we have $A(a)$ is safe range for every constant $a$. Therefore $A$ is ground safe range. For concept $\neg A$, $f(\neg A)(x) = \neg A(x)$ and $\neg A(a)$ is also safe range. Then, $\neg A$ is ground safe range as well.
- Assume $C = C_1 \wedge C_2$ and $C_1, C_2$ are both ground safe range. We have: $f(C)(x) = f(C_1)(x) \wedge f(C_2)(x)$. Since $f(C_1)(a)$, $f(C_2)(a)$ are safe range then so is $f(C)(a)$ . Therefore $f(C)(x)$ is ground safe range.
- Assume $C = C_1 \vee C_2$ and $C_1, C_2$ are both ground safe range. We have: $f(C)(x) = f(C_1)(x) \vee f(C_2)(x)$. Since $f(C_1)(a)$, $f(C_2)(a)$ are safe range then so is $f(C)(a)$ . Therefore $f(C)(x)$ is ground safe range.
- Assume $C = \forall R.C_1$ then $f(C)(x) = \forall y(R(x, y) \rightarrow f(C_1)(y))$. We have $SRNF(f(C)(x)) = \neg\exists y(R(x, y) \wedge \neg f(C_1)(y))$, when we replace $x$ by a constant $a$, we have $rr(SRNF(f(C)(a))) = \emptyset = free(SRNF(f(C)(x)))$. Therefore $f(C)(x)$ is ground safe range.
- Assume $C = \exists R.C_1$ then $f(C)(x) = \exists y(R(x, y) \wedge f(C_1)(y))$. We have $SRNF(f(C)(a)) = f(C)(a)$, when we replace $x$ by a constant $a$, we have $rr(SRNF(f(C)(a))) = \emptyset = free(SRNF(f(C)(a)))$. Therefore $f(C)(x)$ is ground safe range.

■

Assume $\widehat{Q}$ is the concept that explicitly defines $Q$ in $\mathcal{T}$ over $\mathcal{D}$. Apply this theorem and similar arguments as in the proof of theorem 4.11 one can see that: $\{a|(\mathcal{T},\mathcal{D}) \models Q(a)\} = \{a|\mathcal{D} \models \widehat{Q}(a)\}$ and $A(x) = f(\widehat{Q})(x) \wedge T_{\mathcal{D}}(x)$ will be the safe range query which gives the expected answer where $T_{\mathcal{D}}$ is the relation containing all constants in $\sigma_D(I)$.

## 4.4 Applied framework for guarded fragment

In this subsection, we shall consider a decidable fragment of first order logic named '*guarded fragment*' ($\mathcal{GF}$), which has been applied as a first order view for description logic. In term of syntax, the set of $\mathcal{GF}$-formulas is the smallest set such that :

- $R(t_1, ..., t_n)$ is an atomic formula where $R$ is a $n-ary$ predicate and $t_i$ is a constant or variable.
- If $\phi$ is a $\mathcal{GF}$ formula then so is $\neg\phi$
- If $\phi, \psi$ are $\mathcal{GF}$ formulas then $\phi \wedge \psi$ and $\phi \vee \psi$ are in $\mathcal{GF}$ as well
- If $G$ is an atomic $\mathcal{GF}$-formula, $\phi \in \mathcal{GF}$, and $X$ is a finite, non-empty set of variables such as $X \subseteq free(\phi) \subseteq free(G)$ then $\exists X.G \wedge \phi$ and $\forall X.G \rightarrow \phi$ are $\mathcal{GF}$-formulas. $G$ is called as the *guard* of the quantifier.

In term of semantic, given an interpretation $I = (\Delta^I, \cdot^I)$ and an variable assignment $\mathcal{A}$ which assigns each variable to an element in $\Delta^I$. Given two assignments $\mathcal{A}$ and $\mathcal{B}$, we say $\mathcal{B}$ in the interpretation $I$ is an $X-variant$ of $A$ iff $x^{\mathcal{A}} = x^{\mathcal{B}}$ for every $x \notin X$.

- $c^{I,\mathcal{A}} = c^I$ where $c$ is a constant
- $x^{I,\mathcal{A}} = x^A$ where $x$ is a variable
- $I, \mathcal{A} \models R(t_1, ..., t_n)$ iff $(t_1^{I,\mathcal{A}}, ..., t_n^{I,\mathcal{A}}) \in R^I$
- $I, \mathcal{A} \models \neg\phi$ iff $I, \mathcal{A} \not\models \phi$
- $I, \mathcal{A} \models \phi \wedge \psi$ iff $I, \mathcal{A} \models \phi$ and $I, \mathcal{A} \models \psi$
- $I, \mathcal{A} \models \phi \vee \psi$ iff $I, \mathcal{A} \models \phi$ or $I, \mathcal{A} \models \psi$
- $I, \mathcal{A} \models \forall X.G \rightarrow \phi$ iff $I, \mathcal{B} \models G$ implies $I, \mathcal{B} \models \phi$ for every assignment $\mathcal{B}$ in $I$ that is an $X-variant$ of $\mathcal{A}$
- $I, \mathcal{A} \models \exists X.G \wedge \phi$ iff $I, \mathcal{B} \models G$ and $I, \mathcal{B} \models \phi$ for some assignment $\mathcal{B}$ in $I$ that is an $X-variant$ of $\mathcal{A}$

Related to Craig's interpolant theorem and Beth's definability theorem, in $\mathcal{GF}$ the former does not hold [15] while the latter is still satisfied. Therefore, we can apply our query rewriting framework where all $KB$ constraints and implicitly definable queries are in $\mathcal{GF}$ in order to receive a rewritten query in $\mathcal{GF}$ as well. Furthermore, all queries in $\mathcal{GF}$ have very nice property that we called '*ground safe range*' as in this theorem:

**Theorem 4.23 (Ground safe range of $\mathcal{GF}$)** *Queries in the guarded fragment GF of first order logic are ground safe range.*

**Proof** Sketch. One can verify that the ground instance of a $\mathcal{GF}$ formula is always safe range based on induction in the shape of the formula. ■

Therefore, we can also apply theorem 4.11 for rewritings in $\mathcal{GF}$.

Chapter 5

# Rewriting tool

In this chapter, we shall describe our query rewriting tool which is built on top of a well-know automated theorem prover named Prover9. Since it is a piece of software, we follow the typical software development process to illustrate our tool.

## 5.1 Requirements

As we have mentioned in above chapters, our crucial goal is developing a software system which allows users query using a richer language (knowledge base language) than database language. As a very first version of this software, which is also support researchers explore more properties of the theoretical framework, the rewriting tool mentioned in this chapter should fulfill following requirements:

### 5.1.1 System features

- *Checking implicit definability*: Given a knowledge base, a query and a set of database predicates, the tool should return an answer about the implicit definability of the query.
- *Finding the rewritten query*: This feature contains the previous one: if query is not implicitly definable, system will return a corresponding answer, if query is implicitly definable, it will return the rewritten query. Notice that in the case query is implicitly definable but system cannot find a rewritten query in a fixed period of time, the rewriting process should stop and returns a response that it's timeout.
- *Checking the consistency of the knowledge base*: Given a knowledge base, the tool should tell users whether this knowledge base is consistent or not.
- *Exploring safe range property*: Given a knowledge base or a query, the tool should respond users about the safe range property of the input. This feature also should work for the rewritten query (if it is available).

### 5.1.2 External interface requirements

- *User interface*: In term of user interaction, the tool should allow users enter logic formulas in a convenient way as in normal syntax of first order logic. Return formulas have to follow this syntax as well.
- *Software interfaces*: In order to support as many as possible number of users, the tool should be implemented in an independent platform.
- *Communication interfaces*: From the communication with other logic systems perspective, the tool should also support some usual logical syntaxes except the one mention in the first

requirement such as: the syntax proposed by TPTP [27] and the syntax in Common Logic standard [18]. Specification of these syntaxes will be summarized in the next subsections.

### 5.1.3 TPTP syntax

TPTP stands for 'Thousands of Problems for Theorem Provers', which is the official library for automated theorem proving (ATP) systems. This library is also used as the set of problems that all provers attending the The World Championship for Automated Theorem Proving [6] have to solve. According to the website of TPTP [27], the TPTP supports ATP community in terms of :

- A complete set of ATP test problems with the comprehensive related information for each problem.
- Size varying of generic problems.
- A tool to transform a problem into some syntaxes of common ATP systems.
- Guidelines for evaluation an ATP system.
- Syntactic requirements for input and output of ATP systems.

As a consequence, the TPTP syntax is very popular in ATP community. According to TPTP syntax, a TPTP first order formula is defined similarly as a normal first order formula excepts :

- The complete restriction for variable names, constants names, function names and predicate names.
- The symbols $\rightarrow$ and $\leftrightarrow$ are replaced by => and <=> respectively.
- The symbols $\wedge$ and $\vee$ are replaced by & and | respectively
- The symbols $\forall$ and $\exists$ are replaced by ! and ? respectively.
- The set of quantified variables is denoted by $[x_1, ..., x_n]$
- If $\phi$ is a formula then so are $![x_1, ..., x_n] : (\phi)$ and $?[x_1, ..., x_n] : (\phi)$
- $\phi$ a binary connective $\psi$ is formula iff $\phi$ and $\psi$ are unitary formulas where an unitary formula is an atomic formula, a quantified formula or a formula which is bounded by ( and ).

An example of a TPTP first order formula is given as follows:

**Example 5.1**

$$?[X] : (lives(X) \& killed(X, agatha))$$

### 5.1.4 Common Logic syntax

Common Logic is a framework for logic-based languages in order to standardize syntax and semantic for knowledge exchange in information systems. It has been recognized as an ISO standard [18] since 2007. This standard includes the abstract syntax and semantic of Common logic which actually basing on first order predicates and its application for following dialects :

- Common Logic Interchange Format (CLIF)
- Conceptual Graph Interchange Format (CGIF)
- XML-based notation for Common Logic (XCL)

In this part we consider only Common Logic Interchange Format. The syntax of a CLIF formula is defined recursively as follows :

- If $t$ is a variable or a constants then $t$ is a term
- If $f$ is a function with arity $n$ and $t_1, ..., t_n$ are terms then $(ft_1...t_n)$ is a term
- If $P$ is a predicate with arity $P$ and $t_1, ..., t_n$ are terms then $(Pt_1...t_n)$ is an atomic formula
- $t_1$ and $t_2$ are term then $(= t_1t_2)$ is an atomic formula
- If $\phi$ is an formula then so is $(not\ \phi)$
- If $\phi$ and $\psi$ are formula then so is $(bc\ \phi\ \psi)$ where $bc \in \{and, or, if, iff\}$
- If $\phi$ is a formula and $x_1, ..., x_n$ are variables then $(exists(x_1...x_n)\phi)$ and $(forall(x_1...x_n)\phi)$ are formulas.

An example of a CLIF formula is given as follows:

**Example 5.2**

*(exists (x y) (and (Red x) (not (Ball x)) (On x y) (not (and (Table y) (not (Blue y))))))*

## 5.2 Design and support tools

### 5.2.1 Logic formula parser

Listing 5.1: An example of ANTLR grammar

```
grammar Sample;                          //Name of gramma

options {
  language = Java;                       //Language of generated code
}

@header {
  package a.b.c;                         //Package for parser class
}

@lexer::header {                         //Package for lexer class
  package a.b.c;
}

program                                  //Top level rule
        :       'program' IDENT '='
                (constant | variable | function | procedure | typeDecl)*
                'begin'
                statement*
                'end' IDENT '.'
        ;

constant                                 //Rule
        :       'constant' IDENT ':' type ':=' expression ';'
        ;
```

In this section, we shall take into account the tool under the designing process where all necessary softwares and techniques will be considered as well. To fulfill the software interfaces requirement, we will choose Java language to build our tool. Since the input of the tool is first order formulas entered by users then building a good logic formula parser is one of crucial tasks. However, we use Java as the programming language where only arithmetic expressions are parsed automatically, therefore we have to build the parser by ourself. In fact, the parser has to be sufficient because the system need to '*understand*' fully the meaning of each sub part of the formula. General speaking, building this kind of parsers from scratch is not so easy. Fortunately, there is a good software to support us building the parser automatically named ANTLR [2].

ANTLR stands for ANother Tool for Language Recognition, which is a framework for '*constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages*' such as : Java, C++, Python, and so on [2]. In the other words, with the support of ANTLR, users just need to define the grammar by introducing grammar rules and then ANTLR will generate parser code for corresponding language. General speaking, ANTLR provides:

- A complete grammar development environment named ANTLRWorks.
- Complete and detail tutorials for the syntax of grammars. An example of ANTLR grammar is given in listing 5.1
- An integrated grammar development framework for Eclipse with visualized interpreters.

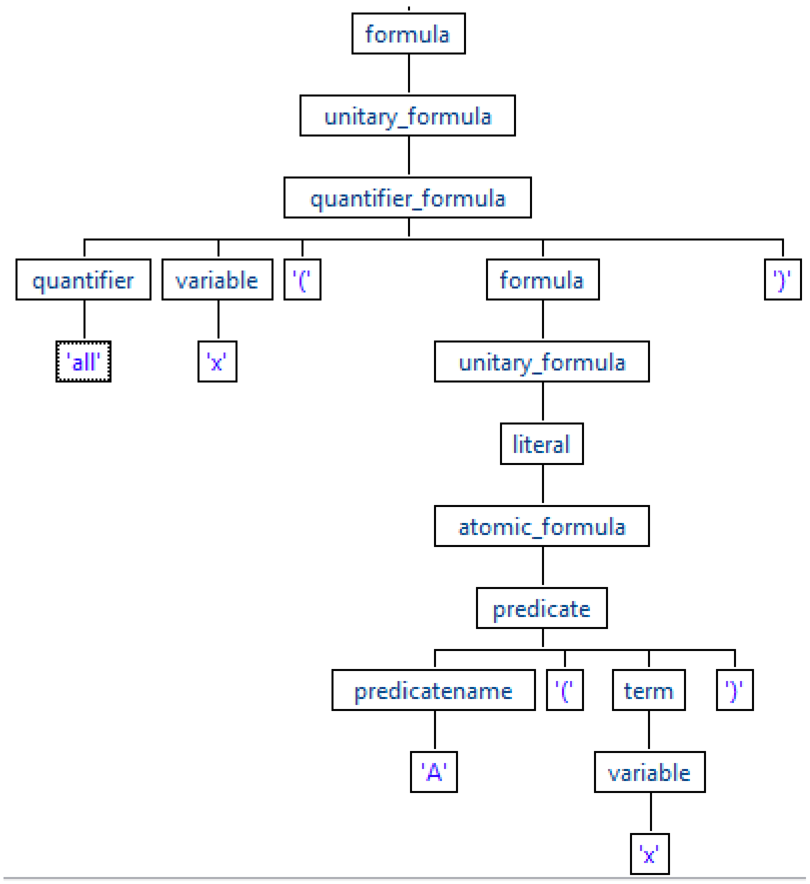


Figure 5.1: Interpreter for '*all x (A(x))*'

With the support of ANTLR, grammar of a first order formula can be defined recursively corresponding to normal first order syntax. Obviously, the restriction for name of predicates, variables, constants and parentheses should be unambiguous in order to have a correct parser. After defining grammar, developers also can embed code for data structure corresponding to the formula such as trees or recursive classes.

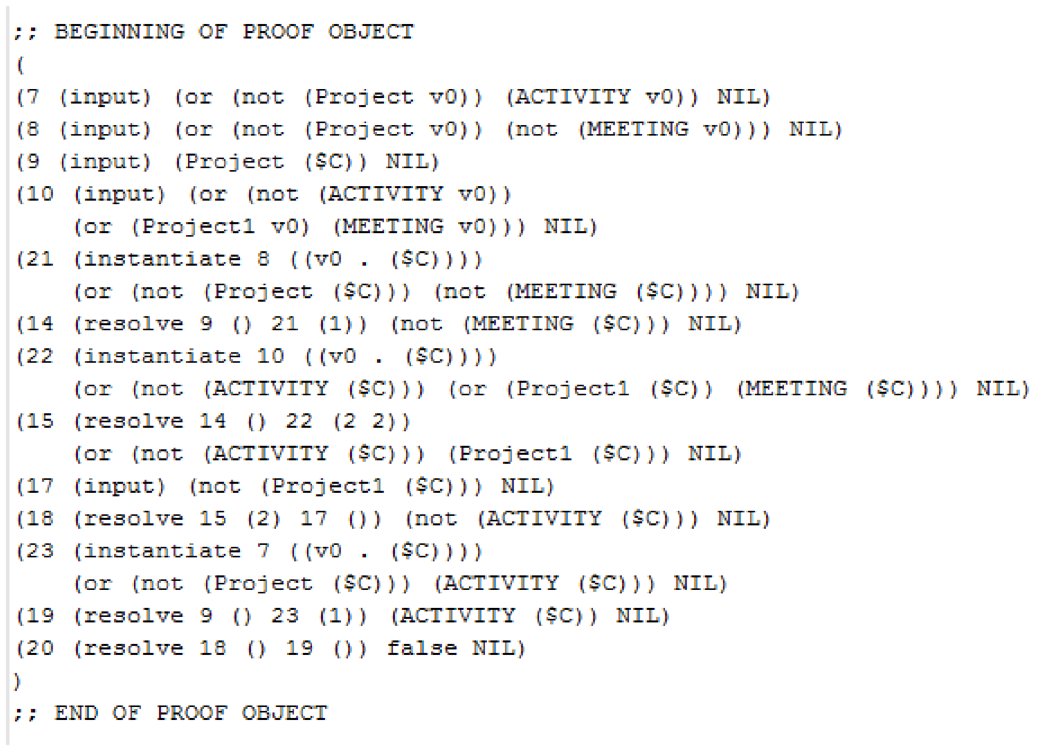
For example, given a complete grammar and a string '*all x(A(x))*', a interpreter for this string is describe in a tree as in the figure 5.1

### 5.2.2 Prover9

From previous chapters, we all know that the problem of checking implicit definability and finding explicit definition can be reduced to the problem of finding a proof for a valid formula.

More precisely, in order to find an explicit definition, one has to know a detail proof and then construct an interpolant from this proof based on some methods mentioned in chapter 3. In order to perform it, we do not build an algorithm for scratch but use some available automated theorem proving systems. In general, a prover where the tool can be built on top should have the following characteristics:

- Deciding whether a formula is valid or not in a short time.
- Giving a detail proof in a well formed structure such that it is '*understandable*' by a computer program.
- Providing interface to communicate with other systems.
- Having good referent documents, manuals as well as a good supporting community.



```
;; BEGINNING OF PROOF OBJECT
(
(7 (input) (or (not (Project v0)) (ACTIVITY v0)) NIL)
(8 (input) (or (not (Project v0)) (not (MEETING v0))) NIL)
(9 (input) (Project ($C)) NIL)
(10 (input) (or (not (ACTIVITY v0))
    (or (Project1 v0) (MEETING v0))) NIL)
(21 (instantiate 8 ((v0 . ($C))))
    (or (not (Project ($C))) (not (MEETING ($C)))) NIL)
(14 (resolve 9 () 21 (1)) (not (MEETING ($C))) NIL)
(22 (instantiate 10 ((v0 . ($C))))
    (or (not (ACTIVITY ($C))) (or (Project1 ($C)) (MEETING ($C)))) NIL)
(15 (resolve 14 () 22 (2 2))
    (or (not (ACTIVITY ($C))) (Project1 ($C))) NIL)
(17 (input) (not (Project1 ($C))) NIL)
(18 (resolve 15 (2) 17 ()) (not (ACTIVITY ($C))) NIL)
(23 (instantiate 7 ((v0 . ($C))))
    (or (not (Project ($C))) (ACTIVITY ($C))) NIL)
(19 (resolve 9 () 23 (1)) (ACTIVITY ($C)) NIL)
(20 (resolve 18 () 19 ()) false NIL)
)
;; END OF PROOF OBJECT
```

Figure 5.2: Ivy output of a Prover9 proof

Based on above criterias we do not choose some tableau based provers for description logic like: Pellet, Fact++ because these provers do not return detail proofs. In terms of resolution systems, there are several candidates such as: Vampire, Prover9, E. Among them, we choose Prover9 and do not pick either Vampire or E even they are faster because their documentation and interface for other systems are quite poor.

Prover9 is a resolution/paramodulation automated theorem prover for first order logic developed by William McCune. Prover9 system not only contains a prover with graphic user interface but also includes:

- Mace4: which searches for finite models and counterexamples. In the other words, given the same input, Prover9 will try to find a proof while Mace4 will attempt to find a counterexample.
- LARD: which is command line version of Prover9, Mace and other programs. It is also considered as an interface of Prover9 with other systems such that using a set com commands, other systems can give inputs to Prover9 and then receive outputs.

- Prooftrans: which reads standard proofs returning by Prover9 and converts them in different ways such as: renumber steps, simplify justifications, expand all steps, turning secondary justifications into explicit steps, produce proofs in XML and produce proofs for checking by the IVY proof checker.

Related to the returned proof of Prover9, normal XML with expanding all steps is a good candidate since we do not have to build a parser for it. However, unification steps are not explicitly described in this form. Fortunately, Prooftrans of Prover9 also returns an output containing these steps named Ivy (an example of Ivy output is given in the figure 5.2), which is actually can be used to check with Ivy proof server. Since this output is just plain text, we need to build a parser for it with the help of ANTLR.

### 5.2.3 Espresso

Another task that we have to concern in our tool is simplifying logic formulas since the methods to compute interpolant mentioned in chapter 3 might return very long formulas which have many redundant subformulas. In general for first order logic, formula optimization in terms of numbers of used predicates, variables, connective symbols is a hard problem. Therefore, in this tool we take into account only the problem of simplifying a propositional logic formula.

In fact, simplifying a propositional logic formula can be reduced to the problem of optimization a boolean circuit, which is a crucial problem for digital circuit designing. Among several approaches to deal with this problem, we select Espresso [5] - a library developed at IBM. To reduce the complexity of the problem, Espresso uses some specific heuristic algorithms. As a consequence, the result might not be the global minimal one but in most case it is very close to the global minimal formula.

### 5.2.4 Architecture

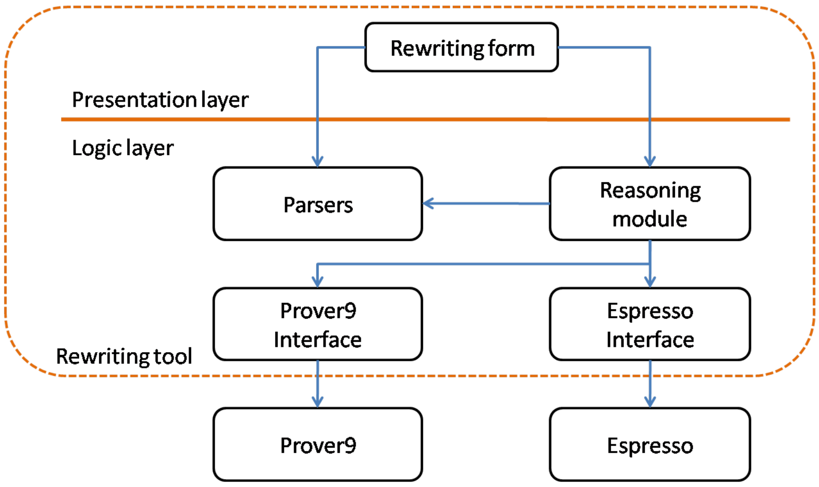


Figure 5.3: Rewriting tool architecture

Along with the support of the tools mentioned above, we can give 2-layers architecture of the tool as in the figure 5.3 where the specification of each layer is indicated as follows:

- *Presentation layer*: This layer provides user interface where users can enter the input and receive the output. Since the input and output of this tool are not so complicated then we just need only one form named Rewriting form to handle this task.
- *Logic layer*: This layer handles all logic operations of the program. It contains the following modules:

– Parsers: This module does all the parsing task included parsing input formulas, returned proofs of Prover9 as well as some mediated types of formulas.
– Reasoning module: The main reasoning services such as checking implicit definability, finding explicit definition, checking consistency, checking safe range are containing in this module. In order to do handle these tasks, this module has to call services/functions from other modules in the same layer.
– Prover9 interface: As reflected in the name, this module behaves as a communicator between the tool and Prover9. It creates standard inputs of Prover9, calls to Prover9 and receives responses.
– Espresso interface: Similar to above module, this module connects to Espresso library and receives answers.

### 5.2.5 Package diagram

In this subsection we consider the package diagram of the tool corresponding to the above architecture. This diagram is in the figure 5.4 where:

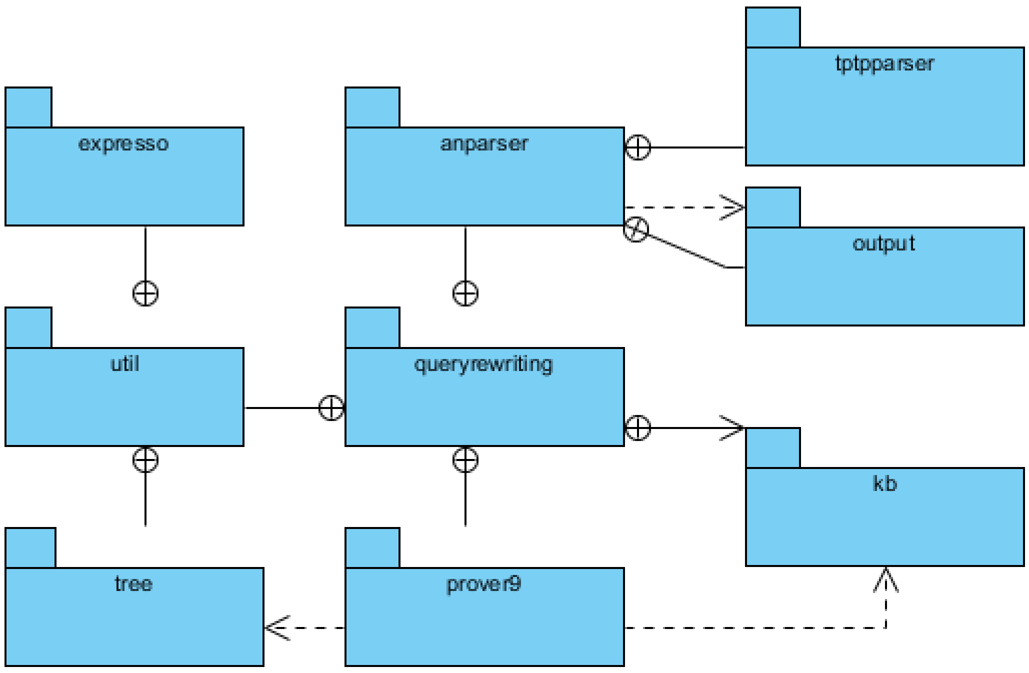


Figure 5.4: Package diagram

- *queryrewriting*: is the top package, which contains other packages as well as classes for user interface such as : QueryRewritingView and QueryRewritingApp.
- *queryrewriting.anparser*: is the package which contains all generated classes from ANTLR as well as classes for data structures corresponding to these parsers. It includes: KB parser for normal formulas, Ivy parser for the ivy output returned by Prover9, CL parser for Common Logic formula and Interpolant parser for intepolant formulas.
- *queryrewriting.kb*: is the package including all operations related to knowledge base and query. It contains: KB and Query classes as well as some operations to create output files and process input files of the tool.
- *queryrewriting.prover9*: is the package containing all classes related to Prover9 such as: Prover9 class for input, output of Prover 9 and Interpolant class for computing interpolant from Prover9 outputs.

- *queryrewriting.until*: is the package which includes some supported services for others such as: Tree structure and Espresso as mentioned in previous sub sections.

**Algorithm 1:** Get explicit definition of Q over KB

```
Input: String KB; String Query
Output: a string: which is an error or the rewritten query
error = parse(KB, Query) ;
if error != null then
    return error ;
end
else
    error = checkSyntax(KQ, Query) ;
    if error != null then
        return error ;
    end
    else
        PDB = selectDatabasePredicates(KB) ;
        KB' = createRenameKB(KB, PDB );
        Query' = createRenameQuery(KB, PDB);
        imDefine = isImplicitlyDefinable(KB,Q,KB',Q');
        if !imDefine then
            return 'Query is not implicitly definable' ;
        end
        else
            proof= getProver9Proof(KB,ground(Q),KB',ground(Q')) ;
            if proof = empty then
                return 'Cannot rewrite the query' ;
            end
            predicateA = getPredicateA(PDB) ;
            predicateB = getPredicateB(PDB) ;
            interpolant = getInterpolantMatrix(proof,predicateA,predicateB) ;
            termA = getMaximalTermA(interpolant);
            termB = getMaximalTermB(interpolant);
            interpolant = getInterpolant(interpolant, termA, termB) ;
            return interpolant ;
        end
    end
end
```

## 5.3 Implementation

In this subsection we shall describe some important algorithms in the tool as well as several sample blocks of codes.

### 5.3.1 Algorithms

The first algorithm is introduced here is the general algorithm for our tool where the input is a string corresponding to knowledge base and a string corresponding to a query; and the output is an error, information about reason why the query is not rewritten or the rewritten query. The whole algorithm can be referred from algorithm 1. Notice that, in order to simulate the scenario where the set of database predicates is a subset of predicates mentioned in KB, in this algorithm, we use function *selectDatabasePredicates* over a knowledge base. Moreover, we also explicit explain the process of making input for Prover9 such as: query $Q$ is implicitly definable iff $KB \wedge KB' \models \forall X(Q \leftrightarrow Q')$ and the interpolant is computed from the proof of $KB \wedge ground(Q) \rightarrow (KB' \rightarrow ground(Q'))$. In the last part of the algorithm, we consider the transformation from a relational interpolant to the Craig one according to the method of Huang mentioned in chapter 3.

The second algorithm taken into account is the algorithm of computing local interpolant of a resolvent based on interpolants of its resolved clauses. This algorithm 2 stimulates exactly the method of Huang described in 3.2.3 for this particular case.

**Algorithm 2:** Get local interpolant for resolution rule

```
**Input**: Clause $C_1 = C \vee L$, Clause $C_2 = D \vee \neg L'$, predicateA, predicateB
**Output**: Local interpolant $P$ of $C_1$ and $C_2$
$interpolant1 = getInterpolant(C_1)$;
$interpolant2 = getInterpolant(C_2)$;
$\pi = getMGU(L, L')$;
if $L$ is from predicateA alone then
    $interpolant = (interpolant1 \vee interpolant2)\pi$ ;
end
if $L$ is from predicateB alone then
    $interpolant = (interpolant1 \wedge interpolant2)\pi$ ;
end
if $L$ is from both predicateA, predicateB then
    $interpolant = ((\neg L' \wedge interpolant1) \vee (L \wedge interpolant2))\pi$ ;
end
return $interpolant$ ;
```

## 5.3.2 Sample codes

Related to implementation, we shall show in this part some blocks of codes such as: how to create a knowledge base from an input string, how to compute interpolant matrix from a proof and how to create Craig interpolant from this matrix.

**Create a knowledge base**

Listing 5.2: An example of ANTLR grammar

```
case NORMAL:
try {
        CharStream stream = new ANTLRStringStream(input);
        KBLexer lexer = new KBLexer(stream);
        TokenStream tokenStream = new CommonTokenStream(lexer);
        KBParser parser = new KBParser(tokenStream);
        KBOutput out = new KBOutput();
        formulas = parser.kb(out);
}
/* Excepion handle code*/
break;
```

**Finding interpolant matrix**

Listing 5.3: An example of ANTLR grammar

```
String interpolant = "";
/* Get type code*/
if(node.getNumberOfChildren() == 0) //Input clause
{
        boolean isInA = isInputA(step);
        if(isInA) interpolant = "$false";
        else interpolant = "$true";
}
else //If not
{
        switch(type)
        {
                case IvyOutput.Type.FLIP :
                        interpolant = flipInterpolant(node);
                        break;
                case IvyOutput.Type.INSTANTIATE:
                        interpolant = instInterpolant(node);
                        break;
                case IvyOutput.Type.PRO:
                        interpolant = propInterpolant(node);
                        break;
                case IvyOutput.Type.RESOLVE:
                        interpolant = resolveInterpolant(node);
                        break;
                case IvyOutput.Type.PARAMOD:
                        interpolant = paramodeInterpolant(node);
                        break;
        }
}
InterOutput.Formula f = createInFormulaFromString(interpolant);
return f.toString();
```

**Create interpolant from the matrix**

Listing 5.4: An example of ANTLR grammar

```
Node node = tree.getRootElement();
String in = interpolant(node);
InterOutput.Formula f = createInFormulaFromString(in);
/* Shorten code using Espresso*/
in = shortenInterpolant(f);
f = createInFormulaFromString(in);
/* Get and arrange maximal terms code*/
String temp = "x";
for(int i= arranged.size() /2 - 1; i >= 0; i--)
{
        /*Replaced term by xi code*/
}
for (int i=0; i< arranged.size()/2; i++)
{
        String var = temp + i;
        if((Integer)arranged.get(i*2 + 1) == 1)
        {
                prefix = prefix + "exists␣" + var + "(";
                endfix = endfix + ")";
        }
        else
        {
                prefix = prefix + "all␣" + var + "(";
                endfix = endfix + ")";
        }
}
return prefix + s + endfix;
```

## 5.4 Usage notes

In this first version of the rewriting tool, a simple graphic interface was implemented using normal user interaction components such as: menu, textbox, combobox, listbox and dialog.

### 5.4.1 Setup

In order to setup the tool, users should follow below steps:

- Make sure that Java 1.5 (or higher) and Prover9 are installed in your computer.
- Unrar the **Package.rar** in a folder in your computer.

- Open the **rewriting.properties** file by a text editor and modify *PROVER9_DIR* entry to your the path of Prover9 in your computer.
- Run the **rewriting.bat** file to start the tool.

After that, the query rewriting window will appear as in the picture 5.5.

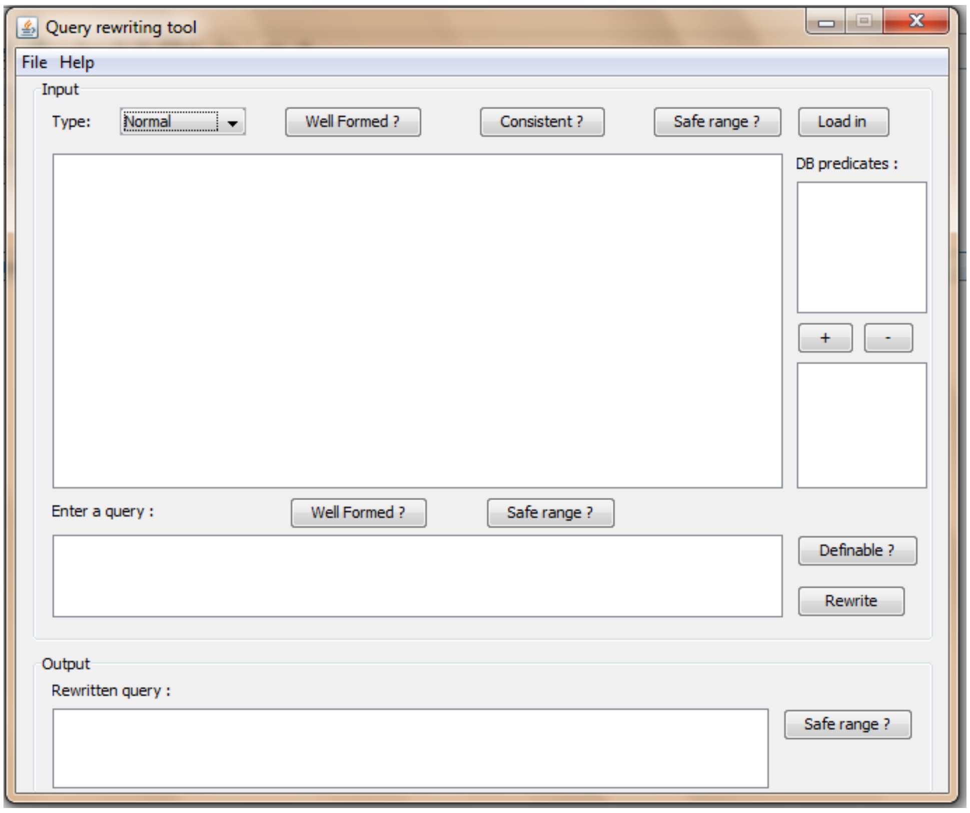


Figure 5.5: Query rewriting window

### 5.4.2 Tool manipulations

1. **Open/Save**: With these operations users can save their input in a text file or open a text file which contains input information such knowledge base and query.
   Go to File $->$ Open and select a file to open an input file
   Go to File $->$ Save and choose file name to save (saved information might includes : knowledge base, query and rewritten query)

2. **Input types**: There are three types of input formulas contained in the commbobox Type such as: Normal, TPTP and CLIF. Users can choose one of them. Notice that the input type and the input should be corresponding to each other, if not, an error will be generated after parsing.
   For syntax of TPTP and CLIF formula, readers can refer to previous sections. For the syntax of Normal formula, users can see below example :

   **Example 5.3**

   $$\textit{all } x, y(A(x, y) \rightarrow B(y)).$$

   Notice that in normal syntax, constants begin with $ and an upper char while predicates begin with an upper char and variables begin with a lower char.

3. **KB operations**: Users can edit the knowledge base on the biggest text box. In order to check whether the knowledge base is well form, safe range or consistent or not, users just need to click to corresponding buttons. An dialog of result will be showed immediately after that.
   In order to load the KB into the system, users have to click to Load in button. After loading, the list of all predicates will appear at the first DB predicates list box. The second list contains all database predicates. To add a database predicate, users click to this item in the first list and click (+) button. To remove a database predicate, users click to this item in the second list and click (-) button.

4. **Query operations**: Similar to knowledge base, users can edit a query in query text box and check whether it is well-formed or not by clicking to Well-formed button. To check whether the query is definable with respect to the knowledge and selected predicates, users click to Definable button. To rewrite the query, users click to Rewrite button. If a rewritten one is found, it will be showed up in Rewritten query text pane.
   Notice that the syntax checking is included in all operations.

Chapter 6

# Conclusion and future works

In this thesis we studied a general framework to rewrite a query over a knowledge base and a set of database predicates. Relied on Craig interpolant theorem and Beth definability for first order logic, we confirmed that it is possible to construct an explicit definition of a query which is implicitly definable in theoretical point of view, where interpolant is always constructed from a proof of validity. Moreover, based on this framework we also clarified some conditions to reduce the problem of query answering over a knowledge base and a database to the model checking problem over the database, which actually can be solved by standard relational database techniques. By studying some important decidable fragments of first order logic such as description logic $\mathcal{ALC}$ and its generalization guarded fragment, we made sure that our conditions are not unambiguous.

Besides, considering the transformation of a first order query to relational one, we did a research on a crucial property of this transformation named safe range. We pointed out that having a safe range rewritten query over a safe range knowledge base and safe range query is not trivially. However, we also showed that it is possible if we add more axioms to the knowledge base with the meaning that every individual has to belong to some categories.

In practical point of view, we proved that our framework can be incorporated with some theorem provers in the sense of automated rewriting by implementing a rewriting tool. With the support of this tool, one will know whether the query is definable or not and if it is the case a definition might be showed up. Besides, the tool also helps users explore the safe range property of the rewritten query. As an additional feature, our piece of software supports several types of popular syntaxes for first order formulas as well in order to exchange results with other systems in future.

In the rest of this chapter, which is also the last main part of this thesis, we would like to mention some interesting future works of our research.

## 6.1 Domain independence

As we mentioned in chapter 4, domain independence of the rewritten query is the essential property to reduce the problem of general query answering to model checking problem. Therefore, finding the conditions of knowledge base and original query to have a domain independent rewritten one is a very important task. We showed that if the knowledge base and query are safe range then there is a domain independent rewritten query. However, we are still exploring properties of input in order to have a stronger result such as the output is always domain independent no matter which method is used to compute it.

## 6.2 Safe range for resolution method

Another open question related to our research is safe range property of rewritten query computed by resolution method. In fact, the prenex form of the interpolant computed based on resolution proof as in Huang paper (mentioned in 3.2.3) might destroy its safe range property. Therefore, it might helf if we find an alternative way to compute Craig interpolant for first order logic which is not in prenex form. Besides, finding a method to transform a prenex formula to a normal one which has maximal part is safe range is also an interesting approach.

## 6.3 Optimization

From the practical perspective, we all agree that there might be many rewritten queries of one original query and the problem of how to get the optimized one in terms of query evaluation is very important. In fact, one has to take into account which criteria should be used to optimize, such as: the size of the rewritings, the numbers of used predicates, the priority of predicates and the number of relational operators.
Moreover, duplication of semantic in rewritings according to knowledge base should be considered as well, because it does not only affect the query evaluation process but also gives options to rewrite. For example, consider a simple knowledge base as follows:

$$\text{all } x, y(V(x,y) <-> (x = y)).$$
$$\text{all } x, y(V1(x,y) <-> V(x,y)).$$

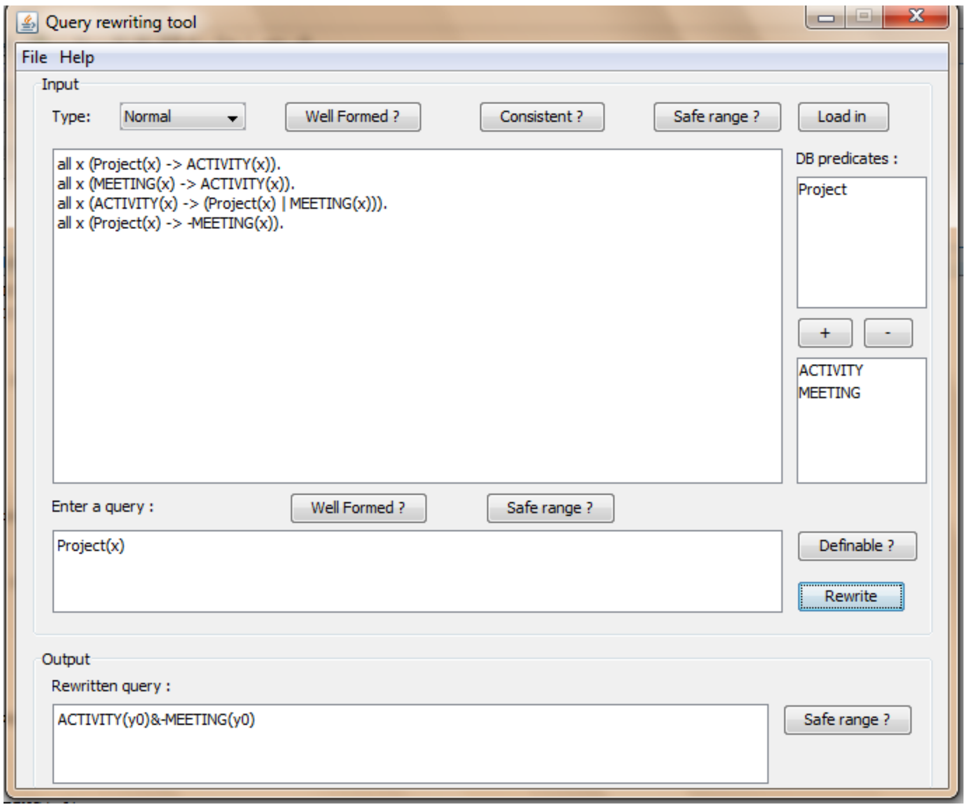
and a query $V1(x,y)$. A rewriting of this query over above knowledge base and database predicate $V$ might be $V(x,y) \wedge (x = y)$, where its conjuncts are equivalent to each other according to the knowledge base.

## 6.4 Decidable fragment

One more interesting topic for future works is exploring more decidable fragments of first order logic where the framework can be applied. Notice that the balance between the decidable property and the expressiveness of knowledge base and queries has to be considered. As mentioned in [4], we also can check the application of framework where knowledge base is presented in one fragment and queries are in the different one, such as: description logic knowledge base with positive first order queries.

# Appendix A

# GUI screenshots



```
all x (Project(x) -> ACTIVITY(x)).
all x (MEETING(x) -> ACTIVITY(x)).
all x (ACTIVITY(x) -> (Project(x) | MEETING(x))).
all x (Project(x) -> -MEETING(x)).
```

DB predicates : Project

ACTIVITY
MEETING

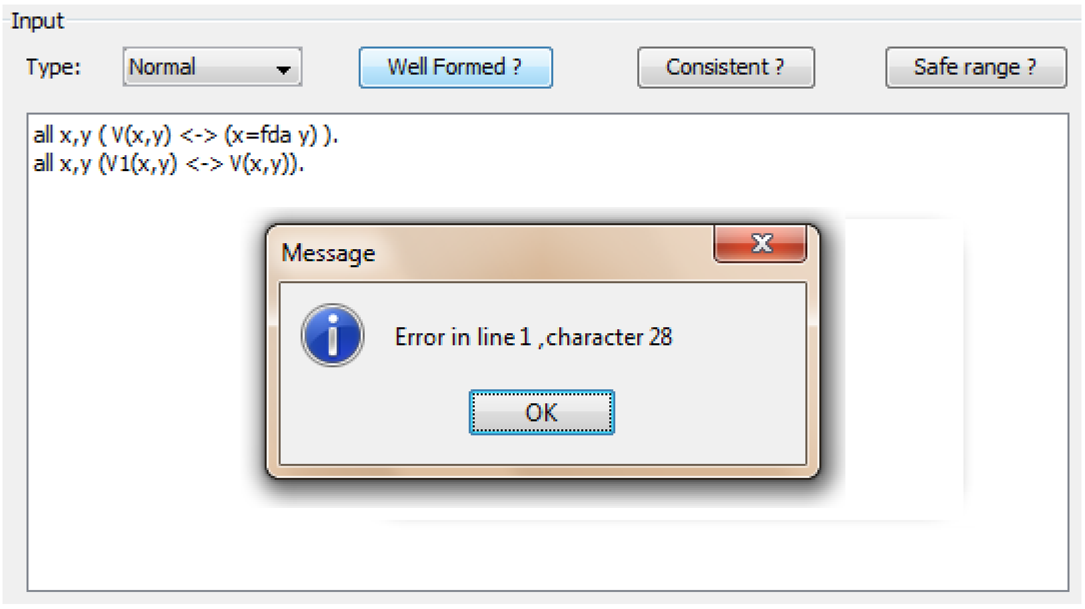
Enter a query :

```
Project(x)
```

Rewritten query :

```
ACTIVITY(y0)&-MEETING(y0)
```

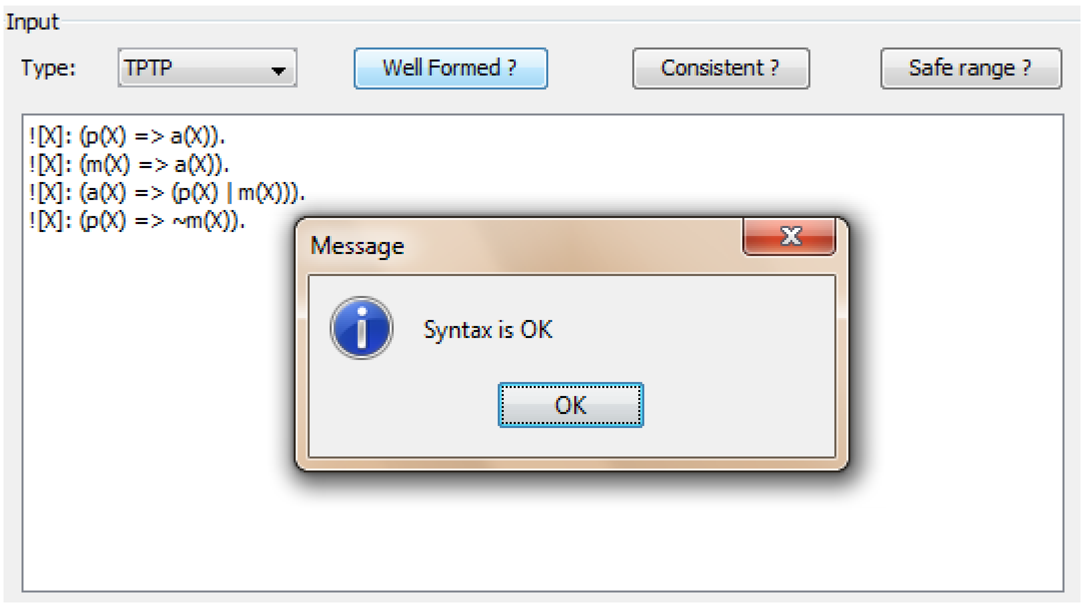
Figure A.1: A successful scenario of the rewriting framework



```
all x,y ( V(x,y) <-> (x=fda y) ).
all x,y (V1(x,y) <-> V(x,y)).
```

Error in line 1 ,character 28

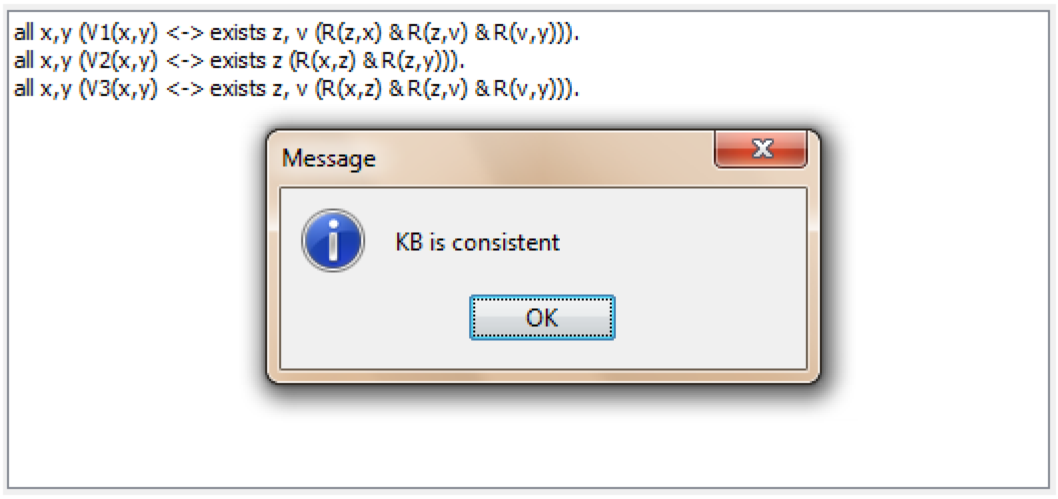
Figure A.2: An example about wrong syntax of input formulas



```
![X]: (p(X) => a(X)).
![X]: (m(X) => a(X)).
![X]: (a(X) => (p(X) | m(X))).
![X]: (p(X) => ~m(X)).
```

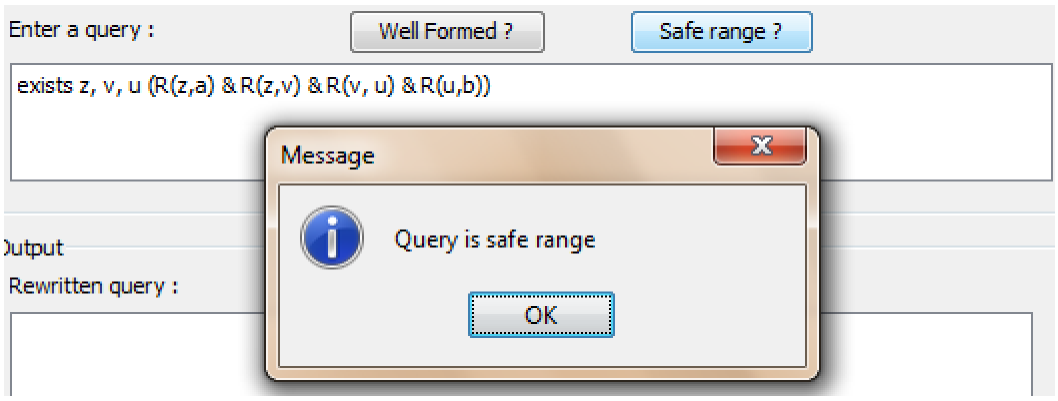
Syntax is OK

Figure A.3: An example about correct syntax of output formulas



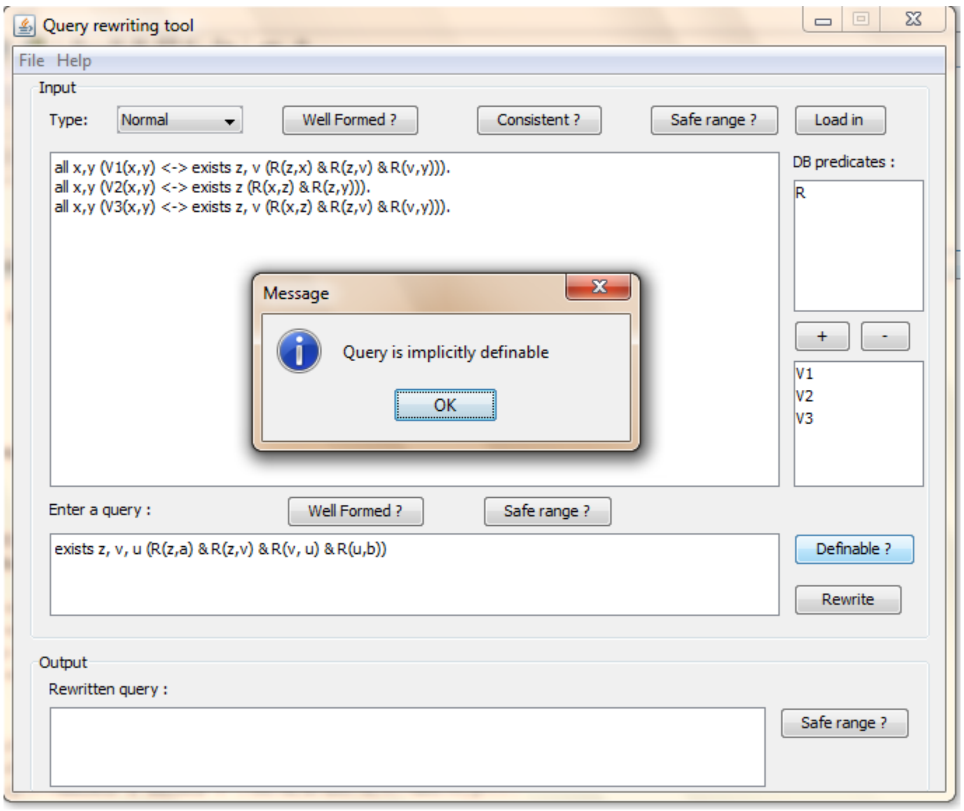
```
all x,y (V1(x,y) <-> exists z, v (R(z,x) & R(z,v) & R(v,y))).
all x,y (V2(x,y) <-> exists z (R(x,z) & R(z,y))).
all x,y (V3(x,y) <-> exists z, v (R(x,z) & R(z,v) & R(v,y))).
```

Figure A.4: An example of a consitent knowledge base



```
exists z, v, u (R(z,a) & R(z,v) & R(v, u) & R(u,b))
```

Figure A.5: An example of a safe range query



```
all x,y (V1(x,y) <-> exists z, v (R(z,x) & R(z,v) & R(v,y))).
all x,y (V2(x,y) <-> exists z (R(x,z) & R(z,y))).
all x,y (V3(x,y) <-> exists z, v (R(x,z) & R(z,v) & R(v,y))).
```

DB predicates: R

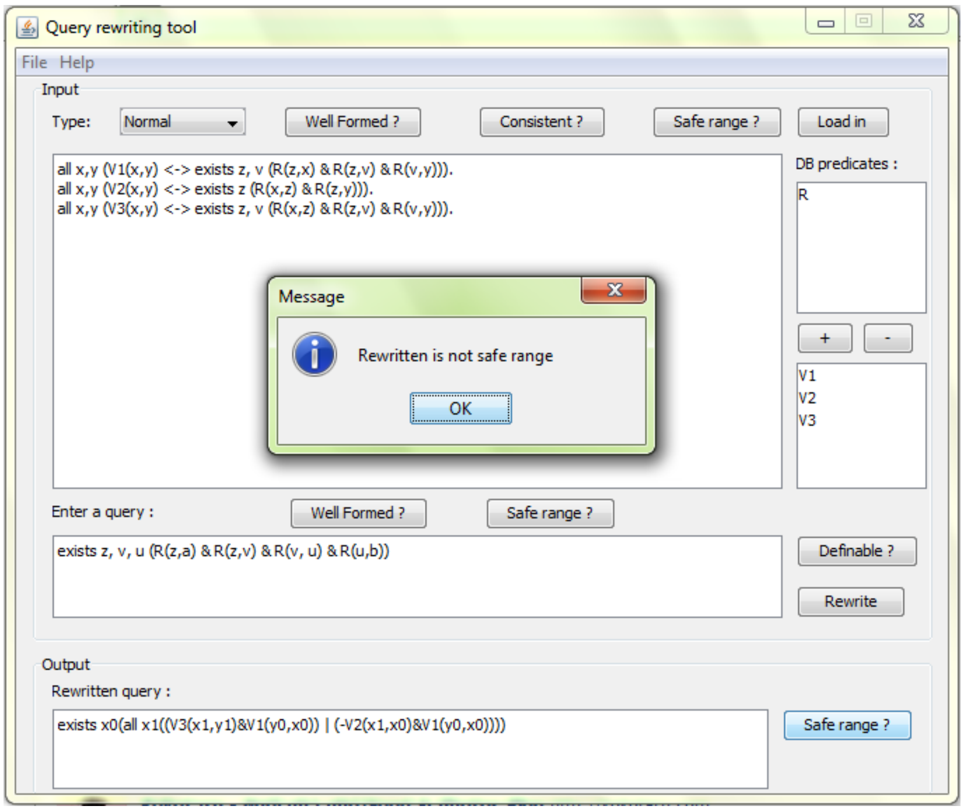
V1
V2
V3

Query is implicitly definable

Enter a query:

```
exists z, v, u (R(z,a) & R(z,v) & R(v, u) & R(u,b))
```

Figure A.6: A positive result for Nash's example in [22]



```
all x,y (V1(x,y) <-> exists z, v (R(z,x) & R(z,v) & R(v,y))).
all x,y (V2(x,y) <-> exists z (R(x,z) & R(z,y))).
all x,y (V3(x,y) <-> exists z, v (R(x,z) & R(z,v) & R(v,y))).
```

Rewritten is not safe range

```
exists z, v, u (R(z,a) & R(z,v) & R(v, u) & R(u,b))
```

```
exists x0(all x1((V3(x1,y1)&V1(y0,x0)) | (-V2(x1,x0)&V1(y0,x0))))
```

Figure A.7: Rewriting of Nash's example in [22] and its safe range property

# Appendix B

# Worked out examples

Let us consider the aforementioned example in chapter 1 to demonstrate behaviors of our rewriting framework. So we have:
The set of predicates: *Student* ($S$), *GradStudent* ($G$) and *UnderGradStudent* ($U$)
The knowledge base (KB) is:

$$\begin{aligned}\forall x(Student(x) \rightarrow (GradStudent(x) \vee UnderGradStudent(x)))\\ \forall x(GradStudent(x) \rightarrow Student(x))\\ \forall x(UnderGradStudent(x) \rightarrow Student(x))\\ \forall x(GradStudent(x) \rightarrow \neg UnderGradStudent(x))\end{aligned}$$

The database predicates are: *Student* and *UnderGradStudent*
The query $Q(x)$ is: $GradStudent(x)$

## B.1 Check definability

First, we will show that the query is definable from database predicates by proving

$$KB \wedge \widetilde{KB} \rightarrow \forall x(Q(x) \leftrightarrow \widetilde{Q}(x)) \tag{B.1}$$

where:
$\widetilde{KB}$ is:

$$\begin{aligned}\forall x(Student(x) \rightarrow (GradStudent(x) \vee UnderGradStudent(x)))\\ \forall x(GradStudent(x) \rightarrow Student(x))\\ \forall x(UnderGradStudent(x) \rightarrow Student(x))\\ \forall x(GradStudent(x) \rightarrow \neg UnderGradStudent(x))\end{aligned}$$

$\widetilde{Q}(x)$ is: $\widetilde{GradStudent}(x)$.
Let $G_1$ is a rename of $\widetilde{G}$. The following is the proof of B.1 returned by Prover9:

Listing B.1: An example of ANTLR grammar

```
1 (all x (S(x) -> G(x) | U(x))) # label(non_clause).  [assumption].
2 (all x (G(x) -> S(x))) # label(non_clause).  [assumption].
4 (all x (G(x) -> -U(x))) # label(non_clause).  [assumption].
5 (all x (S(x) -> G1(x) | U(x))) # label(non_clause).  [assumption].
6 (all x (G1(x) -> S(x))) # label(non_clause).  [assumption].
8 (all x (G1(x) -> -U(x))) # label(non_clause).  [assumption].
9 (all x (G(x) <-> G1(x))) # label(non_clause) # label(goal).  [goal].
```

```
10 -G(x) | S(x).   [clausify(2)].
11 -S(x) | G(x) | U(x).   [clausify(1)].
13 -S(x) | G1(x) | U(x).   [clausify(5)].
14 -G1(x) | S(x).   [clausify(6)].
16 G(c1) | G1(c1).   [deny(9)].
17 -G(x) | -U(x).   [clausify(4)].
18 -G(c1) | -G1(c1).   [deny(9)].
19 G1(x) | U(x) | -G(x).   [resolve(13,a,10,b)].
20 -G1(x) | G(x) | U(x).   [resolve(14,b,11,a)].
21 G1(c1) | U(c1) | G1(c1).   [resolve(19,c,16,a)].
22 -G1(x) | -U(x).   [clausify(8)].
23 G1(c1) | -U(c1).   [resolve(16,a,17,a)].
24 -G1(c1) | U(c1) | -G1(c1).   [resolve(20,b,18,a)].
25 G1(c1) | G1(c1) | G1(c1).   [resolve(21,b,23,b)].
26 G1(c1).   [copy(25),merge(b),merge(c)].
27 -G1(c1) | -G1(c1) | -G1(c1).   [resolve(24,b,22,b)].
28 $F.   [copy(27),merge(b),merge(c),unit_del(a,26)].
```

## B.2 Compute interpolant

### B.2.1 Resolution based method

Now we compute interpolant of $KB \wedge Q(c)$ and $\widetilde{KB} \rightarrow \widetilde{Q}(c)$ based on the resolution proof of $(KB \wedge Q(c)) \rightarrow (\widetilde{KB} \rightarrow \widetilde{Q}(c))$ as follows:

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{-G(x) \vee S(x)^{[\perp]} \quad -S(x) \vee G_1(x) \vee U(x)^{[\top]}}{G_1(x) \vee U(x) \vee -G(x)^{[S(x)]}} \quad G(c)^{[\perp]}
}{G_1(c) \vee U(a)^{[S(c)]}} \quad -G_1(c)^{[\top]}
}{U(c)^{[S(c)]}}
\quad
\dfrac{G(c)^{[\perp]} \quad -G(x) \vee -U(x)^{[\perp]}}{-U(c)^{[\perp]}}
}{F^{[-U(c) \wedge S(c)]}}
$$

where $clause^{[I]}$ means $I$ is local interpolant of $clause$
Then, $S(c) \wedge \neg U(c)$ is our desirable interpolant

### B.2.2 Tableau based method

In tableau based method, we start with the initial set of biased formulas:

$$
\begin{aligned}
S_0 = \{ & L(\forall x(S(x) \rightarrow (G(x) \vee U(x)))),\\
& L(\forall x(G(x) \rightarrow S(x))),\\
& L(\forall x(U(x) \rightarrow S(x))),\\
& L(\forall x(G(x) \rightarrow \neg U(x))),\\
& L(G(c)),\\
& R(\forall x(S(x) \rightarrow (G_1(x) \vee U(x)))),\\
& R(\forall x(G_1(x) \rightarrow S(x))),\\
& R(\forall x(U(x) \rightarrow S(x))),\\
& R(\forall x(G_1(x) \rightarrow \neg U(x))),\\
& R(\neg G_1(c))\}
\end{aligned}
$$

Apply the rule for $\forall$ and remove implication, we have:

$$\begin{aligned}
S_1 = \{&L(\neg S(c) \vee G(c) \vee U(c)), && \text{(B.2)}\\
&L(\neg G(c) \vee S(c))), && \text{(B.3)}\\
&L(\neg U(c) \vee S(c)), && \text{(B.4)}\\
&L(\neg G(c) \vee \neg U(c)), && \text{(B.5)}\\
&L(G(c)), && \text{(B.6)}\\
&R(\neg S(c) \vee G_1(c) \vee U(c)), && \text{(B.7)}\\
&R(\neg G_1(c) \vee S(c)), && \text{(B.8)}\\
&R(\neg U(c) \vee S(c)), && \text{(B.9)}\\
&R(\neg G_1(c) \vee \neg U(c)), && \text{(B.10)}\\
&R(\neg G_1(c))\} && \text{(B.11)}
\end{aligned}$$

where interpolant of $S_1$ can be computed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{S_4 \cup \{R(\neg S(c)\}^{S(c)} \quad S_4 \cup \{R(U(c))\}^{\neg U(c)}}{S_4 = S_3 \cup \{R(\neg S(c) \vee U(c))\}^{(S(c)\wedge\neg U(c))}}\,\vee \quad S_3 \cup \{R(G_1(c))\}^{\top}}{S_3 = S_2 \cup \{L(\neg U(c))\}^{(S(c)\wedge\neg U(c))}}\,B.7 \quad S_2 \cup \{L(\neg G(c))\}^{\perp}}{S_2 = S_1 \cup \{L(S(c))\}^{(S(c)\wedge\neg U(c))}}\,B.5 \quad S_1 \cup \{L(\neg G(c))\}^{\perp}}{S_1^{(S(c)\wedge\neg U(c))}}\,B.3}{}$$

Notice that the notion of $set^I$ is coincident with the notion of $set \longrightarrow I$ as in 3.2.1. Therefore $S(c) \wedge \neg U(c)$ is the interpolant that we need to compute.

## B.3 Get rewriting

By replaceing constant $c$ with variable $x$, we have $\widehat{Q}(x) = Student(x) \wedge \neg UnderGradStudent(x)$

# Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] ANTLR. Another tool for language recognition. `http://www.antlr.org/`, 2005.

[3] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[4] Alex Borgida, Jos Bruijn, Enrico Franconi, Inan Seylan, Umberto Straccia, David Toman, and Grant Weddell. On finding query rewritings under expressive constraints, 2009.

[5] Robert Brayton. Espresso. `http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm`, 1993.

[6] CASC. The world championship for automated theorem proving. `http://www.cs.miami.edu/~tptp/CASC/`, 2010.

[7] Chen Chung Chang and Jerome Keisler. *Model Theory.* Number 73 in Studies in Logic and the Foundations of Mathematics. North-Holland, 1973. Third edition, 1990.

[8] Jan Chomicki. Database consistency: Logic-based approach, slides. `http://www.cse.buffalo.edu/~chomicki/talks-bolzano08.pdf`, 2008.

[9] William Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.

[10] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[11] Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[12] Enrico Franconi. Using first order logic, slides for description logic course. `http://www.inf.unibz.it/~franconi/dl/course/slides/logic/fol/fol-2.pdf`, 2009.

[13] Dov Gabbay and Larisa Maksimova. *Interpolation and Definability Volume1.* Clarendon Press, 2004.

[14] E. Hoogland. *Definability and Interpolation: Model-theoretic investigations.* PhD thesis, 2001.

[15] Eva Hoogland, Maarten Marx, and Martin Otto. Beth definability for the guarded fragment. In *LPAR '99: Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, pages 273–285, London, UK, 1999. Springer-Verlag.

[16] Guoxiang Huang. Constructing craig interpolation formulas. In *COCOON '95: Proceedings of the First Annual International Conference on Computing and Combinatorics*, pages 181–190, London, UK, 1995. Springer-Verlag.

[17] IBM. Query rewriting methods and examples. `http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/admin/c0005293.htm`, 2009.

[18] ISO. Iso standard for common logic. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=39175`, 2007.

[19] Alexander Leitsch. The resolution calculus, alexander leitsch. *J. of Logic, Lang. and Inf.*, 7(4):499–502, 1998.

[20] Roger C. Lyndon. An interpolation theorem in the predicate calculus. *Pacific J. Math.*, 1:129–142, 1959.

[21] K. L. Mcmillan. Interpolation and sat-based model checking. pages 1–13. Springer, 2003.

[22] Alan Nash, Luc Segoufin, and Victor Vianu. Determinacy and rewriting of conjunctive queries using views: A progress report. In *ICDT*, pages 59–73, 2007.

[23] Andrei Popescu, Traian Florin erbnu, and Grigore Rou. A semantic approach to interpolation. *Theor. Comput. Sci.*, 410(12-13):1109–1128, 2009.

[24] Pavel Pudlk. Lower bounds for resolution and cutting plane proofs and monotone computations, 1996.

[25] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[26] Inanç Seylan, Enrico Franconi, and Jos de Bruijn. Effective query rewriting with ontologies over dboxes. In *IJCAI*, pages 923–925, 2009.

[27] tptp. The tptp problem library for automated theorem proving. `http://www.cs.miami.edu/~tptp/`, 2010.

[28] WIKI. Data integration. `http://en.wikipedia.org/wiki/Data_integration`, 2010.

[29] Lawrence Wos, Daniel Carson, and George Robinson. The unit preference strategy in theorem proving. In *AFIPS '64 (Fall, part I): Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, pages 615–621, New York, NY, USA, 1964. ACM.

[30] Lawrence Wos, George A. Robinson, and Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, 1965.