# The Efficiency and Implementation of an Evaluation-Based Reasoning Procedure with Disjunctive Information in First-Order Knowledge Bases

Diplomarbeit im Studiengang Informatik von

Horst Samulowitz
Matr.-Nr. 214 063

Lehr- und Forschungsgebiet Informatik V
Prof. G. Lakemeyer, Ph.D.
Rheinisch-Westfälische Technische Hochschule Aachen

Erstgutachter: Prof. G. Lakemeyer, Ph.D.
Zweitgutachter: Prof. Dr. J. Giesl

# Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht.

Aachen, den 31. Juli 2003

# Contents

In this thesis we investigate a deductive reasoning procedure that can handle incomplete first-order knowledge bases which contain disjunctive information. Mainly, the expressiveness of the underlying first-order logic and the large amount of supported data ($\geq 10^5$ terms) are the essential features of the logical sound reasoning procedure of concern.

We discuss several properties of the reasoning procedure itself and apply some changes that are also used in our implementation. Besides implementing the evaluation-based reasoning procedure, our work investigates the efficiency of this kind of deductive reasoning when employing large datasets.

The ability to apply deductive reasoning efficiently on a first-order knowledge base that consists of a large set of facts, rules, incomplete knowledge and disjunctive information is the main topic discussed in this work.

# Chapter 1

# Introduction

## 1.1 The Trade-Off between Expressiveness and Efficiency

Knowledge in the sense of AI requires more than knowledge about the world that is sufficient to allow acting in the domain of discourse in an appropriate way. In AI a knowledge-based system additionally should behave and act like it does because it makes use of the representation of that knowledge (e.g., world-knowledge) [41]. In other words, it is required that an intelligently interacting computer system needs a large body of knowledge about the world known as common sense [48]. The idea to provide computer systems with such kind of knowledge by representing knowledge explicitly is known as the *Knowledge Representation Hypothesis* [66].

It has been argued that at least first-order logic is necessary to represent world-knowledge [50]. Especially, the ability to handle incomplete knowledge like disjunctive information plays a major role to model the suggested knowledge in an appropriate way. Since it is necessary that a knowledge-based system must be able to infer implicit knowledge and it is a well-known fact that classical logical implication is undecidable in the first-order case there is a problem. There exist various approaches that deal with that problem, but none of them can satisfy both expressiveness and efficiency. Note that while reasoning in first-order knowledge bases is intractable in general humans can reason very effectively on extremely large and complex datasets, although, of course, they cannot draw all possible conclusions.
In the context of knowledge representation reasoning is in general a formal manipulation of the symbols that represent the facts and believed proposi-

tions[1] to produce representations of *new* propositions [41].

In this work we apply *deductive* reasoning in incomplete first-order knowledge bases that is logical sound. Deduction is in some sense the direct application of knowledge in the production of implicit knowledge [69].
Until now only the query evaluation over databases supports deductive reasoning on very large first-order knowledge bases efficiently [40, 36]. But a classical database is a knowledge base that allows no incomplete knowledge since it makes use of the closed world assumption (CWA) [56]. In general, a database is equivalent to a maximally consistent set of function-free ground literals [36].
For example, if we have a database that contains information about students at a university, we could ask if there is a student from Argentina. Then this query would be answered positively only if the database would contain a fact or an entry that supports the query explicitly. If the facts are not explicitly contained in the database the query is answered negatively because of the CWA. In fact, no further reasoning takes place at all.

Now suppose the following terms to be contained in a knowledge base:

$$(isArgentinan(Mary) \vee isArgentinan(John)).$$
$$isStudent(Mary).$$
$$isStudent(John).$$

This kind of disjunctive information cannot be handled by the relational algebra used in classical database. But we would like to be able to answer queries like "Is there a student from Argentina?", which would require the ability to handle disjunctive information.

In our work we use a deductive reasoning procedure that supports incomplete knowledge and in particular disjunctive information contained in the knowledge base. For instance, the just mentioned query would be answered positively by the reasoning procedure of concern even while we do not explicitly know whether Mary or John are from Argentina. The only thing we know is that one of them is from Argentina, but this fact is sufficient to answer the query positively.
At this point it is important to mention that we are interested in large knowledge bases, say more than $10^5$ terms. This guarantees that relatively complex knowledge can be modeled since we support both a huge set of terms and the necessary expressiveness introduced by the underlying first-order logic. At the same time the deductive reasoning procedure should be tractable.
Since logical sound and complete deductive reasoning is undecidable in general, the used deductive reasoning procedure is logical sound, but not complete to preserve tractability [36]. The trade-off between the properties of the

[1] An idea that is expressed by a simple declarative sentence [41].

reasoning procedure and the tractability of the procedure is the main key to maintain efficiency while supporting logical sound reasoning on incomplete knowledge as well on disjunctive information.

In fact, the deductive reasoning procedure that is of concern here places itself between the efficient databases that support only very restricted reasoning capabilities and the undecidable and intractable reasoning procedures that are used in theorem provers for example.
In this work we discuss the efficiency of the reasoning procedure. Therefore, we determine how efficient reasoning with incomplete first-order knowledge bases that contain disjunctive information is when using the given reasoning procedure. In other words, we determine if the underlying logic and the reasoning procedure itself are too expressive to maintain efficiency at the same time.

We will implement a logical sound and decidable reasoning procedure named X that can handle disjunctive information in first-order knowledge bases. The reasoning procedure itself was introduced in [36]. In addition, we will prove several properties of the reasoning procedure concerning its efficiency and present experimental results.

## 1.2 Related Work

### 1.2.1 Reasoning with Incomplete First-Order Knowledge Bases

In [46] the tractability for reasoning with incomplete first-order knowledge bases is discussed. It could be shown, that the efficiency of deductive reasoning with incomplete first-order knowledge bases is comparable with classical query evaluation in databases if the knowledge base is of a specific format. Note that this result does not hold for arbitrary first-order knowledge bases.

The general idea was to reduce a deductive reasoning procedure to database query evaluation. To reach this result a bottom-up database query evaluation algorithm was adapted. The underlying query evaluation algorithm is mainly based on work introduced and extended by [3, 29].

The reader is referred to the original paper. At this moment, the reader should only be aware of the fact that there exists a tractability result for reasoning with incomplete first-order knowledge bases. Note that *no* disjunctive information is contained in the knowledge base.

### 1.2.2 Limited Reasoning

There exists various work that presents limited forms of reasoning in both the propositional case and the first-order case.
In the propositional case the classical logical entailment is restricted most frequently by only allowing a limited use of *Modus Ponens* or by not supporting it at all [45]. For limited reasoning there exist two logical languages in general. Namely, the classical language with a adapted semantic or a modal language with a belief operator. The belief operator leaves the classical sublanguage that deals with the classical semantic unchanged, but implements at the same time an entailment that is weaker than that of the classical language [45]. For instance, suppose a belief operator named B. Then entailment corresponds to the validity of formulas like $(B\alpha \supset B\beta)$ in contrast to the classical entailment based on the validity of formulas like $(\alpha \supset \beta)$.
In [12, 14, 15, 24] the classical language is used with a non-standard semantic to provide a decidable reasoning procedure. For instance, [12] implements a tractable proof system that is based on a non-deterministic truth table and whose entailment consists mainly of unit propagation. On the other hand in [33, 39] reasoning is accomplished within a newly introduced logic of belief. The proposed belief implication in [39] that uses two modal operators to handle implicit and explicit belief is tractable for formulas in conjunctive normal form (CNF). The reasoning is mainly based on tautological entailment, a fragment of relevance logic [2].
But especially the reasoning based on tautological entailment could not be transferred in an appealing way from the propositional case into the first-order case [36] which will be explained in the next chapter. In [35, 55] the first-order case was discussed and it could be shown that the reasoning required not only considerable machinery, but additionally the expressiveness was decreased at the same time [36, 45].

In conclusion, these approaches are not a solution yet since they are inefficient or support only an inadequate expressiveness or both. Further details concerning limited reasoning can be found in [45].

### 1.2.3 Datalog

In contrast to limited reasoning there also exist approaches that allow full inference, but restrict the underlying logical language at the same time. One example for this kind of approach is Datalog which we will discuss here briefly. In general, the family of knowledge representation languages known as *description logics* belong to this approach. For an in-depth survey on description logics the reader is referred to [4].

*Datalog* is a simplified logical programming language that is integrated in database management [22]. The term Datalog refers to PROLOG-like rules without function symbols that are treated like logic programs [70]. Recall, that rules are equivalent to Horn-clauses. From a database point of view, Datalog is an extension to the relational algebra that allows recursion [22].

In general, the used predicates can be divided into two groups. The first group consists of the *extensional* predicates that are relations contained in the database and the second group consists of *intensional* predicates or rules contained in the Datalog program [70, 22].
Hence, a relational database is identified with a set of ground clauses or facts [17]. The Datalog program - consisting of a set of rules - uses the relational database as input to answer queries [17]. In fact, an answer to a query is a resulting database that contains ground clauses gained by the corresponding Datalog program applied on the original database.

There exist several extensions to the original Datalog like presented in [71], but this issues are not further addressed here. The computational complexity was examined in [1]. It could be shown that Datalog only captures queries that can be answered in polynomial time.
At this point it is only important to note that Datalog is restricted to Horn-clauses and the CWA is still used. Therefore, the expressiveness of Datalog is not as powerful as the one that is supported by the underlying logic of the reasoning procedure presented here.
An extensive overview concerning the entire relationship between logic and databases is given in [49].

### 1.2.4 Propositional Satisfiability

The following sections give a brief introduction to the Satisfiability Problem (SAT), its solutions and why it is essential to take a look at the algorithms that solve SAT albeit we deal with first-order logic here.

The aim of this section is to show how much research is done in the propositional case and how difficult it is to validate the efficiency of a new algorithm in this area even while there exist a lot of benchmarks and competitions [57]. In contrast, there do not exist any benchmarks and results that correspond to knowledge bases and deductive reasoning procedures as they are suggested in this work here.

#### The Satisfiability Problem (SAT)

SAT is the problem of deciding if there is an assignment for variables in a propositional formula that makes the formula true. Even if we deal with first-

order logic here, it is clear that propositional satisfiability (SAT) is closely related to deductive reasoning in first-order logic. Simply, because of the fact that the propositional case is a sub-problem of first-order logic. The border between propositional and first-order reasoning is often blurred [26]. One example are problems that make use of quantified formulas but are constrained to finite domains with explicitly named domain elements.

SAT was the first problem shown to be NP-complete [10] and therefore intractable in general. Since then - 1971 - there has been a large amount of research concerning the satisfiability problem. Moreover it could be shown that there are many practical instances of SAT that can be solved very efficiently.

There are several huge groups of researchers involved into SAT like the AI research and the theorem proving group. Many AI problems like planning [31] are encoded into SAT quite naturally. Theorem proving is for example concerned with satisfiability since the question if a formula $\varphi$ is inferable from a set of formulas $\Phi$ can be answered by showing that $\Phi \cup \neg\varphi$ is not satisfiable and vice versa.

### Solving the Problem

There exist two essential different goals for methods that solve SAT - methods that claim to be complete and those which are incomplete or approximate. A complete algorithm is the famous David-Putnam algorithm [18]. Unfortunately, *all* complete algorithms are exponential in space or time. As long as $P \neq NP$ holds, it is not feasible to overcome this intractability in general, but researchers all over the world are highly motivated to improve their algorithms as far as possible.

Many complete algorithms are based on the David-Putnam algorithm and implement extensions like branching heuristics, intelligent backtracking, parallelization, etc. [26]. An upper bound for the original David-Putnam algorithm is $O(1.696^N)$ where $N$ is the number of variables. But extensions like presented in [5] can solve problems easy which are beyond the scope of the normal David-Putnam algorithm and it is furthermore suggested that it performs as good or better than stochastic SAT algorithms in most of the cases.

While [58] and the improvement made in [60] introduce an algorithm with the best known running time for randomized 3-SAT to date, [16] actually presents a *deterministic* algorithm for k-SAT based on local search that runs in time $1.481^N$ up to a polynomial factor. Additionally, these bounds seem to be better than all previous deterministic k-SAT algorithms could obtain.

Aside from complete algorithms there are many approximate algorithms

[26]. Firstly, [34] presents a greedy algorithm that chooses truth assignment at random. It is greedy in the following sense: it flips the truth value of a variable that increases the number of satisfied clauses. Flipping the truth assignment of a variable without raising the number of satisfied clauses is called a sideway move.

While in [34] no sideway moves are allowed and flipping is repeated until no improvement is possible, [61] comes up with an algorithm called GSAT that allows sideway flips. Starting with a random truth assignment, it changes the variable assignment via hill climbing to the largest possible number of satisfied clauses. If there is no assignment that does not change the number of satisfied clauses, sideway moves are allowed. Without sideway moves the performance of GSAT degrades immensely. [28] shows that a huge part of search is concerned with exploring large plateaus where sideway flips predominate. GSAT guarantees relatively good performance even on large instances of SAT [61]. But note that no deductive tasks are accomplished by GSAT and it works only on problems that can be formulated in a propositional language [40].

There are a lot of implementations that are based on GSAT. Some algorithms make use of clause weights [51] and can achieve good improvements when applied to certain classes of problems. GSAT with random walk [63] flips a variable with probability $p$ and otherwise hill-climbs normally. WalkSAT [63] makes the idea of GSAT with random walk even more central to the algorithm. Hill Climbing returns the variables in an unsatisfied clause, and the next flip of a variable is based on random or greediness [26].

Although simulated annealing is a famous local search algorithm it is not that popular for solving SAT even when [68] says that it works better than GSAT on hard random 3-SAT problems. Surprisingly, other approaches that use neural networks and genetic algorithms to solve SAT are comparably rare [26]. In [67] there is a Hopfield Network introduced that works very well on hard 3-SAT problems. One further interesting approach are hybrid methods that make use for example of GSAT and the Davis-Putnam algorithm to solve special classes of SAT [72].

### The Benchmark Problem

As shown above there are lots of algorithms that solve SAT. The problem is to classify those algorithms by there efficiency, because some algorithms might work very well on some instances of SAT but underlie an exponential blow up on other instances.

Besides there are many different opinions on how to characterize a hard and easy instance of SAT. The conventional picture drawn is like *easy-hard-*

*easy*. Formulas with few clauses are *under-constrained* [*easy to solve*] and hence have many satisfying assignments. Therefore it is easy to find a satisfying assignment. Formulas with very much clauses are *over-constrained* [*easy to solve*] and usually unsatisfiable which will lead to a fast search too [64].
Formulas lying in between are the so called *critical constrained* [*hard to solve*] formulas and much harder to solve, because they have relatively few satisfying assignments if they have any at all. '*...the hardest area for satisfiability is near the point where* 50% *of the formulas are satisfiable*' as said in [64]. Empirical concluded in [64] is that the region of 50% satisfiable clauses occurs at a fixed ratio of the number of clauses to the number of variables.

Very hard instances of SAT outcrop when the number of clauses is 4.3 times the number of variables . This phenomena is called a phase transition. Recent research has shown that if the computing time grows polynomially with problem size a continuous transition is found, but a discontinuous transition is observed when exponentially much time is required [52].

In [27] it is said that this conventional drawn picture is inadequate. There are problems not lying in the classical phase transition region that can be even harder than those lying in the median of the transition. Relying on experimental data it is suggested that there are regions which underlie a *constraint gap*, where the number of constraints on variables is minimal while simultaneously the depth of search required to solve the problems is maximal.

Hence, while it is not obvious where the hardest instances of SAT are hidden and it seems that satisfiability testing might be quite easy on average [64], this section should emphasize how difficult it is to verify having an efficient algorithm developed.

Last thing to be mentioned is that there is a suggested general format (like from [19]) to save SAT instances so that researchers can easily exchange for example hard instances and do not overcome the fault to choose instances at random, because a randomly chosen instance of SAT will be easy to solve with utmost probability. This has the advantage that algorithms can be compared in a fair and broad sense.

As we will see later on we are *not* able to compare our results with other approaches, because there do not exist incomplete first-order knowledge bases of this type and reasoning procedure of this expressiveness as suggested here [36]. We do not even have the opportunity to make use of an existing knowledge base. Simply, because of the fact that appropriate test knowledge bases do not exist. For instance, it would be of considerable advantage if we could make use of the Cyc knowledge base [13] or a suitable subset.

## 1.3 Thesis Structure

This thesis is structured as follows:

- In **Chapter 1** we motivate our work, introduce deductive reasoning informally and present related work while we concentrate especially on research done in the propositional case.
- **Chapter 2** introduces the required notations and elementary definitions to enable a deeper understanding of the reasoning procedure. Additionally, all main features of the reasoning procedure are explained here.
- In **Chapter 3** we discuss several properties of the reasoning procedure itself. Especially, the use of inequalities plays a major role in this chapter.
- In **Chapter 4** we present the implementation of all major features included in the reasoning procedure. This chapter also contains various examples to clarify the proceeding of the algorithms presented.
- **Chapter 5** contains several results concerning the efficiency of our approach. We show that the characteristic of the knowledge base has a major influence on the efficiency of the reasoning procedure.
- In **Chapter 6** we assess the work that was done during this project and present ideas for future research.

Additionally, we like to mention that we assume that the reader is familiar with basic first-order logic and PROLOG, or any other similar logical programming language.

# Chapter 2

# Fundamentals

## 2.1 Introduction

In this chapter we introduce the basics required to enable a complete understanding of the reasoning procedures presented some subsections later. Furthermore the logical language corresponding to the reasoning procedure is introduced briefly.

But we will not only discuss notation and definitions here, but we will additionally present $V$ - the precursor of the reasoning procedure $X$ which is of concern here. We do so, because we would like to provide a step by step introduction.
While $V$ can handle some incomplete knowledge by not using the CWA [56], $X$ is even able to handle disjunctive information additionally. Since both procedures operate on first-order knowledge bases it is a known fact that a complete logical reasoning procedure would be *undecidable* when allowing incomplete knowledge. Therefore both procedures are *incomplete* but *decidable* and can handle incomplete knowledge and disjunctive information respectively.
At the end of this chapter we will introduce $X$ itself.

While it is not necessary to follow each detail of the procedure $V$, it is crucial for the following chapters to get in touch with the notations, definitions and the equations of $X$. Especially, the features *Unit Propagation* and *Reasoning by Cases* introduced by $X$ should be understood very well since they belong to the main topics of the entire work. So, even if the reader is only interested in the practical part of this work he or she has to go through this.

We use the same notations and definitions as presented in [40, 36]. Readers that are common with this literature may skip the entire chapter.

Here, we only summarize the main ideas underlying the work by Lakemeyer and Levesque. For an in-depth survey the reader is referred to the original papers [40, 36].

## 2.2 First-Order Knowledge Bases

The reasoning procedure is based on a standard first-order language $\mathcal{L}$ without function symbols except constants and an equality predicate. While making use of the unique name assumption [25] an infinite set of constants $\mathcal{C}= \{c_1, c_2, ...\}$ is assumed.

---

**Notation**

First of all elements of $\mathcal{L}$ are called formulas and formulas without free variables are called sentences. The standard symbols for quantifiers, negations, etc. are used while only $\neg$, $\vee$, $\exists$ belong to the logical language. In some examples we will use $\wedge$, $\forall$ and $\supset$ as an abbreviation. In addition, $\forall$ is for instance used as an abbreviation for $\forall x$.

The symbol $l$ will range over literals and $\overline{l}$ will express its complement. $\theta$ will range over substitutions of all variables by constants, so that $\alpha\theta$ is the result of applying the substitutions to $\alpha$.

Furthermore, $\alpha_d^x$ denotes $\alpha$ with all occurrences of the variable $x$ substituted by the domain constant $d$. The symbol $\rho$ will range over atoms whose arguments are distinct variables, so that $\rho\theta$ will range over ground atoms. Note that neither atoms nor literals include equalities.
Finally, $e$ will mean quantifier-free formulas whose only predicate is equality.

---

Furthermore it is assumed that quantification is interpreted substitutionally with regard to $\mathcal{C}$. This assumption is founded on the fact how standard interpretations of $\mathcal{L}$ are defined:

**Definition** A standard interpretation $I$ of $\mathcal{L}$ is one where equality ($=$) is interpreted as identity, and the denotation relation between $\mathcal{C}$ and the domain of discourse is bijective.

This kind of standard interpretation can be described by the following set of axioms about equality given that the considered logical theory only considers a finite number of constants. Of course finite knowledge bases will always fulfill this restriction.

**Definition** The set $\varepsilon$ is the axioms of equality, which includes equivalence relation and substitution of equals of equals, and the set of formulas $\{c_i = c_j \mid i \neq j\}$.

On those definitions and assumptions the interrelationship between the satisfiability of equality and a closed formula and the existence of a standard model was proven [40]. The following theorem states this coherence:

**Theorem 1** *Suppose $S$ is any set of closed formulas, and that there is an infinite set of constants that do not appear in $S$. Then $\varepsilon \cup S$ is satisfiable iff it has a standard model.*

Since we now introduced the essential properties of the logical language $\mathcal{L}$, we are able to present the definition of the form of the used first-order knowledge bases. The following definition belongs to knowledge bases where $V$ operates on.
As you will see the definition does not allow clauses and thereby no disjunctive information at all. When introducing $X$ later on the definition is extended to contain disjunctive information.

**Definition** A knowledge base is called *proper* if $\varepsilon \cup KB$ is consistent and $KB$ is finite and of the form $\forall(e \supset \rho)$ or $\forall(e \supset \neg\rho)$.

For example the following term would be a valid entry in a *proper KB*:

$$\forall(X \neq a \supset \neg P(X))$$

In contrast, $\forall(X \neq a \supset P(X) \vee Q(X))$ is *not* a valid entry since clauses are not allowed (yet). Please note that equality terms may consist of any kind of logical combination like negations, disjunctions and conjunctions.

## 2.3 A Deductive Reasoning Procedure

While deductive reasoning was introduced briefly in the last chapter, we now get step by step in touch with the deductive reasoning procedures to be used in first-order knowledge bases that are of concern in this work here.

As already said in the introduction we think that there exists an high demand in AI to work with extremely large knowledge bases that hold more than $10^5$ facts [36]. Currently there are very few deductive reasoning procedures that can handle such kind of sets. As seen in the introduction GSAT [28] and other methods are capable to perform good results on huge data

sets, but can not achieve deductive tasks and are restricted to propositional languages [40]. Again, the following sections are mainly based on [40, 36].

In the following sections reasoning procedures are introduced that are capable of working on large knowledge bases. Although the knowledge bases are large they are restricted in their expressiveness.
While the first procedure presented handles quantifiers, equality and incomplete knowledge and a knowledge base ($KB$) that consists of function-free ground literals, the second procedure introduced allows function-free ground *clauses* in a knowledge base in addition. By supporting clauses the last mentioned decision procedure is able to handle disjunctive information.

### 2.3.1 The V-Procedure

#### Introduction

$V$ is a deductive reasoning procedure that works on *proper* first-order knowledge bases by evaluating the query. In contrast to a normal database the knowledge base may contain both complete and incomplete knowledge. In fact this is the one and only difference to common databases - the closed world assumption (CWA)[56] does not hold anymore. Even though this might be interpreted as no big difference - only allowing incomplete knowledge - the price to pay is high.
This increase in expressive power makes complete logical reasoning *undecidable* in the first-order case. For example, the knowledge base that is equivalent to an empty set of literals requires that all valid formulas must be known when the CWA is not longer assumed. This problem is already co-NP hard in the propositional case and undecidable in the first-order case [40]. Therefore $V$ is an incomplete but decidable and logical sound reasoning procedure that can handle incomplete knowledge.

As said before it is not necessary to read the following subsections, but it is recommended to get a better feel for deductive reasoning and on the proceeding of such a procedure if you are not common to it. In the following subsection we glimpse at $V$ when presenting the corresponding equations as defined in [40] and give some further explanations and examples.

#### The Equations

As can be seen from the following equations the reasoning procedure determines the return value of a query by evaluating it. When evaluating an existential we make use of the set $H_k^+$ that contains the union of constants, that appear in the knowledge base and the query and $k$ new constants that

are not mentioned in the knowledge base or the query.
$V$ uses a 3-valued answer from $\left\{0, \frac{1}{2}, 1\right\}$. Returning 0 means *known to be false*, $\frac{1}{2}$ means *unknown* and 1 means *known to be true*.
Note that we do not say that the procedure decides if a query is *true* or *false*. The return values only state answers that are implied by the reasoning mechanism used in $V$ which is due to the fact that the reasoning accomplished by $V$ is not complete for arbitrary queries. We give a simple example to show incompleteness some lines later.

The first step in the reasoning procedures consists of the recursive decomposition of the query by the matching equations of the reasoning procedures provided that the query is composed one.
For example the query $P(x) \vee Q(x)$ is decomposed by $V$ to $V[P(x)]$ and $V[Q(x)]$ (see equation 2.4). Besides negation also the existential is decomposed by using the set $H_1^+$ to substitute the free variables in the query by constants (see equation 2.5).

As can be seen in the equations, decomposition has an quite intuitive impact on the return value of $V$. For example, when handling a disjunctive query the maximum value of the disjunctive parts is returned, a negation simply causes the inverse return value.
After decomposing a formula $\alpha$ the single parts of the formula are reduced to ground atomic formula by substituting free variables by domain constants $(\rho\theta)$.
In general the knowledge encoded in a query - respectively a formula - will be reduced to knowledge of ground atomic formulas.

---

$$V[\rho\theta] = \begin{cases} 1 & : \text{ if there is a } \forall(e \supset \rho) \in \text{KB such that } V[e\theta] = 1 \\ 0 & : \text{ if there is a } \forall(e \supset \neg\rho) \in \text{KB such that } V[e\theta] = 1 \\ \frac{1}{2} & : \text{ otherwise} \end{cases} \tag{2.1}$$

$$V[t = t\prime] = 1 \text{ if } t \text{ is identical to } t\prime, \text{ and 0 otherwise.} \tag{2.2}$$

$$V[\neg\alpha] = 1 - V[\alpha]. \tag{2.3}$$

$$V[\alpha \vee \beta] = \max\{V[\alpha], V[\beta]\}. \tag{2.4}$$

$$V[\exists x.\alpha] = \max_{d \in H_1^+}\{V[\alpha_d^x]\}. \tag{2.5}$$

---

At this point the procedure determines the return value by deciding if the term $(e \supset \rho)$ is contained in the knowledge base whereby the substitution

$\theta$ is applied on $e$ also and the return value of $V[e\theta]$ is tested (see equation 2.1). If the query consisted of a composed formula this return value may be modified as described before and as defined within the procedure.
Comprising, the answer of $V$ is acquired by decomposing the query, reducing the knowledge encoded in the query to the knowledge of ground atomic formulas and deciding if this knowledge is contained in the actual knowledge base.
For example, if we assume $V[p] = 0$ and $V[q] = 1$, $KB = \{p, q\}$, $V[p \vee q]$ would return 1. To show incompleteness we can give a simple example. Suppose $V[p]$ to be $\frac{1}{2}$ then $V[p \vee \neg p] = \frac{1}{2}$ and not the wanted answer 1.

Levesque proved logical completeness if the query is in a certain normal form named $\mathcal{NF}$ [40]. In general $\mathcal{NF}$ is an extension to the *Blake Canonical Form* ($\mathcal{BCF}$)[7]. In addition to the properties that hold for formulas in $\mathcal{BCF}$, formulas that are in $\mathcal{NF}$ are also closed under negation and may contain arbitrary combinations of $\vee$ and $\wedge$.
It is said in [40] that every query can be equivalently transformed into this special normal form. However, this property was only proven in the propositional case, but not in the first-order case.

Anyway, transforming the query into this normal form is intractable in general [40]. Hence, if the reasoning procedure efficiently returns logical correct answers, the query has to be transformed in an computationally intractable way. In [40] there are applications like problem solvers and planners suggested that depend on very large knowledge bases and in which the transformation of the query could be done offline. Then the application could be sure of a logical sound and complete answer evaluated by the reasoning procedure.

It was also shown in [36] that $V$ is complete for arbitrary queries when only tautological entailment [2] and not the classical logical entailment is considered. In general the propositional tautological entailment allows besides the standard two-valued assignment, that formulas can additionally be assigned neither true nor false or both values. Consequently, the connection between the falsity and truth of a sentence is not existent anymore [36]. For example, $p \wedge (p \supset q)$ does not tautologically entail $q$, because $p$ as well as $\neg p$ can be supported by some setup[1].
Therefore tautological entailment is a much weaker notion than implication as known in standard logic due to the four-valued setups which include the set of two-valued assignments [11]. This weaker kind of entailment enables $V$ to be sound as well as complete for tautological entailment and arbitrary

[1] A setup was originally defined by using the four truth values true, false, neither or both [20].

queries [36].

While we will not discuss the implementation of $V$ in our work here we would like to mention that it could be shown in [46] that $V$ itself can be implemented efficiently in some cases. Here the word "efficient" means that queries can be evaluated in a comparable complexity as they can be evaluated in classical databases. Since it is assumed in [46] that a given query is in $\mathcal{NF}$ the reasoning accomplished is also logical correct.
The general idea was to reduce $V$ to database query evaluation and to gain this result a bottom-up database query evaluation algorithm was adapted. The underlying query evaluation algorithm is mainly based on work introduced and extended by [3, 29].
This is a very important result since we would like to implement the successor of $V$ that introduces additional expensive features to handle disjunctive information and if $V$ would be intractable then it would be of no question that $X$ would be intractable too.

To give a more practical insight concerning $V$ suppose you would implement $V$ in PROLOG as function with two parameters, v(Query,ReturnValue) namely. Then you could depict the existential for example as:

---

```
v(exists(Variable,Query), ReturnValue) :-
    isSingleFormula(Query),
    member(Constant, DomainConstants),
    substitute(Variable, Constant, Query, GroundFormula),
    inKnowledgeBase(GroundFormula, ReturnValue).
```

---

This is of course a very simple and shortened version of the implementation, but it is just to emphasize the fact that there are only few features in $V$ that would prevent an efficient implementation. The amount of terms and especially constants is one of the antagonists to efficiency.
The decomposition of formulas accomplished by the recursive definition of $V$ can be intuitively transferred into PROLOG. This is of course true for $X$, too.
Although even with $V$ as it will be with $X$ the question arises how to store and how to manage more than $10^5$ *proper* terms. But before we go into deeper detail now we leave this topic for a later chapter to come.

In conclusion, $V$ is a decidable and logical sound reasoning procedure that infers if a query is known to be true, known to be false or unknown by evaluating it while supporting knowledge bases that can contain both complete and incomplete knowledge.

### 2.3.2 The X-Procedure

**$proper^+$ Knowledge Bases**

While [40] presented a deductive reasoning procedure that operates on function-free ground literals in [36] an extension is drawn that handles disjunctive information.
This extension was motivated by the fact that incomplete knowledge about some individuals has various applications. While *proper* knowledge bases and $V$ allow reasoning in huge sets of predicates, including positive as well as negative instances and handling predicates that are left open for certain individuals it is for example *not* possible to include the following term in the knowledge base:

$$(isStudent(Mary) \vee isStudent(John))$$

But especially this kind of terms that contain incomplete knowledge about some individuals and the corresponding reasoning are of great interest as stated in the introduction.

Since $X$ can handle clauses among other things we have to extend the *proper* knowledge bases used with V to include clauses. Therefore the following definitions are introduced:

**Definition** If $c$ is a disjunction of literals whose arguments are distinct variables, $\forall(e \supset c)$ is called a $\forall - clause$.

**Definition** Then a $KB$ is called a $proper^+$ $KB$ when the $KB$ is a finite and non-empty set of $\forall - clauses$. Given a $proper^+$ $KB$, $gnd(KB)$ is defined as $\{c\theta \mid \forall(e \supset c) \in KB \text{ and } \varepsilon \models e\theta\}$.

In regard to the definition of standard interpretations some sections ago, this means that a $proper^+$-KB is a finite representation of the normally infinite set $gnd(KB)$. The set is usually infinite because in $gnd(KB)$ every formula of the KB is included with all possible substitutions of variables. Note that we have an infinite set of constants.

To give an example for a valid $proper^+$ $KB$ the following knowledge base would be one:

$$KB = \{\forall(x \neq a \supset P(x)), \forall(x = y \supset \neg P(x) \vee Q(y))\}$$

Since clauses have many applications in general, they have two predominate ones [36]. First they can be used to represent rules and secondly incomplete knowledge about individuals.

| *KB1* | *KB2* |
|---|---|
| $P(a) \vee P(b) \vee P(f)$ | $P(a) \vee Q(e) \vee Q(c)$ |
| $P(a) \vee P(e) \vee Q(f)$ | $Q(d) \vee P(b) \vee Q(a)$ |
| $P(a) \vee Q(e) \vee P(c)$ | $P(a) \vee P(e) \vee P(f)$ |
| $P(a) \vee Q(e) \vee Q(c)$ | $P(c) \vee Q(e) \vee P(a)$ |
| $Q(a) \vee P(b) \vee P(d)$ | $Q(a) \vee Q(b) \vee Q(g)$ |
| $Q(a) \vee P(b) \vee Q(c)$ | $P(a) \vee P(e) \vee Q(f)$ |
| $Q(a) \vee Q(b) \vee Q(g)$ | $Q(b) \vee Q(a) \vee P(g)$ |
| $Q(a) \vee Q(b) \vee Q(g)$ | $Q(a) \vee P(d) \vee P(b)$ |

**Table 2.1:** Two knowledge bases that contain incomplete knowledge about individuals

As discussed in [36] the first application of clauses can be fulfilled by $X$, but the second which requires solving a combinatorial puzzle is nearly given up for the sake of efficiency.
To give you a brief impression what kind of complexity occurs when incomplete knowledge about individuals is involved, consider the following example. In the table 2.1 two different knowledge bases are depicted containing the two predicates $P$ and $Q$.
If you now try to answer the query $\exists X.(P(X) \wedge Q(X))$ you will observe that is quite hard to determine that only one of the two knowledge bases supports the query.

As we will see, $X$ deals with incomplete knowledge introduced by clauses in a limited way.

### Unit Propagation

Next we present one of the main features that is included in the reasoning procedure $X$:

**Definition** If $S$ is a set of ground clauses, then $UP(S)$ is the least set which contains $S$ and if $\{l\} \cup c$ and $\overline{l}$ are in $UP(S)$, then so is $c$. In other words $UP(S)$ is simply the application of *Unit Propagation* to the set $S$.

$X$ uses $UP(S)$ to decide if a literal can be inferred or not.
While we introduced unit propagation formally, we would like to give some explanations and examples at this point, because it is important to understand how unit propagation works and when it can be applied successfully.
As said in the introduction unit propagation is an often used method in the propositional case. There exist various efficient implementations of unit

propagation for the propositional case like presented in [73], [44] and [30]. In fact the complexity of the appliance of unit propagation is linear in the propositional case [73].
The use of unit propagation ensures simple applications of *Modus Ponens* as shown in the following example that consists of literals only:

$$KB = \{p, \neg s, (\neg p \vee q), (\neg q \vee s \vee r)\}$$

After applying unit propagation (first time):

$$KB = \{p, \neg s, q, (\neg q \vee r)\}$$

After applying unit propagation (second time):

$$KB = \{p, \neg s, q, r\}$$

Since $X$ makes use of unit propagation (see equation 2.6) the answer to $X[KB, r]$ will equal 1. As you can see in the example the *unit clauses* $p$ and $\neg s$ are propagated along the disjunctions in a way that new unit clauses arise - like $q$ - what consequently causes $r$ to be known to $X$.
Note that $X$ while using unit propagation supports simple applications of *Modus Ponens* in contrast to tautological entailment, for example.
While the use of literals only makes unit propagation similar to applying unit propagation in the propositional case we must be aware of the fact that we have to consider the equality terms when using unit propagation with $proper^+$-terms in general.
At this point we again present a simple example that shows what is necessary to apply unit propagation successful:

$$KB = \{(X = a) \supset P(X), (X = b) \supset (\neg P(X) \vee Q(X))\}$$

Here we can not apply unit propagation because of the mutual exclusive equalities. And even if equalities match the equality term of the resulting term must be adapted accordingly.
This means that in contrast to the propositional case we do not only have to propagate unit clauses and delete the inverse predicate from the disjunctions, but additionally have to take care of the corresponding equality terms. This will belong to the main topics in the chapters to come.
While it is not necessary at this point to know everything about the consequences on equality terms when applying unit propagation in a $proper^+ - KB$, you should be aware of the *difference* to the propositional case.
When we investigate the interaction of equality terms and unit propagation later on (chapter 3) we will see that this particular difference calls the use of unit propagation in connection with $proper^+$-terms into question - at least for some specific types of $proper^+$-terms.

### Reasoning by Cases

As we mentioned before, $X$ deals with incomplete knowledge introduced by disjunctive information. $X$ handles such kind of knowledge by using reasoning by cases.
Hence, besides unit propagation the application of reasoning by cases is a further main feature contained in $X$. And because this feature will play a major role in our work we will introduce the principle of it here briefly and will examine it more closely when presenting our implementation.
Suppose the following knowledge base:

$$KB = \{(P(a) \vee Q(b)), (\neg P(a) \vee Q(b))\}$$

Since reasoning by cases assumes every part of a disjunction each by each as to be true and then testing a given formula or query, it follows that $\exists X.Q(X)$ would be true.
Simply because of the fact that adding $P(a)$ to the knowledge base would cause $Q(b)$ to be true by unit propagation and the effect of adding $Q(b)$ will of course support the query. Accordingly we will see that $X$ answers in the same way.

If reasoning by cases with more than one level is supported, the process is started again while the added terms of the previous levels remain.
At this point the reader should be aware of two facts. First reasoning by cases is a much more complicated method than unit propagation since it makes use of unit propagation as sub-process for example.
Second in general the complexity of reasoning by cases grows exponentially with the level that is defined, because we do not have a criteria that prevents the growth of the search space sufficiently. In the next chapter we will introduce the criteria we used in our implementation.

### The Equations

Until now we described which features the reasoning procedure uses and at this point we present $X$ itself in the following equations as defined in [36]. In the next section we determine in which equations the main features are applied.

---

$$X[KB, l] = \begin{cases} 1 & : \text{ if } l \in UP(gnd(KB)), l \text{ a literal} \\ 0 & : \text{ otherwise} \end{cases} \tag{2.6}$$

$$X[KB, t = t\prime] = \begin{cases} 1 & : \text{ if } t \text{ is identical to } t\prime \\ 0 & : \text{ otherwise} \end{cases} \tag{2.7}$$

$$X[KB, \neg(t = t\prime)] = 1 - X[KB, t = t\prime]. \tag{2.8}$$

$$X[KB, \neg\neg\alpha] = X[KB, \alpha]. \tag{2.9}$$

$$X[KB, \alpha \vee \beta] = \begin{cases} 1 & \text{there is a } \forall(e \supset c) \in KB \text{ and a } \theta \in H_k^+ \\ & \text{such that } X[KB, e\theta] = 1 \text{and for all } l \in c, \\ & X[KB \cup \{l\theta\}, \alpha] = 1 \text{ or } X[KB \cup \{l\theta\}, \beta] = 1, \\ & \text{where k is the number of free variables in c} \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

$$X[KB, \neg\alpha \vee \beta] = \min\{X[KB, \neg\alpha], X[KB, \neg\beta]\}. \tag{2.11}$$

$$X[KB, \exists x.\alpha] = \begin{cases} 1 & \text{there is a } \forall(e \supset c) \in KB \text{ and a } \theta \in H_k^+ \\ & \text{such that } X[KB, e\theta] = 1 \text{and for all } l \in c, \\ & \text{there is a } d \in H_{k+1}^+ \text{ such that } X[KB \cup \{l\theta\}, \alpha_d^x] = 1, \\ & \text{where k is the number of free variables in c} \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

$$X[KB, \neg\exists x.\alpha] = \min_{d \in H_1^+} \{X[KB, \neg\alpha_d^x]\}. \tag{2.13}$$

---

### Properties of *X*

We begin this section by presenting the case where the reasoning of both introduced procedures is equivalent. The following proof was given in [36]:

**Corollary 2** *If KB is proper then for any sentence* $\alpha$*,*
$X[KB, \alpha] = 1$ *iff* $V[KB, \alpha] = 1$

Hence, when $X$ is used on *proper* $KB$s its reasoning is equivalent to that of $V$.

In contrast to its precursor $X$ returns only 0 and 1 and is therefore not a three-valued procedure anymore. Nevertheless it is possible to gain the same answers like $V$ does because of the fact that using the query and its negation together yields the same response in total.
$V$ answers a single query $\alpha$ by returning if $\alpha$ is known to be true, known to

be false or unknown. To reach the same result with $X$ we have to use two queries, namely $\alpha$ and $\neg\alpha$. Hence, the return value 0 here means *unknown* and not known to be false anymore.

The major differences to $V$ can be found in equations 2.7, 2.10 and 2.12. In fact those are the lines that are concerned with literals, disjunctions and the existential.
The first mentioned equation uses $UP(S)$ to decide if a literal can be inferred or not.
Equations 2.10 concerning disjunctions and 2.12 concerning existential quantifiers allow reasoning by cases, but only with respect to a single clause in $S$. Note that the number of applications of reasoning by cases is limited by the structure of the query. In fact, reasoning by cases is allowed exactly *once* for each appearance of a disjunction and existential in the query.

In our implementation we allow a *user-defined* level of reasoning by cases. This difference to $X$ was inspired by a slightly different version of $X$ - namely $W$ presented in [37].
This means for example that the user can define that the query $\exists X.P(X)$ should be answered by using two levels of reasoning by cases.

In the procedure there is nothing said on how to choose the next clause for reasoning by cases - the choice is a non-deterministic one. This of course must be changed in the practical implementation since our algorithm must have some criteria at least to choose the next clause to enable an efficient implementation. It is obvious that we can not try *every* clause of the $10^5$ possible ones. The criteria used is presented in the next chapter.

While we will not talk about every detail of $X$ we would like to say some words on the role that $H_k^+$ plays.
First of all writing $\theta \in H_k^+$ means that the substitutions may only range over the constants included in $H_k^+$.
As denoted in equations 2.10 and 2.12 $X$ does not allow case splitting for any clause included in the knowledge base, but is restricted to split substitution instances over $H_k^+$ of a clause in the knowledge base only. So the choice of a clause is restricted by $H_k^+$.

Both $V$ and $X$ accomplish logical sound reasoning, but are not logical complete. Note again that $X$ only supports reasoning by cases when an existential or a disjunction is included in the query. Suppose the following terms to be contained in the knowledge base:

$$(P(a) \vee P(b))\ (\neg P(a) \vee P(b))$$

Although it is obvious that the query $P(b)$ is supported by the given knowledge base, $X$ will return "unknown", because of the reasons described before.

Another example to show incompleteness of $X$ is contained in the example presented earlier (table 2.1). $X$ is not able to handle this kind of knowledge bases, because it is simply not sufficient to apply reasoning by cases to only one clause when solving an existential or a disjunction.
Although the knowledge base $KB2$ entails the given query $\alpha$, $X$ would return the value "unknown" ($X[KB2, \alpha] = 0$). The solving of the puzzle would require noticeable more levels of reasoning by cases than one - in fact 8 are required. Note that the required number of levels is equal to the number of clauses contained in the knowledge base.
The property that $X$ is *not* able to solve this kind of combinatorial puzzles is a desired effect since the reasoning accomplished by $X$ was planned to stay tractable [36]. For instance, tautological entailment is able to solve such kind of puzzles, but at the same time it is a subject to cause intractability [36]. Note that tautological entailment is tractable in the propositional case [39], but [35, 55] showed that this result could not be transferred in the first-order case.

One other interesting fact that could be shown in [36] is that even if the query is converted to the earlier mentioned normal form $\mathcal{NF}$ the question if the query is entailed by a $proper^+$ knowledge base stays undecidable in general. As showed before this was different when only *proper* knowledge bases were considered and therefore no clauses were of concern.

Furthermore the following important property of $X$ was proven in [36]:

**Theorem 3** *$X$ is decidable.*

The proof is mainly founded on the reasons that both the knowledge base and the set $H_k^+$ are finite and that it is always possible to decide if a literal $l$ is a member of $gnd(UP(KB))$ due to the fact that no full *Resolution* is applied.

All in all it is quite obvious that the two features unit propagation and reasoning by cases are also the main difference in complexity to the reasoning procedure $V$. As described in an earlier section $V$ does not make use of such kind of rather complex methods. Details on this topic are presented in the subsequent chapters.
Consequently, it remains to be shown whether $X$ is *efficient* computable.

## 2.4 Summary

In this chapter we introduced two deductive reasoning procedures - $V$ and $X$ namely - as they were presented in [40, 36]. Before the design and the

features of each procedure were presented, we gave a brief introduction to the notation and definitions and the underlying logic.

While the importance of *Unit Propagation* and *Reasoning by Cases* was described, both are consequently of special regard in the subsequent chapter since they are crucial for the reasoning procedure itself and therefore extremely relevant for an efficient implementation of $X$.

Besides Lakemeyer and Levesque presented a new version of the reasoning procedure in [37] named $W$ which is very similar to $X$ except the fact that the depth of reasoning by cases is user-defined. In our implementation we will concentrate on $X$ mainly, but will allow the user-defined level of reasoning by cases.
There is not only a new variant of the reasoning procedure presented but there is also a new logic introduced to give a more predictable and intuitive insight on how those rather complex and recursive procedures answer. For further details please refer to [37].

In conclusion, $X$ is decidable and performs logical sound reasoning as $V$ does. But in contrast to $V$ it can also handle disjunctive information by using the main features unit propagation and reasoning by cases.

# Chapter 3

# Exploring Properties of the Decision Procedure

## 3.1 Introduction

In this chapter we will discuss two properties introduced by the decision procedure. The first section will investigate the growth of the equality terms that takes place when unit propagation is applied. Concerning this topic recall the following example knowledge base:

$$KB = \{(X \neq a \supset P(X)), (X \neq b \supset (\neg P(X) \vee Q(X))\}$$

If we apply unit propagation this results in the following $proper^+$ term:

$$(X \neq a \wedge X \neq b \supset Q(X))$$

Note the growth in the equality term.
We show that inequality causes an exponential growth of the equality terms with regard to the number of unit propagations applied. This is one of the main reasons why we will *exclude* inequality from the equality term in general when we implement $X$ later on.

Second we will show the interchangeability in the order of the generation of ground terms and the application of unit propagation. This is necessary to allow the application of unit propagation without generating all possible ground instances first as it was defined in the reasoning procedure $X$.
Additionally, we will present in this section the essential difference between unit propagation in the propositional case and the first-order case.

| Equality Term | Restriction |
|---|---|
| $Variable$ | *Not Restricted* |
| $Variable \neq Constant$ | *Semi Restricted* |
| $Variable = Constant$ | *Fully Restricted* |

**Table 3.1:** The partitioning of equality-terms relating to the assigned value of a variable

## 3.2 The Growth of Equality-Terms

### 3.2.1 When growth takes place

In this section we will determine the growth of the equality terms '$e$', when we apply unit propagation. First, we recall why there is any growth of $e$ in a proper term. Since we discussed in the last chapter how unit propagation works, it is quite easy to realize that there are cases when applying unit resolution causes $e$ to grow, because the equalities of the resulting proper term have to be updated.
Note that we can apply unit propagation only if there exists a proper term that contains more then one predicate and another proper term that contains exactly one predicate ('*Unit Clause* or *Unit Term*').
Additionally a matching unit term must contain the negated version of a predicate which is one of the predicates from the proper term with multiple predicates. Furthermore the equalities of both proper terms must match, that is equalities may not be mutually exclusive.

Our first step to determine when growth takes place and when the number of equality terms stays constant is that we divide equality terms into three categories. Those categories are based on restrictions relating to variables.

As shown in table 3.1 a variable can be assigned three different kinds of values: all possible values (domain constants), all possible values except one or exactly one value. For each of those assignments exist different kinds of properties when unit propagation is applied and therefore we will proceed through them each by each.
The three classes can also be interpreted in a hierarchical way since every class describes an explicit level of restriction. Then the highest level of restriction would be assigning a variable to a constant and the lowest level would be no restriction at all.
We will call the $proper^+$ term which contains more than one predicate the *disjunctive term* and the other proper term as before *unit term*. Note that at this point we deal only with proper terms where unit propagation can be

| | | $e_1$ | | |
|---|---|---|---|---|
| | | *Not restricted* | *Semi restricted* | *Fully restricted* |
| $e_2$ | *Not restricted* | $e_1 = e_2$ | $e_1$ | $e_1$ |
| | *Semi restricted* | $e_2$ | $e_1 \wedge e_2$ | $e_1$ |
| | *Fully restricted* | $e_2$ | $e_2$ | $e_1 = e_2$ |

**Table 3.2:** The resulting equality term depending on the classes $e_1$ and $e_2$ belong to

successfully applied to.
We assume for now that an equality term $e$ contains only *one* variable and *one* corresponding assignment. Furthermore, $e_1$ corresponds to the equality term of the disjunctive term and $e_2$ to the unit term.

Concerning the equalities we observe the following when unit propagation is applied:

- If $e_1$ restricts a variable not at all, it will be overwritten by $e_2$ in any case, since $e_2$ restricts a variable on a higher level of restriction or at least at the same level. It is obvious that no growth of equalities will take place if we only replace $e_1$ by $e_2$.

- If $e_1$ restricts a variable to have any value except one, it will be replaced by $e_2$ only if $e_2$ restricts a variable to a constant.

- If $e_2$ semi-restricts a variable, the equality term of the resulting disjunctive term will be of the following form $e_1 \wedge e_2$, if $e_2 \neq e_1$. We note a growth in this case.

- If $e_2$ does not restrict the variable in any way, $e_1$ just remains the equality term of the resulting proper term.

- If $e_1$ restricts a variable to be exactly one constant, then $e_2 = e_1$, hence $e_1$ resides and so there is again no growth at all.

All these observations between the equality term of the disjunctive term and the unit term are depicted in the table 3.2.

So there is only one case where growth takes place - namely if $e_1$ and $e_2$ both semi-restrict a variable. For example, $X \neq a \supset P(X) \vee Q(X)$ and $X \neq b \supset \neg P(X)$ will resolve to $X \neq a \wedge X \neq b \supset Q(X)$. All other combinations will resolve in a replacement of $e_1$ by $e_2$ or $e_1$ simply remains.

### 3.2.2 The interaction between Growth and Unit Propagation

So far we discovered that in this kind of constellation there is only one type of growth possible.
But the considered equality terms contained only one variable each. This simple configuration allowed us to determine the worst case of growth on the lowest level. We will use that result while now equality terms can now include any number of variables.

In our implementation that we will present in the subsequent chapter we assume that all equality terms are in *Disjunctive Normal Form* (DNF). We decided in favor of the DNF, because we depend on a fast comparison between equality terms of two $proper^+$ terms when implementing $X$. Since DNF allows us to split the equality term into the conjunctions of single assignments of variables and test the equalities one by one we can easily decide if equality terms are compatible or not.
In contrast, if we would not use DNF we would trade off time for space. At this point the reader will realize that the growth caused by unit propagation presented in the last section and the use of the DNF will cause an exponential growth. We will return to the topic concerning the choice of the normal form later on.
In this and in the subsequent section we will make use of the following definitions and properties.

**Definition** Two equality terms $e_1$ and $e_2$ *match* if there is at least one substitution $\theta$ for which the following holds:

$$\epsilon \models e_1\theta \text{ and } \epsilon \models e_2\theta$$

In other words two equalities match if they are not mutually exclusive.

Now we present an equivalent representation of $proper^+$ KBs $\{(e \supset c)\}$ when $e$ in DNF and $e$ contains no inequalities. Therefore, we need the following notations. Given $e$, let $\theta_e$ be a substitution which maps only variables occurring in $e$ to constants. In a similar way $\theta|_e$ restricts $\theta$ to the variables contained in $e$.

**Definition** Let KB be a $proper^+$ KB $\{(e \supset c)\}$ and $e$ in DNF and contain no inequalities. Then $KB_{e-free} = \{\ \forall(c')|$ there exists a $\forall(e_1 \vee ... \vee e_n \supset c) \in KB$ so that there is a $e_i$ and $\theta_{e_i}$ and $\epsilon \models e_i\theta_{e_i}$ and $c' = c\theta_{e_i}\}$

Note that every $proper^+$ knowledge base can be modified to a *e-free* knowledge base in this way, if equality terms are in DNF and contain *no* inequalities.

**Lemma 4** *For all standard interpretations I the following equation holds:*

$$I \models KB \text{ iff } I \models KB_{e-free}$$

**Proof** "⇒": Let $I \models KB$ and let $\forall(c') \in KB_{e-free}$. Hence, there exists $\forall(e_1 \vee ... \vee e_n \supset c) \in KB$, so that there exists an $e_i\theta_{e_i}$ and $\epsilon \models e_i\theta_{e_i}$ and $c' = c\theta_{e_i}$. Now $I \models \forall(e_1 \vee ... \vee e_n \supset c)$ and by the assumption follows that $I \models e_i\theta$ for some $i$. Then $I \models (e_1 \vee ... \vee e_n)\theta_{e_i}$. Hence, $I \models \forall c\theta_{e_i}$, i.e. $I \models \forall c'$. "⇐": Let $I \models KB_{e-free}$ and let $\forall(e_1 \vee ... \vee e_n \supset c) \in KB$. We show that $I \models \forall(e_1 \vee ... \vee e_n \supset c)$. Suppose, $I \models e_i\theta$ for some substitution $\theta$. It suffies to show $I \models c\theta$. Then there exists a $\theta_{e_i} = \theta|_{e_i}$, so that $I \models e_i\theta_{e_i}$ i.e. $\epsilon \models e_i\theta_{e_i}$. By definition of $KB_{e-free}$ there is a $c' \in KB_{e-free}$ with $c' = c\theta_{e_i}$. Hence, since $I \models \forall c'$ holds by the assumption we have $I \models \forall c\theta_{e_i}$ and therefore, $I \models c\theta$. ∎

From Lemma 4 the next theorem follows immediately.

**Theorem 5** *For every proper$^+$ KB $\{\forall(e \supset c)\}$ with e in DNF and e contains no inequalities and the corresponding equality free $KB_{e-free}$ $\{\forall(c)\}$ the following holds:*

$$KB \models \alpha \text{ iff } KB_{e-free} \models \alpha$$

Note that the original representation is more compact, but the representation used here is equivalent and causes only a minor growth of the KB. Suppose that the maximal number of disjunctive equality terms is $k$. Then the growth lies in $O(m * k)$ while $m$ denotes the size of the original KB.

Before we continue the observations of the special case mentioned at the beginning of this section we first of all present a result given by [47].

**Theorem 6** *(Liu, [47])*
*Let KB be an e-free knowledge base $\{\forall(c)\}$ when $|KB| = n$ and $\mathcal{C}_{\mathcal{KB}}$ denotes the set of constants contained in KB while $|\mathcal{C}_{\mathcal{KB}}| \leq n$. In addition, k denotes the maximal number of variables in one clause c.*
*Then the closure under unit propagation applied to the entire KB results in a knowledge base $KB'$ while $|KB'| \leq n^{k+1}$.*

**Proof-Sketch** Let $\theta_{\mathcal{C}_{\mathcal{KB}}}$ range over substitutions of all variables by constants $c_i$ while $c_i \in \mathcal{C}_{\mathcal{KB}}$ and $gnd_{\mathcal{C}_{\mathcal{KB}}}(KB) = \{c\theta_{\mathcal{C}_{\mathcal{KB}}} \mid \forall(c) \in KB\}$.
Then $|UP(KB| \leq |gnd_{\mathcal{C}_{\mathcal{KB}}}(UP(KB))|$. By an extension of Theorem 12 it would be possible to show $|gnd_{\mathcal{C}_{\mathcal{KB}}}(UP(KB))| = |UP(gnd_{\mathcal{C}_{\mathcal{KB}}}(KB))|$. Since unit propagation applied to ground instances corresponds to unit

propagation applied in the propositional case the following equation holds: $|UP(gnd_{\mathcal{C}_{\mathcal{KB}}}(KB))| \leq |gnd_{\mathcal{C}_{\mathcal{KB}}}(KB)|$.
While we have maximally $k$ variables in each of the $n$ clauses contained in KB and have maximally $n$ constants the maximal number of possible ground instances equals $n * n^k$. Hence, $|gnd_{\mathcal{C}_{\mathcal{KB}}}(KB)| \leq n^{k+1}$. Consequently, $|UP(KB| \leq n^{k+1}$. ∎

At this point we return to the case where inequalities are supported and examine the growth caused by the application of unit propagation.

While again $e_1$ and $e_2$ represent the equalities of the disjunctive term and the unit term respectively, $|e_j|$ will denote the total number of single equalities in $e_j$. In addition, $e_{1i}$ will indicate the i-th clause of an equality term. Furthermore, $|e_{1i}|$ holds the number of conjunctive equality terms that are contained in the i-th clause while $n$ $(m)$ stands for the total number of disjunctive equality terms in $e_1$ $(e_2)$.
Since we do not restrict an equality term to hold exactly one variable anymore, the growth will now additionally increase by all possible combinations between disjunctive equality terms in $e_1$ and $e_2$.
For example, if we have the following equality terms:

$$e_1 : (X \neq C_1 \wedge Y \neq C_2) \vee (X \neq C_2 \wedge Y \neq C_1)$$
$$e_2 : (X \neq C_3 \wedge Y \neq C_4) \vee (X \neq C_4 \wedge Y \neq C_3)$$

If we would apply unit propagation now we had to adept $e_1$ in the way we determined before. Since this is not a matter of replacing or keeping a equality term, the single equality terms from $e_1$ and $e_2$ add up, but not in a linear way.
The growth is not linear because disjunctions cause as much combinations of $e_{1i}$ and $e_{2j}$ as there are disjunctions in $e_1$ and $e_2$. This is in general the well known drawback of a conversion of a formula to DNF. In our case we have to convert two formulas that only consist of disjunctions connected by a conjunction to a formula in DNF. This causes the mentioned growth.

In our example one part of the resulting equality term which only regards the first part of $e_1$, namely $e_{11}$, would be:

$$(X \neq C_1 \wedge Y \neq C_2 \wedge X \neq C_3 \wedge Y \neq C_4) \vee (X \neq C_1 \wedge Y \neq C_2 \wedge X \neq C_4 \wedge Y \neq C_3).$$

We count 8 single equality terms for this part of adaption and the final resulting equality term would hold 16 single equality terms in total, so that the number of equality terms in this case doubles in total.
We again observed a worst case scenario here since besides using semi-restricted equalities only, we also assumed that every variable which is contained in $e_1$ is also contained in every single term in $e_2$ which is normally

not the case; recall that $e_1$ is the equality term of a proper term that holds different predicates and $e_2$ constraints variables that belong to exactly one predicate only.
When we assume that we only use semi-restricted equalities and every clause from $e_1$ has to be combined with every clause from $e_2$, then we can determine the number of single equality terms in the resulting equality term in the following way.
First we write $e_1$ and $e_2$ in detail as defined:

$$e_1 : e_{11} \vee e_{12} \vee ... \vee e_{1n}$$
$$e_2 : e_{21} \vee e_{22} \vee ... \vee e_{2m}$$

So the number of single equality terms ($|e_1|$) in $e_1$ can be calculated by $\sum_{i=1}^{n} |e_{1i}|$; $|e_2|$ analogous.
Consequently the resulting equality term is of the following form:

$$e_{resulting} : ((e_{11} \wedge e_{21}) \vee (e_{11} \wedge e_{22}) \vee ... \vee (e_{11} \wedge e_{2m}) \vee (e_{12} \wedge e_{21}) \vee ... \vee (e_{12} \wedge e_{2m}) \vee ... \vee (e_{1n} \wedge e_{21}) \vee ... \vee (e_{1n} \wedge e_{2m}))$$

Since we are interested in the total number of single equality terms in $e_{resulting}$, we can determine $|e_{resulting}|$ by the following equation:

$$|e_{resulting}| = \sum_{i=1}^{n}(m * |e_{1i}| + \sum_{j=1}^{m} |e_{2j}|) \tag{3.1}$$

While we recognize $|e_1|$ and $|e_2|$ in equation 3.1 and $|e_2|$ is added up n-times the equation can be simply rewritten as:

$$|e_{resulting}| = m * |e_1| + n * |e_2| \tag{3.2}$$

Consequently the growth of equality terms is determined by $|e_{resulting}| - |e_1|$ and hence:

$$\text{Maximum Growth of Equality Terms} = (m-1) * |e_1| + n * |e_2| \tag{3.3}$$

As we can observe in equation 3.1 the growth of equality terms is the composition of the growth of semi-restricted variables and the combination of disjunctive terms in $|e_1|$ and $|e_2|$. We decreased the number of equality terms by $e_1$ in this equation, because we determined the growth and not the total size of the resulting equality term.

Notice that we assumed that there are *only* semi-restricted variables as well in $|e_1|$ as in $|e_2|$ and every $|e_{1i}|$ has its counterpart in every $|e_{2j}|$ which

is quite unusual as said before.
In that we showed that only semi-restricted variables cause any growth at all and maximised the number of combinations of single equality terms equation 3.3 marks out the maximum possible growth of equality terms.
To describe the size of the resulting equality term in a more simple way we assume that $n = m$ and $|e_1| = |e_2|$ so that we get the following equation:

$$|e_{resulting}| = n * (2 * |e_1|) \tag{3.4}$$

With the assumptions made we notice that the number of single equality terms doubles if we have no disjunctive equality terms at all ($n = 1$). If $n > 1$, $|e_1|$ is additionally multiplied by the number of disjunctions in $|e_1|$. This might give a more intuitive feel of the growth of equality terms.

As we showed before the resulting equality term will hold as much as single equality terms as denoted in 3.1.
If we assume again that $|e_1| = |e_2|$, $n = m$ and additionally $|e_{1i}| = |e_{2j}| = 1$ for all $i, j$ (consequently $|e_1| = |e_2| = 1$) and we apply unit propagation now not only once, but several times the number of single equality terms will increase dramatically. The number of unit propagations applied is denoted as $|UPs|$.

From 3.4 and since $\sum_{i=1}^{n} |e_{1i}|$=n because $|e_{1i}| = 1$ we derive the size of the resulting equality term when we apply unit propagation the first time:

$$|e_{resulting}| = n * (2 * n) = 2 * n^2 \tag{3.5}$$

At the same time the number of disjunctive equality terms will be determined by keeping in mind that $m = n$:

$$|e_{disjunctive-terms}| = n * m = n^2 \tag{3.6}$$

If we combine the following equality terms for example where $n = 3$, we will have the denoted values for the number of single equality terms and number of disjunctive equality terms:

$$e_1 : e_{11} \vee e_{12} \vee e_{13}$$
$$e_2 : e_{21} \vee e_{22} \vee e_{23}$$

$$|e_{resulting}| = 18 \quad |e_{disjunctive-terms}| = 9$$

Note again that a disjunctive equality term (e.g. $e_{11}$) consists of single equality terms.

If we now apply unit propagation a second time with a unit term that has as before $m$ (in our example is $m = 3$) disjunctive equality terms in total, we

must be aware of the actual number of disjunctive equality terms and single equality terms of the disjunctive term generated by the first application of unit propagation.
Now $n$ in equation 3.1 will be equal to $|e_{disjunctive-terms}|$ (e.g. 9) and since the number of single equality terms simply add up when $|e| = 1$ in each disjunctive equality term and by each application of unit propagation, we can rewrite 3.1 in the following way:

$$|e_{resulting_2}| = \sum_{i=1}^{|e_{disjunctive-terms}|} (m * (|e_{1i}| + |e_{2i}|) + \sum_{j=1}^{m} |e_{2j}|) \tag{3.7}$$

Equation 3.7 denotes the number of single equality terms after applying unit propagation twice. We add $|e_{1i}|$ and $|e_{2i}|$ because now each of the disjunctive equality terms of the disjunctive term consists of the addition of the old number of single equality terms and the single equality terms of the first used unit term.

Taking into account the assumptions made above and since we continue applying unit propagation this will result in the recursive definition of $|e_{disjunctive-terms}|$ while $|e_{disjunctive-terms_1}| = n * m$:

$$|e_{disjunctive-terms_{|UPs|}}| = m * |e_{disjunctive-terms_{|UPs|-1}}| \tag{3.8}$$

While we assumed that $n = m$ we have:

$$|e_{disjunctive-terms_{|UPs|}}| = n^{|UPs|+1} \tag{3.9}$$

In fact this equation already states the exponentially growth of equality terms with regard to the number of applications of unit propagation very clearly. Note that we always use $|e_{disjunctive-terms_{|UPs|-1}}|$ when determining $|e_{resulting_{|UPs|}}|$.
Furthermore, we can rewrite the increase in size of a single equality term in a disjunctive term in the following way since $|e_{1i}| = |e_{2j}| = 1$:

$$|e_{growth_{|UPs|}}| = \sum_{i=1}^{|UPs|} 1 = |UPs| \tag{3.10}$$

Consequently, we can rewrite while we now also substitute $m$ by $n$ since $m = n$ and $|e_{1i}| = |e_{2j}|$ by 1:

$$|e_{resulting_{|UPs|}}| = \sum_{i=1}^{n^{|UPs|}} n * (|UPs| + 1) \tag{3.11}$$

If we now apply unit propagation $(n-1)$-times this will result in the following equation:

$$|e_{resulting_{n-1}}| = \sum_{i=1}^{n^{n-1}} n^2 \tag{3.12}$$

**Theorem 7** *Let $e_{max}$ denote the equality term contained in the KB so that $|e_{max}| \geq |e_i|$ for every $e_i \in KB$.*
*Then there exist proper$^+$ KBs $\{\forall(e \supset c)\}$ with e in DNF such that the closure under unit propagation results in a KB so that $e_{max}$ grows exponential in the size of $e_{max}$.*

**Proof** Equation 3.12 determines the number of equalities contained in the resulting equality term after unit propagation is applied $n-1$-times while $n$ is the length of the equality term of the original disjunctive term. We now construct a general example knowledge base which meets the assumptions underlying the equation.
Suppose a KB that contains a disjunctive term that has an equality term $e_{max}$ with $|e_{max}| = n$ while the disjunctive term is of the following form:

$$(X \neq c_1 \vee X \neq c_2 \vee ... \vee X \neq c_n) \supset P_1(X) \vee ... \vee P_n(X)$$

Note that $c_i \neq c_j$ if $i \neq j$. Hence, we need at least $|C_{max}| = n$ constants in the disjunctive term.
Additionally, the knowledge base contains $n-1$ unit terms while the complement of every predicate is contained in the disjunctive term and a predicate is not contained twice in the $n-1$ unit terms. At least the following $n-1$ unit terms must be contained in the knowledge base:

$$(X \neq a_1 \vee X \neq a_2 \vee ... \vee X \neq a_n) \supset \neg P_1(X)$$
$$...$$
$$(X \neq u_1 \vee X \neq u_2 \vee ... \vee X \neq u_n) \supset \neg P_{n-1}(X)$$

Note that the equalities correspond to the same variable as they do in the disjunctive term and we have $n$ equalities each. However, none of the equality terms contains a constant from the set $C_{max}$ nor a constant that is used in another equality term of the unit terms of concern. This prevents that identical equalities are contained in the equality terms of the disjunctive term and the corresponding unit terms.
Thereby we fulfill the assumption underlying Equation 3.12, especially $|e_{max}| = |e_{unit_1}| = ... = |e_{unit_{n-1}}| = n$. In addition, unit propagation can be applied successfully since we only deal with inequalities and there exist $n-1$ matching predicates.
Therefore, we can apply unit propagation $n-1$-times. Consequently, as depicted in Equation 3.12 we notice an exponential growth in the size of $|e_{max}|$

concerning the equality term $e_{max}$. Hence, the closure under unit propagation can cause an equality term to grow exponentially in its size. ∎

It is quite obvious that this can not be handled efficiently even if the number of the equalities, predicates and applications of unit propagation are rather small. As said before this is one of the reasons why we excluded inequalities from equality terms. More reasons are discussed in the following chapter.

Now we return to the topic concerning the use of DNF. The reader might think at this point that the presented growth of equality terms is mainly caused by the conversion to DNF and this is in fact true.
If we would not convert the equality terms to DNF we conjecture that the growth would not be exponential anymore since we would not have to take care of the huge number of possible combinations of the single equalities. In fact, after a successful application of unit propagation we would simply add the corresponding equalities of the unit term to the equalities of the original disjunctive term and no further processing (e.g., normal form conversion) would take place.
But as mentioned before we haven chosen in favor of DNF, because it allows us to compare equalities in a efficient way. And as also said before if we do not store equality terms in DNF a rather complex satisfiability test is required. For example, we would then have to take care of the above mentioned possible combinations and the connections between the single variables when testing a single equality.

We need the following notations for the next observations concerning the satisfiability test of equality terms. In this context, $\alpha$ denotes an arbitrary propositional formula and $p$ an atomic formula. Then we construct a formula $e_\alpha$ so that the following holds: $\alpha$ is satisfiable iff $e_\alpha$ is satisfiable. In $e_\alpha$ every $p \in \alpha$ is replaced by $X_p = c_p$ respectively where $X$ is a variable and $c_i$ a constant. In addition, $\mathcal{V} \models e$ iff $\varepsilon \models e^{X_1 \ldots X_n}_{v(X_1)\ldots v(X_n)}$ for all $X_i$ in $e$.

**Lemma 8** *Let $I$ be a truth assignment of the the atoms $p \in \alpha$. Additionally, $\mathcal{V}_\mathcal{I}$ is a variable mapping so that $\mathcal{V}_\mathcal{I}(X_p) = c_p$ if $I(p) = true$ and $\mathcal{V}_\mathcal{I}(X_p) = c^*$ if $I(p) = false$ when $c^* \neq c_p$.*

$$\textit{Then } I \models \alpha \textit{ iff } \mathcal{V}_\mathcal{I} \models e_\alpha$$

**Proof** The proof is by induction on the structure of $\alpha$. In the base case we have $I \models p$ iff $\mathcal{V}_\mathcal{I}(X_p) = c_p$ iff $\mathcal{V}_\mathcal{I} \models X_p = c_p$.
"$(\alpha_1 \vee \alpha_2)$": $I \models (\alpha_1 \vee \alpha_2)$ iff $I \models \alpha_1$ or $I \models \alpha_2$ iff $\mathcal{V}_\mathcal{I} \models e_{\alpha_1}$ or $\mathcal{V}_\mathcal{I} \models e_{\alpha_2}$ iff $\mathcal{V}_\mathcal{I} \models (e_{\alpha_1} \vee e_{\alpha_2})$.

"$(\alpha_1 \wedge \alpha_2)$": $I \models (\alpha_1 \wedge \alpha_2)$ iff $I \models \alpha_1$ and $I \models \alpha_2$ iff $\mathcal{V}_{\mathcal{I}} \models e_{\alpha_1}$ and $\mathcal{V}_{\mathcal{I}} \models e_{\alpha_2}$ iff $\mathcal{V}_{\mathcal{I}} \models (e_{\alpha_1} \wedge e_{\alpha_2})$.
"$(\neg\alpha)$": $I \models \neg\alpha$ iff $I \not\models \alpha$ iff $\mathcal{V}_{\mathcal{I}} \not\models e_\alpha$ iff $\mathcal{V}_{\mathcal{I}} \models \neg e_\alpha$. Note that it is sufficient to choose one constant $c^*$ so that $c^* \neq c_p$ for all $p \in \alpha$. ∎

**Lemma 9** *Let $\mathcal{V}$ be a variable mapping for $e_\alpha$. Additionally, $I_v$ is a truth assignment for every $p \in \alpha$ so that $I_v(p) = true$ if $\mathcal{V}(X_p) = c_p$ and $I_v(p) = false$ if $\mathcal{V}(X_p) \neq c_p$.*

$$\textit{Then } \mathcal{V} \models e_\alpha \textit{ iff } I_v \models \alpha$$

**Proof** Similar to the proof of Lemma 8. ∎

**Theorem 10** *The satisfiability problem for the formulas e is NP-hard.*

**Proof** Let $\alpha$ be a propositional formula, $e_\alpha$ as in Lemma 8 and Lemma 9. It is obvious that we can convert $\alpha$ in linear time to $e_\alpha$. Then we show that $\alpha$ is satisfiable iff $e_\alpha$ is satisfiable. "$\Rightarrow$": Let $I \models \alpha$ then $\mathcal{V}_{\mathcal{I}} \models e_\alpha$ by Lemma 8. "$\Leftarrow$": Let $\mathcal{V} \models e_\alpha$ then $I_v \models \alpha$ by Lemma 9. ∎

If for example, SAT instances would be in DNF, then it would be rather simple to determine if a formula is satisfiable or not. But the conversion to DNF itself would be very complex. In fact, if we would give up the conversion to DNF we would shift the complexity from space to time, but would stay exponential in both cases. Note that the conversion to DNF applies to each $proper^+$ term separately which seems practical assuming that $proper^+$ terms are very small compared to the size of the entire knowledge base.
Since inequalities are seldom in general and not used in our implementation at all we think that it was the right choice to use DNF since it allows us the very important feature to compare equalities fast. But also note at this point that the use of no inequalities would also cause the satisfiability test to be rather simple.

In this section we showed that there are three types of equality terms on which unit propagation has different effects. In particular we could show that the combination of two inequalities causes a growth in the equality terms. Furthermore, we presented the equation 3.9 that described the coherence between the size of an equality term and the number of applications of unit propagation very clearly. In fact, we could show that the size of the resulting equality terms grows exponentially in its size with regard to the number of unit propagation applied.

Additionally, we discussed the use of DNF since it is the main factor that causes exponential growth in space. Due to the fact that we can only shift this complexity from space to time, we think that the use of DNF is the right choice with regard to the requirements introduced by $X$.

Finally, this section contains one of the main reasons that causes inequalities to be excluded from the equality term in general.

## 3.3 The Interchangeability in the order of the generation of Ground Terms and the application of Unit Propagation

In this section we show the interchangeability in the order of the generation of ground terms and the application of unit propagation.
To introduce this topic we will first of all present an example to clarify an essential difference between unit propagation applied in the propositional case and the first-order case.

### 3.3.1 Unit Propagation in the Propositional Case and the First-Order Case

As we showed in the last chapter the application of unit propagation is identical in both cases when only the predicates are of concern, but there is a crucial difference concerning the result of a successful applied unit propagation.

To clarify this we can choose for instance the following set of literals:

$$\{l, (\neg l \vee m)\}$$

After applying unit propagation the set will contain the two literals $l$ and $m$ only. This is due to the fact that we can delete the clause $(\neg l \vee m)$ after applying unit propagation since this clause represents only *one* ground instance. Consequently, the set is *decreased* in its size.

If we now turn to a knowledge base that contains the following $proper^+$ terms, we will see that we are *not* allowed to delete any $proper^+$ terms.

$$\begin{gathered}((X \neq c) \supset P(X) \vee Q(X)) \\ (X = a \supset \neg P(X)) \\ (X = b \supset \neg Q(x))\end{gathered}$$

Instead of generating all ground instances, we first apply unit propagation which has the following effect on the example knowledge base:

$$(X = a \supset Q(X))$$
$$(X = a \supset \neg P(X))$$
$$(X = b \supset \neg Q(x))$$

And this result is achieved, because unit propagation updates the equality term of the disjunctive term with the equality term of the unit term and the original disjunctive term is *deleted*. In consequence no further applications of unit propagation are possible.

On the other hand we can apply unit propagation twice on the same knowledge base if we first generate all possible ground instances. In this case one term represents exactly one ground instance and not a set of ground instances anymore. Note the similarity to the propositional case which would allow us to delete single literals and not to keep the original disjunctive term.

$$(P(a) \vee Q(a))$$
$$(P(b) \vee Q(b))$$
$$(P(d) \vee Q(d))$$
$$\ldots$$
$$(\neg P(a))$$
$$(\neg Q(b))$$

In this set of ground instances we can apply unit propagation twice and gain the following set of ground instances:

$$(P(a) \vee Q(a))$$
$$(P(b) \vee Q(b))$$
$$(P(d) \vee Q(d))$$
$$\ldots$$
$$(\neg P(a))$$
$$(\neg Q(b))$$
$$(P(b))$$
$$(Q(a))$$

Note the crucial difference. The literal $P(b)$ is contained in this last set, but not in the set that was generated when we first applied unit propagation. Hence, if we delete $proper^+$ terms after applying unit propagation the suggested interchangeability does not hold.

At the same time this depicts the essential difference between unit propagation applied in the propositional case and the first-order case. We are *not* allowed to delete any $proper^+$ terms of the knowledge base except when they are redundant[1]. Consequently, the set of terms will not decrease as in the

[1] $Proper^+$ terms are called redundant if they represent the identical set of ground instances.

propositional case after applying unit propagation, but will *increase*.
And this is a major difference to the propositional case. This implies that a knowledge base *grows* in a different way than examined before when we apply unit propagation. We will return to this topic in the subsequent chapter.

Additionally, note that this is *not* caused by the use of inequalities. The same result is for example gained when we replace the first term of the original knowledge base by the term $((X = a \vee X = b) \supset P(X) \vee Q(X))$.
In this case there would not be an infinite number of possible ground instances, but exactly two. But again this set of ground instances would allow us to apply unit propagation twice and as before one additional literal would be generated in comparison to the approach when applying unit propagation first.

### 3.3.2 The Coherence between Unit Propagation and the Generation of Ground Terms

As said before we want to show in this section that the interchangeability in the order of the generation of ground terms and the application of unit propagation holds.
At this point we assume that every equality term in the given knowledge base fully restricts a single variable to a single constant and that the equality terms are in DNF. This proceeding allows us to introduce the topic in a simplified way.
Since all equality terms are in DNF we can split the equalities and can create a $proper^+$ term for each of the equality terms.
As an example we observe the following $proper^+$ term:

$$(X = a \vee X = b \supset P(X) \vee Q(X))$$

We can rewrite this term due to the fact that equalities are in DNF by creating two distinct terms:

$$(X = a \supset P(X) \vee Q(X))$$
$$(X = b \supset P(X) \vee Q(X))$$

If we rewrite every term in the entire knowledge base a single $proper^+$ term represents exactly one ground instance and consequently $|gnd(KB)| = |KB|$.
And since $gnd(KB) = \{c\theta | \forall(e \supset c) \in \text{ KB and } \epsilon \models e\theta\}$ and there exists only one possible substitution $\theta$, because there is only one fully restricted equality for each variable, the suggested interchangeability holds since every $proper^+$ term represents exactly one ground term in $gnd(KB)$.

Now we assume that the variables can also be restricted in any possible way and the equality terms are not necessarily in DNF. For instance, this implies that one $proper^+$ term can represent as many ground instances as there are constants.
To describe what effect unit propagation has when it is applied on this kind of $proper^+$ terms, we first introduce the condition that has to be fulfilled from the equality term to apply unit propagation successfully. Note that also the predicates must match, but this property will not change when we generate the ground terms first or afterwards.

Since we do not delete any $proper^+$ terms after applying unit propagation we can apply unit propagation on a disjunctive term that contains variables that are not restricted or semi restricted with regard to one predicate as often as there are unit terms with a matching equality term. The fact that $proper^+$ terms are not deleted ensures that all possible ground terms can be created later on.
Note that only variables are of concern that belong to the predicate that is affected by the application of unit propagation.
Hence, there will be as many new $proper^+$ terms as there are matching unit terms since every applied unit propagation will cause the equality term of the resulting disjunctive term to be adapted. Note that it is possible that redundant terms are created.
Consequently, a *single* $proper^+$ term will support as many successful applications of unit propagation as there are matching unit terms with regard to the corresponding predicate.

The same number of applications of unit propagation will be applied when the ground terms are generated first, because the generation of ground terms will not affect the number of matching equalities. This property also holds for the subsequent applications of unit propagation which are caused by chaining in reasoning.
Note that when generating ground terms the variables in the $proper^+$ terms are simply replaced by all appropriate substitutions.
Before we show the interchangeability in the order of the generation of ground terms and the application of unit propagation we first of all define unit propagation when it is applied to a $proper^+KB$.

**Definition** Let KB be a $proper^+$ knowledge base $\{\forall(e \supset c)\}$ while $c = (l_1 \vee l_2 \vee ... \vee l_n)$. Then UP(KB) is the least set which contains KB and if $\forall(e_1 \supset \{l\} \cup c) \in UP(KB)$ and $\forall(e_2 \supset \left\{\overline{l}\right\}) \in UP(KB)$ with $l\theta = l'\theta$ where $\theta$ is an

most general unifier (MGU) [2] and $(e_1 \wedge e_2)\theta$ is satisfiable[3], then $\forall((e_1 \wedge e_2)\theta \supset c\theta) \in UP(KB)$.

**Lemma 11** *For every proper$^+$ knowledge base $\{\forall(e \supset c)\}$ and clause $c$ the following holds:*

$$c \in UP(gnd(KB)) \text{ iff } c \in gnd(UP(KB))$$

**Proof** "$\Rightarrow$": Let $c \in UP(gnd(KB))$. We show this by induction on the number $n$ of applied unit propagations ($UPs$) used to generate $c$. Suppose $n = 0$. Then there exists $\forall(e \supset c') \in KB$ and $c \in gnd(\forall(e \supset c'))$. Then $c \in gnd(UP(KB))$. Suppose, the Lemma holds for every clause $c$ that is obtained by application of unit propagations so that the number of UPs is $< n$. Then there exits $c_1 = \{l_1\} \cup c$ and $c_2 = \{\neg l_1\}$. $c_1$ and $c_2$ are obtained by using at most $n - 1$ UPs each.
Hence, by induction, $c_1 \in gnd(UP(KB))$ and $c_2 \in gnd(UP(KB))$. Consequently, $\forall(e_1 \supset c_1')$ and $\forall(e_2 \supset c_2')$ are contained in $UP(KB))$ and $c_1 \in gnd(\forall(e_1 \supset c_1)$ and $\forall(e_1 \supset c_1')$. Then there exists $c_1' = \{l_1'\} \cup c'$ and $c_2 = \{\neg l_1''\}$.
Thus there exists an MGU $\theta$ so that $l_1'\theta = l_1''\theta$ since $l_1$ is a ground instance of $l_1'$ and $l_1''$ and $(e_1 \wedge e_2)\theta$ is satisfiable since both $e_1$ and $e_2$ are satisfiable for this instantiation, with $\forall(e_1 \wedge e_2)\theta \supset c'\theta \in UP(KB))$ and $c \in gnd(\forall(e_1 \wedge e_2)\theta \supset c'\theta)$, i.e. $c \in gnd(UP(KB))$.
"$\Leftarrow$": Let $c \in gnd(UP(KB))$. We show this by induction on the number $n$ of applied unit propagations ($UPs$) used to generate $c$. Suppose $n = 0$. Then there exists a $\forall(e \supset c') \in KB$ so that $c \in gnd(\forall(e \supset c'))$, i.e. $c \in UP(gnd(KB))$.
Suppose, the Lemma holds for every clause $c$ that is obtained by application of unit propagations so that the number of UPs is $< n$. Let $c$ be a ground instance of a clause in $UP(KB)$ generated by $n$ applications of unit propagations. Then the two terms $c_1 = \forall(e_1 \supset \{l_1\} \cup c')$ and $c_2 = \forall(e_2 \supset \{\neg l_2\})$ are contained in $UP(KB)$, so that $l_1\theta = l_2\theta$ for some MGU $\theta$, $(e_1 \wedge e_2)\theta$ are satisfiable and $c \in gnd(\forall(e_1 \wedge e_2)\theta \supset c'\theta)$.
Hence, there are $c_1' \in gnd(c_1)$ and $c_2' \in gnd(c_2)$, so that $c_1' = \{l_1'\} \cup c$ and $c_2' = \{\neg l_1'\}$. By induction, $c_1' \in UP(gnd(KB))$ and $c_2' \in UP(gnd(KB))$ and consequently $c \in UP(gnd(KB))$. ∎

[2] In this context a MGU corresponds to simply renaming of variables. For instance, assume the following two terms: $\forall X.Y.((X = a \wedge Y = b) \vee (X = a \wedge Y = a) \supset P(X,Y) \vee Q(X,Y))$ and $\forall Z.(\neg P(Z,Z))$. Then $\{X/X, Y/X, Z/X\}$ is an MGU for $P(X,Y)$ and $\neg P(Z,Z)$. The resulting term is: $\forall X.((X = a \wedge X = b) \vee (X = a \wedge Y = a) \supset Q(X,X)$.

[3] Two equality terms are satisfiable if $\varepsilon \models \exists\theta(e_1 \wedge e_2)$.

Then the next theorem follows immediately from Lemma 11.

**Theorem 12** *For every proper$^+$ KB* $\{\forall(e \supset c)\}$ *the following equation holds:*

$$UP(gnd(KB)) = gnd(UP(KB)) \tag{3.13}$$

In consequence, we proved the interchangeability in the order of the generation of ground terms and the application of unit propagation.

# Chapter 4

# Implementation

## 4.1 On implementing *X*

In this chapter we will present an implementation of the reasoning procedure $X$. Since we are interested in an efficient approach, some features of $X$ and the underlying definitions had to be changed.

First of all we will restrict the defined $proper^+$ terms to contain *no* inequalities anymore. This proceeding is justified by various reasons. For example, inequality causes a major drawback concerning efficiency as we proved in the last chapter. At least if the equality terms are represented in DNF.
This has of course several consequences concerning our implementation, but further details are presented in the corresponding sections.

At this point we present the foundation of our implementation - namely the encoding of $proper + KBs$. It is of great importance how the data is represented, because we depend on a fast access of the data.
Because we would like to handle more than $10^5$ terms we suppose that the use of a standard database (MySQL [53]) should increase both - manageability and efficiency. Our view is supported by the fact that efficient list handling is only efficient in most of the $PROLOG$-systems as long as a list contains not more than $10^5$ elements. For example $ECL^iPS^e$ PROLOG can handle lists that contain about $40,000$ elements quite well, but efficiency decreases rapidly when there are more elements of concern [32].

The proposed structure is optimized to implement one of the key features introduced by $X$ - *Unit Propagation* namely. We would like to use standard database features for efficient handling of large data sets, but furthermore we would like to use them to implement one part of the *Unit Propagation*-algorithm as well. For example we will determine a *matching unit clause* by

an easy SQL-statement.

We act in this way, because it is a common fact that databases include features that can handle large data sets efficiently [23]. One example is the method of *indexing* which allows us very fast search.
Since $X$ is mainly based on unit propagation we are depending on a very efficient implementation. All the more we can make use of database features, the more the efficiency of our implementation of $X$ increases.

We will not achieve to implement unit propagation as an SQL-Statement only, but we will get close to that goal. The things left are done by a few *PROLOG*-lines.

Then we will present how our implementation processes a given query. This topic includes how a query that contains for example disjunctions and conjunctions is decomposed and evaluated. For each of the different operators of the logical language we will present a method to test if a query is known to be true or is unknown.

In the main the efficiency depends on the implementation of unit propagation and reasoning by cases in respect to a large set of terms. Additionally, the implementation of the quantifiers is of special importance since quantifiers are already expensive in general.

After we discussed unit propagation and the evaluation of a query, we have to deal with one more key feature of $X$ - namely reasoning by cases (RbC).
Reasoning by cases is more difficult to implement than unit propagation, because it is as said in the Chapter 2 a much more complex method that makes use of unit propagation as a sub-process for example.

We will present a criterion that is rather simple but can be efficiently applied. The criterion restricts the set of possible clauses that is going to be used by reasoning by cases. Furthermore we again try to use as much database features as possible to implement it.
When presenting the algorithm belonging to reasoning by cases this is in fact the essence of the entire implementation, since the second main feature unit propagation and the evaluation of a query are both involved. Consequently all important features are contained in this algorithm.

After we introduced the two main features of $X$ and how queries are evaluated, we will present what kind of preprocessing takes place *before* any query is processed. Within the preprocessing stage we are not bound to any time limits and that is why we can apply unit propagation to the entire knowledge base, for example.
In consequence queries that would require the method of unit propagation are answered instantly. Especially queries that require simple applications of *Modus Ponens* can be answered immediately.

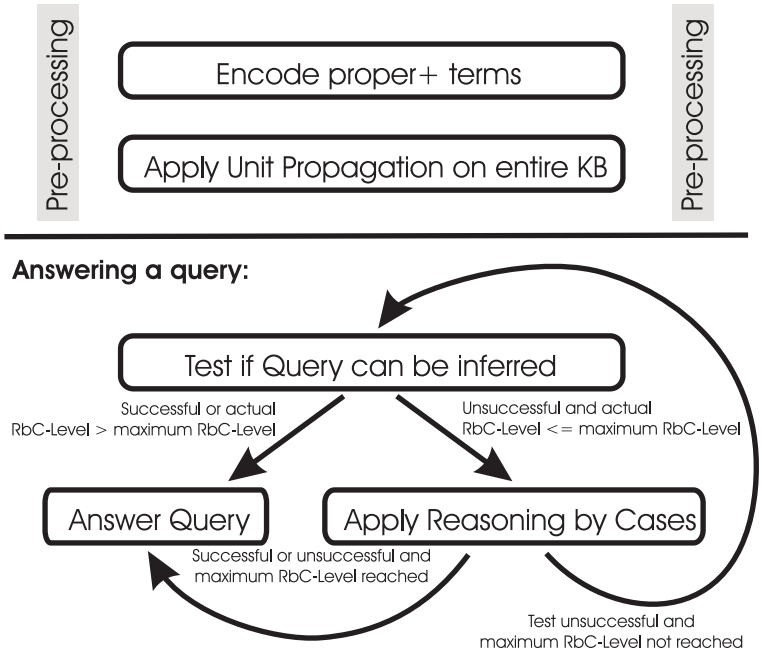


**Figure 4.1:** A brief overview on the way our implementation works like

But we will also see that preprocessing and the presented data structure allow us to handle a large number of unit terms without any major drawbacks concerning efficiency.

In the next section we will present every feature of our implementation using one detailed example that requires the evaluation of an existential quantifier, reasoning by cases twice and from there also unit propagation. This section will additionally show the advantages of the earlier presented data-structure.

To give you a first impression of the implementation made we depict the general scheme in the figure 4.1. Within the figure you see that our approach as two phases in general. First we encode $proper^+$ terms and apply unit propagation on the entire knowledge base in the preprocessing phase.
After this step our reasoning procedure is ready to answer queries. If the query can be inferred by the knowledge base we answer the query directly. If the test is unsuccessful we apply reasoning by cases if the user allows it. Otherwise the answer to the query is "unknown".
As the figure states the defined level of reasoning by cases (denoted with "RbC-Level") plays a major role since the given level decides how often reasoning by cases is applied and the entire algorithm is repeated.
As said when the definition of $X$ was introduced the structure of the query

decides if reasoning by cases is applied at all. In fact, $X$ only supports reasoning by cases when the query contains a disjunction or an existential quantifier.
However, in our approach we allow a user-defined level for reasoning by cases and so every reply to a query can use the feature of reasoning by cases if the user allows it to. This way of allowing a user-defined level for reasoning by cases corresponds to the modified version of $X$ [37].
In the figure the user-defined level of reasoning by cases corresponds to the maximally allowed level of reasoning by cases.

## 4.2 The use of Inequality

While $proper^+$ terms allow inequalities as part of the equality term, we will *not* allow inequalities in our implementation because of the following reasons.
Inequalities cause a major drawback in complexity as we discussed in the last chapter. For example, if the *Disjunctive Normal Form* (DNF) is used to represent an equality term then the size of equality terms will grow exponentially with regard to the number of unit propagations applied (chapter 3).

Furthermore inequalities introduce a huge impact on inefficiency when applying reasoning by cases. Remember how $X$ implements reasoning by cases. It adds every literal of one single ground instance one by one, applies unit propagation and tests if for each of the literals of the chosen term the query is implied by the knowledge base.
Suppose that you choose a $proper^+$ term like $(X \neq a) \supset P(X) \vee Q(X)$ for reasoning by cases. Since we are only allowed to add one single ground instance (e.g. $(X = b) \supset P(X)$) this would imply that we can choose from the entire set of constants in the domain of discourse except the constant $a$.
Now think of the following terms contained in the knowledge base:

$$(X \neq a \supset P(X) \vee Q(X))$$
$$(X = a \supset \neg P(X) \vee Q(X))$$

Additionally we have the query $\exists X.Q(X)$. If we choose the first term for reasoning by cases we would have to add as many ground instances of this term as there are constants in the domain of discourse to prove that the query does not hold.
And since we think of knowledge bases that consist of more than $10^5$ $proper^+$ terms the set of constants is of a comparable size. Consequently this would

cause at least the same number of operations to be executed and therefore it would require a large amount of time in total.
This is of course a worst case scenario since we assume that we have to go through all constants, but there are other more common cases that also cause notable inefficiency. Think for example of the case where you have only some possible ground instances introduced by inequality but more than one level of reasoning by cases.
For each of the possible ground instances you now have to go through the levels of reasoning by cases. Then the search space belonging to reasoning by cases is multiplied by the number of possibilities caused by the mentioned inequalities. Note that additional possibilities may also exist at each level of reasoning by cases.
Comprising, all of the reasons mentioned above show that inequalities prevent an efficient implementation in our approach. Note that there might exist other approaches that can handle inequalities efficiently.

At this point we will also discuss the use of inequalities. Inequalities are mainly used to assign a possible infinite set of constants to a predicate. For example, $(X \neq a) \supset P(X)$ depicts the fact that $P(X)$ holds for every constant except $a$.
We could not suggest a practical case where the use of inequalities would be crucial or not be replaceable by another feature like a predicate for example. Think for example of the following $proper^+$ term contained in the knowledge base:

$$(X \neq John \supset isStudent(X))$$

This term states that everybody except John is a student. In our point of view such kind of statements are required very seldom and are of no use in the context of the example.
Consequently, the use of inequalities with regard to single individuals (constants) is very seldom. Normally inequality is used with properties, but not with individuals.

In contrast we support inequalities in the query since the use of inequalities allows us to ask queries of the following kind:

$$Query = (X \neq John \wedge isStudent(X))$$

The query asks, if there is another individual besides John that is a student. We think that this use of inequalities is very useful and would not be possible without inequalities.
Inequalities from a query are not directly involved in the reasoning process

itself and are only used when testing if a predicate is contained in the knowledge base or not. Therefore they can not cause any drawbacks concerning efficiency.

In consequence, we consider inequalities contained in the knowledge base to introduce a major drawback on efficiency, are difficult to handle and support no features that are essential in practice. Hence, we do not support inequalities in our implementation.

## 4.3 Encoding *proper+* terms

### 4.3.1 How we encode *proper+* terms

In order to meet the requirements mentioned above, we will first of all have to find a way to localize predicates and the corresponding equality-terms in a simple way. In particular we would like to solve this task with as few SQL-Statements as possible.
We will present an encoding now that allows an efficient access of all details of *proper+* terms.
Therefore we encode *proper* + $KB$-terms in a compound of numbers (or letters) because of two reasons mainly:

1. equality-terms are encoded in a way that allows simple comparing with other equality-terms

2. by using the encoding we have an easy and quick access of the predicates where *Unit Propagation* can be applied on

This allows a fast application of unit propagation and more features are presented when the encoding is introduced. Recall that unit propagation comprises two steps:

1. matching predicates

2. matching equality terms

Essentially we create two types of tables to represent *proper+*-terms, but it will be four tables in total since we need one type of tables multiple times.
The first type is able to contain disjunctive terms as well as unit terms while the second holds all equality terms of the *proper+* terms in the knowledge base.
We will make use of one table that contains only disjunctive terms and two

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| *Term ID* | *Predicate ID* | *Variables* | *Act. RbC-Level* | *Ori. RbC-Level* | *New* |

**Table 4.1:** The defined columns in the table "pTerm-pred" holding the predicates of a *proper*$^+$ term

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| *Term ID* | *Equality Term ID* | *Equality Encoding* | *RbC-Level* | *Updated* |

**Table 4.2:** The defined columns in the table "pTerm-equal" holding the equality terms of a *proper*$^+$ term

that will only hold unit terms - therefore we need four tables in total. Note that there are only two types of tables in total.

At this point it is only important to remember that there exist different tables, but their function will be discussed when the main features unit propagation and reasoning by cases are introduced. Especially, the presented detailed examples will provide a deeper insight.

The first type of table named "pTerm-pred" contains six columns (see table 4.1):

1. a term identifier
2. a predicate identifier
3. variable names
4. actual *Reasoning by Cases* level
5. origin *Reasoning by Cases* level
6. a flag named "new"

The term identifier is a unique mapping to a single *proper+*-term; the predicate identifier does the same for predicates. We need the variable names to extract the right equalities from the entire equality-term corresponding to the actual term and to determine if two equalities are unifiable. The remaining columns will be used for *Unit Propagation*, *Reasoning by Cases*, or both.

The second type of table named "pTerm-equal" contains five columns (see table 4.2):

1. a term identifier
2. an equality term identifier
3. the encoded equality term
4. the *Reasoning by Cases*-level
5. a flag named "updated"

The term identifier establishes the connection between the two types of tables. The equality term identifier is needed to store disjunctive equality terms that correspond to one *proper*+-term. Again the last two remaining attributes are used to apply unit propagation and reasoning by cases.
The equality term is encoded as described in the table. An encoded equality term contains four elements, namely the label of the variable, the position of the variable in the argument array of the predicate, a sign with values 0 and 1 to determine if the variable equals the following constant or may have any other value assigned except that constant mentioned. Please note that we could handle equality terms like $(X = Y \wedge Y \neq Z)$, but they are not supported in our implementation.
We handle variables that are not restricted in any way by assigning the 'don't care' symbol '$*$' to the attributes sign and constant. If a predicate has more than one variable the equality encoding of each variable are combined while using a delimiter.
We need both the attribute variable names and the attribute variable position since we use the first to extract the required equalities for a term from the encoded equality term in the table "pTerm-equal" and the second to compare equalities of the same variable position when applying unit propagation later on. This is due to the fact that variable identifiers might be different along various terms.
Please note that we need to convert equality terms of the $proper^+$ knowledge base into DNF. Only this conversion makes our representation of equality terms possible and additionally allows us a very fast method to test if equalities are fulfilled.

First of all when the equality terms are in DNF it is possible to split and store them in different entries in a table. This allows us to check equalities in a very simple way, since we only have to test if all the equalities of a single entry are fulfilled.
If they would not be in DNF the test if a given assignment to a variable holds would be rather difficult since it could be necessary to test various equalities and their possible combinations inside the entire equality term.

| *proper*$^+$*-term* | *Variable Name* | *Variable Position* | *Sign* | *Constant* |
|---|---|---|---|---|
| $(X = a) \supset P(X)$ | $X$ | 1 | 1 | $a$ |
| $(X \neq c) \supset P(X)$ | $X$ | 1 | 0 | $c$ |
| $P(X)$ | $X$ | 1 | $*$ | $*$ |

**Table 4.3:** How equality-terms are encoded; e.g. the equality-term of $(X = a) \supset P(X)$ will be encoded as $X|1|1|a$

In other words it would be necessary to apply some kind of satisfiability test that is not needed when using DNF (see previous chapter).

We need at least one additional different kind of table that contains the mapping of predicates to numbers whereby predicates with a negative sign are mapped to the negated number of the positive predicate. These are the numbers that are used as the predicate identifier in the table "pTerm-pred". If it is required we can make use of one more table that contains a mapping for constants. It might be necessary to convert constants in some domain of discourse into a more compact representation like a numerical identifier.
As mentioned before we use the table type "pTerm-pred" three times. In the first table we store proper terms that contain multiple predicates, the second and third table contain proper terms having only one predicate (*unit terms*). We use the third table only when we reason by cases.
As we will see later on, this fragmentation is very useful when we implement $X$ since it will allow us to reduce the amount of data that has to be inspected when reasoning by cases, for example.

In our encoding there exists *exactly one* entry for each *Unit Term* with the same predicate and sign. For example, the *Unit Terms* $(X = a) \supset \neg P(X)$ and $(X = b) \wedge \neg P(X)$ will have one entry in the table "pTerm-predU". The different equality terms are stored with the help of the equality identifier in the table "pTerm-equal".

It follows that it is quite easy to identify if a *Unit Term* with a specific predicate is contained in the knowledge base or not, because we only have to determine if there exists one entry for the predicate in the table.
If we need to check the equalities of this predicate we can do so by reading all equalities from the table "pTerm-equal" with the actual term identifier.
Since nearly every database allows the use of indices we create an index on the predicate identifier which results in a search time that is logarithmic in the size of the database [65]. This is how we establish a quick access of predicates.

| *Term Identifier* | *Predicate Identifier* | *Variable Names* |
|---|---|---|
| 1 | $-1$ | $X$ |
| 1 | 2 | $X, Y$ |

**Table 4.4:** The encoding of $(X \neq a \wedge Y = b) \vee (X = d) \supset \neg P(X) \vee Q(Y, X)$ in the table "pTerm-pred"

| *Term Identifier* | *Equality Identifier* | *Equality Encoding* |
|---|---|---|
| 1 | 1 | $X\|1\|0\|a - Y\|2\|1\|b$ |
| 1 | 2 | $X\|2\|1\|d - Y\|1\| * \|*$ |

**Table 4.5:** The encoding of $(X \neq a \wedge Y = b) \vee (X = d) \supset \neg P(X) \vee Q(Y, X)$ in the table "pTerm-equal"

### 4.3.2 An Example Encoding

While we talked about properties of single parts in the encoding so far, we would like to present all parts of the encoding in one example now. Tables 4.4 and 4.5 show how the term

$$(X \neq a \wedge Y = b) \vee (X = d) \supset \neg P(X) \vee Q(Y, X)$$

is encoded in the database table "pTerm-pred" and "pTerm-equal".
In our example the predicate $P$ is mapped to 1 and $Q$ to 2. We excluded the values for the RbC-Level and other flags, because they are not of concern here and will be discussed later on.

All together we presented a quite simple encoding of *proper+*-terms, that allows us a fast comparison of equalities and a useful foundation to decide if a predicate is included in the knowledge base or not in a quick way.

## 4.4 Unit Propagation

### 4.4.1 Implementation

For the given representation of *proper+*-terms, we will now present an algorithm for unit propagation. The algorithm tries to use as much database features as possible given the current encoding of *proper+*-terms.
Here unit propagation can basically be applied if we have a *proper+*-term with more than one predicate (*disjunctive term*) and a *proper+*-term with exactly one predicate (*unit term*) which is the negated version of one of the predicates of the disjunctive term.

At this point the reader must be aware of the crucial difference between an application of unit propagation in the propositional case and the first-order case.
When unit propagation is successfully applied on a clause this original clause is *deleted* in the propositional case. But since $proper^+$ terms can represent even an infinite number of ground instances, we are not allowed to delete the disjunctive term where unit propagation was applied to.
Only when the $proper^+$ term represents exactly one ground instance we are allowed to delete the disjunctive term since this case is equal to the propositional case (e.g. $(X = a \supset P(X) \vee Q(X))$). An example is given in the subsequent chapter.
As a consequence, the number of *proper+*-terms in our knowledge base will *increase* in general instead of decrease when we apply unit propagation. Only redundant terms and terms that represent exactly one ground instance can be deleted. This fact has an obvious disadvantage since it causes a growth of the original knowledge base (see previous chapter).
And an increased number of disjunctive terms will also increase the set of disjunctive terms that will be of concern when reasoning by cases is applied. Consequently, more possibilities than introduced by the original knowledge base must be tested as we will see when we present the implementation of reasoning by cases.

In our presentation of unit propagation we will not always state explicitly that the disjunctive terms remain in the knowledge base. But we have of course to take care of this property when implementing unit propagation. Therefore we will first discuss how unit propagation is applied in general and then we present how we handle the just mentioned topic.

In general a single successful application of unit propagation in our case exists of three steps:

1. Identify those pairs of *proper+*-terms where unit propagation can be applied on
2. Check the equalities of each pair for compatibility
3. Adapt equalities in the disjunctive *proper+*-term

As said in the last section our encoding of *proper+*-terms allows a fast execution of the first step. To explain how we do this, we have to say which data is actually stored in the four tables used.
The first table contains disjunctive *proper+*-term ("pTerm-pred") only, the second and third only store unit terms ("pTerm-predU1" and "pTerm-predU2") and "pTerm-equal" is the one and only table that holds the equalities.

In other words the tables "pTerm-predU1" and "pTerm-predU2" will not contain multiple lines with the same term identifier as "pTerm-pred" has to have multiple lines with the same term identifier.
For the moment it is only necessary to know that we have one table for disjunctive terms, one for unit terms and one for equalities. The table "pTerm-predU2" used for unit terms will be of concern when we introduce reasoning by cases in the next section.

We now identify the required pairs of terms by using the *Join*-Operator of SQL and apply it to the table that contains only disjunctive term and the table that holds only unit terms. Our join attribute is the predicate identifier and the join condition is that the disjunctive term must hold the complementary predicate of the unit term.
For example if a disjunctive term contains the predicate $P$ mapped to 1 the join condition is satisfied when a unit term holds the negated predicate $\neg P$ mapped to $-1$.
It is commonly known that a join operation is one of the most expensive operators in databases in general [23] since it requires quadratic complexity in relation to the number of datasets in both tables.
There exist different types that reduce complexity like the use of bucket hashing and the use of indices [65, 23]. While our implementation is build on MySQL we make use of the *Index Join*.
At this point we will not delve into complexity issues and will continue with the introduction of our approach. In the following chapter more details will be discussed.
So far we determined the possible unit propagation pairs. For each of those pairs we first copy the actual disjunctive term and set the corresponding "new" flag to the value 2. The value 2 indicates that this disjunctive term is a newly generated one. Handling things this way is necessary to allow every possible application of unit propagation.
As said before one disjunctive term may represent various ground instances. And if we would not copy the disjunctive term and apply all changes on the copy the update of equalities might prevent further application of unit propagation.
Suppose the following set of terms:

$$X = a \vee X = b \supset P(X) \vee Q(X)$$
$$X = a \supset \neg P(X)$$
$$X = b \supset Q(X)$$

If we now would apply reasoning by cases copying the disjunctive term before, we would gain the following set of terms:

$$X = a \supset Q(X)$$
$$X = a \supset \neg P(X)$$
$$X = b \supset \neg Q(X)$$

As you can see no further applications of unit propagation are possible, because of the updated equality term and the fact that no disjunctive term is available anymore. Hence, we would not be able to gain the term $(X = b \supset P(X))$ although this would be required by a correct application of unit propagation.
We mark the copy of the disjunctive term with a specific value, because these newly generated terms must be reconsidered by *all* unit terms. Consider the following example:

$$X = a \vee X = b \vee X = c \supset P(X) \vee Q(X) \vee R(X)$$
$$X = a \supset \neg P(X)$$
$$X = a \vee X = b \supset \neg Q(X)$$
$$X = c \supset \neg R(X)$$

This would generate the following new terms:

$$X = a \supset Q(X) \vee R(X)$$
$$X = a \vee X = b \supset P(X) \vee R(X)$$
$$X = c \supset P(X) \vee Q(X)$$

From this set for example the second term must be reconsidered by the original unit terms since unit propagation can be applied again. In fact, the term $X = b \supset R(X)$ results when using the original unit term $X = a \supset \neg P(X)$ with the second term of the above set.
Note that a reconsideration with all of the unit terms has not to take place with the already included disjunctive terms ($new = 1$), because every possible combination has been already determined.
Therefore, we will use the join operator with all unit terms and newly generated disjunctive terms and with all new unit terms and all disjunctive terms. We can do so, because we can identify the new unit terms as well as the newly generated disjunctive terms by using the flag "new".

When we have copied the disjunctive term we are ready to check if the equalities of the concerned predicates are compatible. Therefore we use the data generated by the join of the two tables whereby we use the term identifier of both terms of the pair to select the corresponding equalities from the table "pTerm-equal".
Two equality terms are compatible if they are not mutually exclusive. For example, $(X = a)$ and $(X \neq b)$ are compatible, but $(X = a)$ and $(X \neq a)$ are not (see also Chapter 3).

In a few lines of *Prolog* code we can check if the equalities match. If they do so it might be necessary to adapt the equality term of the disjunctive term. Consider again the example considered before: the updated equality term of the combination of $(X = a)$ and $(X \neq b)$ will result in $(X = a)$.
Realize that inequalities add up in the updated equality term; e.g. $X \neq a$ and $X \neq b$ result in $X \neq a \wedge X \neq b$. We already remark at this point that this is the reason for a major drawback concerning complexity. A closer examination of equality terms and their interaction will be discussed in the next chapter. Note that an equality term of a unit term will never be changed.

When updating equalities we make use of the flag "updated" to take care that we do not use any new generated equality term during one application of unit propagation. After a single and complete application the value of the flag is reset.
If equalities match we can apply the essence of unit propagation since we are working on a copy and simply delete the line from the table with the corresponding term- and predicate identifier from "pTerm-pred". If only one predicate remains, we copy the term to a table that contains only unit terms.

To summarize our implementation of unit propagation, we sketch the algorithm in the following lines:

---

1. Determine all possible pairs where unit propagation could be applied while considering the following sets of terms:
   - All disjunctive terms $(new = 1)$ and all new unit terms $(new = 1)$; then set the flag "new" to 0 at all unit terms
   - All new disjunctive terms $(new = 2)$ and all unit terms; then set the flag "new" to 1 at all disjunctive terms
2. For each pair do the following:
   - Copy the actual disjunctive term and proceed on the copy only. Use the flag "new" to mark this copy as a newly generated disjunctive term $(new = 2)$; if the disjunctive term already exists do nothing.
   - Test if equalities of the unit- and disjunctive term are compatible
   - If so apply unit propagation and adapt equalities in the disjunctive term and if there is only one predicate left in the disjunctive term than copy the term to the table "pTerm-predU" and delete the entry in "pTerm-pred"; else delete the copy of the disjunctive term again

3. Repeat the steps 1 and 2 until the set of possible unit propagation pairs is empty

---

Note that the meaning of the flag "new" is different among disjunctive and unit terms. Unit terms can be new (1) and old (0). Disjunctive terms are always new (1) when unit propagation is applied since they always have to be considered when we determine the set of possible unit propagation pairs. Additionally, we need the value 2 to indicate newly generated disjunctive terms in the last cycle of the algorithm. The value 0 is used when we reasoning by cases is applied later on.

We do not have to check the validity of a pair, because one single predicate will be contained only once in the tables that contain only unit terms. Hence, it is not possible that some earlier pair deletes the predicate in the same term that is of concern in the actual pair.

We use the flag "new" indirect as termination criterion of our algorithm. In the beginning the flag is set to 1 and every unit term is thought of when determining possible unit propagation pairs. After we have applied preprocessing for example all unit terms will be marked as visited.
After determining the first set of possible pairs all unit terms are considered to be visited and consequently the flag is set to 0. We can act in this way, because if a unit term is not to be considered a part of a possible unit propagation pair - in other words the complement of the predicate is not contained in any disjunctive term - it will be *never* a part of a possible unit propagation pair.
So in the next turn those visited unit terms are not of concern anymore - at least when we consider only the old disjunctive terms in the knowledge base. As described before the newly generated disjunctive terms reconsider all available unit terms since this is necessary. But the flag "new" of these disjunctive terms is directly set to "1" again, so that in the next turn they only are of concern in combination with new generated unit terms.
Please note that a newly generated unit term is only marked as new, when it is not already included in the knowledge base.

The algorithm terminates when the set of pairs is empty. Since the set depends on the newly generated unit terms and disjunctive terms it is obvious that this algorithm will always terminate.

While it is legal to mark the visited unit terms and not to consider them anymore with the old disjunctive terms, it is not correct to do the same for disjunctive terms. If a disjunctive term is reduced to a unit term by applying unit propagation all remaining disjunctive terms must be reconsidered to determine the possible unit propagation pairs. Simply, because of the fact that the newly created unit term was not of debate before and therefore can

| *Term Identifier* | *Predicate Identifier* | *Variable Names* | *new* |
|---|---|---|---|
| 1 | 1 | $X$ | 1 |
| 1 | 2 | $X, Y$ | 1 |
| 1 | 3 | $X, Y, Z$ | 1 |

**Table 4.6:** The encoding of $((X \neq e) \wedge (Y = a)) \vee ((X = a) \wedge (Y \neq e) \wedge (Z = b)) \supset P(X) \vee Q(X, Y) \vee R(X, Y, Z)$ in the table "pTerm-pred"

cause new pairs.

Comprising the presented algorithm is used to implement one of the main features introduced by $X$. As we said before we need an efficient implementation of unit propagation to enable a efficient implementation of the entire reasoning procedure. For this reason the use of database features was essential.

### 4.4.2 Example

To give a more intuitive feel how our approach applies unit propagation, we give a detailed example at this point. In this example we only like to show how unit propagation is executed in our implementation. Again you should be aware of the crucial difference between unit propagation applied in the propositional case and the first-order case.
Additionally, note that we allow inequalities even if we assume that they are not contained in our knowledge base. Hence our presented method supports inequalities as said before and is therefore fully compatible with the original reasoning procedure.

Suppose the following $proper^+$ terms are in our knowledge base:

1. $(X \neq e \wedge Y = a) \vee$
   $(X = a \wedge Y \neq e \wedge Z = b) \supset P(X) \vee Q(X, Y) \vee R(X, Y, Z)$
2. $X = a \supset \neg P(X)$
3. $\neg P(X)$
4. $(X = a \wedge Y = g) \vee Y \neq c \supset \neg Q(X, Y)$

The tables 4.6 and 4.7 show the encoding of the disjunctive proper term and the unit terms while table 4.8 holds the corresponding equality terms. The predicates are mapped to numbers in their alphabetical order.

When we now go through the algorithm step by step we first of all determine all possible unit propagation pairs. Here a pair simply consists of two

| *Term Identifier* | *Predicate Identifier* | *Variable Names* | *new* |
|---|---|---|---|
| 2 | $-1$ | $X$ | 1 |
| 3 | $-2$ | $X, Y$ | 1 |

**Table 4.7:** The encoding of $(X = a) \supset \neg P(X), \neg P(X))$ and $\forall(((X = a) \wedge (Y = g)) \vee (Y \neq c) \supset \neg Q(X, Y)$ in the table "pTerm-predU1"

| *Term Identifier* | *Equality Identifier* | *Equality Encoding* |
|---|---|---|
| 1 | 1 | $X\|1\|0\|e - Y\|2\|1\|a - Z\|3\| * \|*$ |
| 1 | 2 | $X\|1\|1\|a - Y\|2\|0\|e - Z\|3\|1\|b$ |
| 2 | 1 | $X\|1\| * \|*$ |
| 2 | 2 | $X\|1\|1\|a$ |
| 3 | 1 | $X\|1\|1\|a - Y\|2\|1\|g$ |
| 3 | 2 | $X\|1\| * \| * -Y\|2\|0\|c$ |

**Table 4.8:** The encoding of all equality terms mentioned in the example

term identifiers, in our implementation it contains as much data as possible to reduce access to the database.
As said before we join the two tables using the complement of a predicate as join condition. The result of this operation can be seen in table 4.9.

Consequently we have the two pairs (1,2) and (1,3). First we copy the disjunctive term (term identifier of copy: 4) and then we check the equalities of the pair (1,2).
Since both disjunctive term and unit term have multiple disjunctive equality terms and all of them are compatible, the number of equality terms increases and reaches three in total. In fact there exist four terms, but two are identical to each other.
Equal terms are not stored in the knowledge base, because every entry must be unique in regard to a term identifier and a reasoning by cases level. This feature is implemented by the table definition and MySQL.

| *dt.Term ID* | *dt.Predicate ID* | *ut.Term ID* | *ut.Predicate ID* |
|---|---|---|---|
| 1 | 1 | 2 | $-1$ |
| 1 | 2 | 3 | $-2$ |

**Table 4.9:** The join of the two tables "pTerm-pred" and "pTerm-predU1" from the example where "dt" indicates the table with disjunctive as "ut" indicates the table with unit terms

| *Term Identifier* | *Predicate Identifier* | *Variable Names* | *new* |
|---|---|---|---|
| 1 | 1 | $X$ | 1 |
| 1 | 2 | $X, Y$ | 1 |
| 1 | 3 | $X, Y, Z$ | 1 |
| 4 | 2 | $X, Y$ | 1 |
| 4 | 3 | $X, Y, Z$ | 1 |
| 5 | 1 | $X$ | 1 |
| 5 | 3 | $X, Y, Z$ | 1 |

**Table 4.10:** After the fist application of unit propagation the line with the term identifier "4" and predicate identifier "1" is deleted in the table "pTerm-pred"

The same is done for the other pair and after the first cycle in our algorithm the tables "pTerm-pred" and "pTerm-equal" look like depicted in the tables 4.10 and 4.11. The second copy of the disjunctive term has the term identifier 5.

According to the introduced scheme we again identify the possible pairs for unit propagation. Since there are no new created unit terms from the last cycle, but new disjunctive terms we combine them with all unit terms contained in the knowledge base.
The resulting pairs are: (4,3) and (5,2). As described before, we again copy the disjunctive term, check equalities and since there are compatible equality terms, we can apply unit propagation successfully.
And this time a new unit term is generated. In fact both pairs generate a new unit term, but they are identical to each other, so that there is only one new unit term and therefore only one additional term identifier in the end.
The result is shown in the table 4.12. This time we only show the entries of "pTerm-predU", which also holds the just generated new unit term, since the table "pTerm-pred" holds no new disjunctive terms. Additionally we depict the equality terms of all unit terms in the table 4.13

Note that in all entries of the table "pTerm-predU" the value of the flag "new" is set to 0 except of the just created unit term.
Now that we have finished the second cycle of our algorithm, the algorithm terminates although we generated a new unit term, because there are no more possible unit propagation pairs in the knowledge base left.

| *Term Identifier* | *Equality Identifier* | *Equality Encoding* |
|---|---|---|
| 1 | 1 | $X\|1\|0\|e-Y\|2\|1\|a-Z\|3\|*\|*$ |
| 1 | 2 | $X\|1\|1\|a-Y\|2\|0\|e-Z\|3\|1\|b$ |
| 2 | 1 | $X\|1\|*\|*$ |
| 2 | 2 | $X\|1\|1\|a$ |
| 3 | 1 | $X\|1\|1\|a-Y\|2\|1\|g$ |
| 3 | 2 | $X\|1\|*\|*-Y\|2\|0\|c$ |
| 4 | 1 | $X\|1\|0\|e-Y\|2\|1\|a-Z\|3\|*\|*$ |
| 4 | 2 | $X\|1\|1\|a-Y\|2\|0\|e-Z\|3\|1\|b$ |
| 4 | 3 | $X\|1\|1\|a-Y\|2\|1\|a-Z\|3\|*\|*$ |
| 5 | 1 | $X\|1\|0\|a-Y\|2\|1\|g-Z\|3\|1\|b$ |
| 5 | 2 | $X\|1\|0\|e-Y\|2\|1\|a-Z\|3\|*\|*$ |
| 5 | 3 | $X\|1\|1\|a-Y\|2\|0\|c-Y\|2\|0\|e-Z\|3\|1\|b$ |

**Table 4.11:** The encoding of all equality terms mentioned in the example after the first application of unit propagation

| *Term Identifier* | *Predicate Identifier* | *Variable Names* | *new* |
|---|---|---|---|
| 2 | $-1$ | $X$ | 0 |
| 3 | $-2$ | $X, Y$ | 0 |
| 6 | 3 | $X, Y, Z$ | 1 |

**Table 4.12:** After the second application the following unit terms are stored in the table "pTerm-predU"

| *Term Identifier* | *Equality Identifier* | *Equality Encoding* |
|---|---|---|
| 2 | 1 | $X\|1\|*\|*$ |
| 2 | 2 | $X\|1\|1\|a$ |
| 3 | 1 | $X\|1\|1\|a-Y\|2\|1\|g$ |
| 3 | 2 | $X\|1\|*\|*-Y\|2\|0\|c$ |
| 6 | 1 | $X\|1\|0\|e-Y\|2\|1\|a-Z\|3\|*\|*$ |
| 6 | 2 | $X\|1\|1\|a-Y\|2\|0\|e-Y\|2\|0\|c-Z\|3\|1\|b$ |
| 6 | 3 | $X\|1\|1\|a-Y\|2\|1\|a-Z\|3\|*\|*$ |
| 6 | 4 | $X\|1\|1\|a-Y\|2\|1\|g-Z\|3\|1\|b$ |

**Table 4.13:** The encoding of all equality terms corresponding to the unit terms after applying unit propagation twice

## 4.5 Evaluation of the Query

### 4.5.1 Introduction

In the following sections we will describe how disjunctions, conjunctions, quantifiers and single predicates are evaluated in our implementation. Since we already discussed one of the two main features of $X$ we now focus on how queries are decomposed and evaluated. In fact, we will use the decomposition as introduced in $X$ in general.

Think for example of the following query $\exists X.P(X)$. In $X$ it is defined that this query is answered by using the set of constants $H^+_{k+1}$ and thereby generating many ground instance of the query and test if this ground instance is included in the knowledge base.
When you recall that we would like to deal with large knowledge bases that contain more than $10^5$ terms which implies round about the same number of constants it is quite obvious that substituting a variable with each constant of the set is not an efficient way to answer the query.

Therefore we will present methods that are more efficient especially for the two quantifiers. At this point we will make use of the fact that we do not support inequalities in our implemented reasoning procedure.

Finally, we would like to mention that we did not invest in a user friendly input of the query. The input of a query is predefined by a given structure and very complex queries are not supported yet although they could be handled by the reasoning procedure itself.
We acted in this way because normally we only have very short queries and the main focus of this work lies on the efficiency and implementation of the reasoning procedure itself

We already mentioned in the second chapter that the recursive definition of the reasoning procedures presented are not difficult to implement. For example, the decomposition of complex formulas is quite similar to language processing with grammar rules which is a well known strength of PROLOG [9].
From there those parts of the implementation - namely the skeleton of $X$ - will not play any major role here. Of course we will describe how we evaluate a query as said before, but we will not present solutions how very complex queries are decomposed.

### 4.5.2 Format of the Query

In our implementation we assume first of all that the query $Q$ is in *DNF*. We assume that every formula $\alpha_i$ has the following form when $e_i$ denotes

equality terms and $P_j$ a predicate or its complement:

$$\alpha_i = (e_1 \wedge ... \wedge e_j \wedge P_1 \wedge ... \wedge P_k)$$

Then a query $Q$ must have the following format:

$$\forall X_1 \forall X_2 \ ... \forall X_n \exists Y_1 \exists Y_2 \ ... \ \exists Y_m \ (\alpha_1 \vee ... \vee \alpha_n)$$

The reason why we do not support queries like $\exists X.\forall Y.\ \alpha$ is due to the implementation of the $\forall$-quantifier that we will discuss in a subsequent section. To give you a more intuitive feel which kind of queries the implementation can answer we provide some example queries:

- $(X = a \wedge P(X)) \vee (X = b \wedge Q(X))$
- $\exists X.(X \neq a \wedge P(X))$
- $\exists Y.((X = a \wedge P(X,Y)) \vee (X = c \wedge Q(X,Y)))$
- $\forall X.\exists Y.(P(X,Y) \wedge Q(X,Y))$

In general, we do not support queries like $(e \supset P)$. Only $\forall(e \supset P)$ is handled in our implementation.
We do not support the evaluation of equality terms only. For instance, we can not handle a query like $\forall(X \neq a)$. In addition, queries like $\forall(X = a \wedge X \neq a \supset P(X))$ are not supported. This is due to the fact that equality terms are not evaluated in a distinct way. We think that it would be no major problem to evaluate equality terms only, but mainly lack of time caused this restriction.

### 4.5.3 Quantifier-free Queries

We begin with the description of the way how queries only containing a single predicate like $(X = a \wedge P(X))$ is tested. By using the table that holds the encoding of every predicate, we would search the table that stores the unit terms for the corresponding encoding. If it would contain the encoding, we would determine the corresponding equality term and test if the equalities from the equality included in the query and in the database match. If it would not contain the encoding we answer "unknown".
The equality of $proper^+$ terms $(t_1 = t_2)$ does not play a major role in our implementation, but it would be possible to compare the encoding of $proper^+$ terms in a simple way.

If the query consists of a conjunction of predicates all of them must be known to be true. Every single predicate is tested as described before. Note

that $X$ does not support reasoning by cases when the query only consists of a conjunction. Recall, that in our implementation the application of reasoning by cases does not depend on the structure of the query, but only on the user-defined value. Hence, a query only consisting of conjunctions could be answered with the help of reasoning by cases.
And since the query is in DNF we simply have to test if one of the disjunctions is known to be true.

A query can be of a complex format and will then be decomposed as defined in $X$. Suppose the following query:

$$Query = ((X = a \wedge Y = b \wedge P(X) \wedge Q(Y))) \vee (X = a \wedge R(X))$$

This query would be decomposed in the two parts $((X = a \wedge Y = b \wedge P(X) \wedge Q(Y)))$ and $(X = a \wedge R(X))$ and each of the parts would be tested by the methods presented before.

Comprising, our implementation of quantifier-free queries is nearly equal to the definitions presented within $X$.

### 4.5.4 The Existential

#### Introduction

In the procedure $X$ the existential is implemented through the substitution of domain constants that are contained in $H_k^+$.
Recall that the set $H_k^+$ holds every constant from the query, every constant contained in the knowledge base and $k$ additional constants contained nowhere else.
In fact this implies that a query that contains an existential is answered by substituting the corresponding variables by domain constants and for each created ground formula it is tested if it is contained in the knowledge base or not.
It is obvious that this can not be done efficiently since we have a large number of constants when we assume knowledge bases with more than $10^5$ $proper^+$ terms.

There exist different ideas to solve this problem. One idea would store all constants used among a single predicate and thereby decrease the number of possible constants in an essential way, because it would be only necessary to substitute the variables with these constants.
This is due to the fact that a predicate can only be fulfilled by the constants that are connected to it. Suppose the following $proper^+$ terms to be contained in the knowledge base:

$$X = a \wedge Y = b \supset Q(X, Y)$$
$$X = d \wedge Y = f \wedge Z = e \supset R(X, Y, Z)$$
$$X = g \supset P(X)$$

Now assume the query $\exists X.P(X)$. Then it would make no sense to substitute the variable by all the constants contained in the knowledge base, because only the constant $g$ is connected to the predicate $P$.
Consequently, if you use the method of only using constants that are connected to one predicate you have decreased the number of possible constants immensely.
This should provide a deeper insight in the problem described above. In our example there exist five constants and thereby five ground instances that would serve as input for the reasoning procedure.
This might cause reasoning by cases for each of the ground instances, but at least the application of the test if a ground instance is part of the knowledge base although it is predetermined that none of them will succeed. Note that the majority of constants is useless for this task.
And now assume that there are thousands of constants contained in the knowledge base. This is the reason why we had to decrease the set of constants in a more restrictive way than done in $X$ to solve the existential.

The approach that we use in our implementation to solve the existential is even more effective than the first presented idea.
If the query exists of only one predicate we simply check if the predicate from the query is included in the knowledge base.
If it is contained in the knowledge base there must be a constant that fulfills the query otherwise the existential fails. If the query consists of conjunctions of predicates we check if the different equalities for the same variable are not mutually exclusive. If they are not mutually exclusive there exists a substitution that satisfies the predicates.

Further details concerning this method and the corresponding assumptions are presented in the following subsections.

### Decomposition of the Query

Additionally to the assumptions about the query made in the last section we assume at this point that the query contains no other quantifiers than the existential.
Taking those assumptions into account the format of a general query containing an existential is given by:

$$\exists X_1 \exists X_2 \ ... \ \exists X_n \ \alpha, \ \alpha \text{ is in DNF}$$

Since $\alpha$ is in DNF we can split it up into the conjunctions $\alpha_1$ to $\alpha_m$. Therefore we can subdivide the query into the following distinct queries [59]:

$$\begin{aligned} Query_1 &= \exists X_1 \exists X_2 \ ... \ \exists X_n \ \alpha_1 \\ Query_2 &= \exists X_1 \exists X_2 \ ... \ \exists X_n \ \alpha_2 \\ &... \\ Query_m &= \exists X_1 \exists X_2 \ ... \ \exists X_n \ \alpha_m \end{aligned}$$

We use those parts of the original query to answer the entire query. So if we can show that a $Query_i$ is known to be true the query can be answered immediately. If this can not be shown for any of the queries the answer to the query is unknown. We can act in this way since $\alpha$ is in DNF.

### Testing the Query

In this section we present the testing method that is used when the query contains an existential. This method will be used in the entire algorithm to check if the knowledge base implies the given existential.

The test we apply for each of the mentioned queries is depicted in the following tasks:

---

1. Test if each predicate of the conjunction is stored as unit term in the database

2. If so, read all of the corresponding equalities into the memory and test if the equalities with regard to the existential and the query itself are not mutually exclusive.
   If they are mutually exclusive fail, else return *true*.

---

The first task (1.) can be easily accomplished by as many SQL-statements as there are predicates in the query. The "SQL-Select" will return a term identifier for each of the predicates. If there exists no unit term for one of the predicates the query can not be known to be true.

For the second task (2.) we use the set of term identifiers returned by the first task. We use them to read the corresponding equalities from the table "pTerm-equal". Now we have to examine the equality terms carefully. First of all we have to take care of the restrictions introduced by the existential. For example, if we have the following query:

$$\exists X \exists Y.(P(X,Y) \wedge Q(Y,a))$$

In this example it is implied that the equalities of the variable $Y$ must be compatible. Now suppose the following unit terms to be in the knowledge base:

$$(X_1 = a \wedge X_2 \neq b \supset P(X_1, X_2)$$
$$(Y_1 = c \wedge Y_2 = a \supset Q(Y_1, Y_2)$$

Note that we allow inequalities in our example since our testing method concerning the existential supports the use of them.
To identify the corresponding equalities in each unit term we use the variable position in each of the predicates that is occupied by the variable that is bound to the existential. The variable position provides the identification of the matching equality term within each of the unit terms.
Returning to the example we have to test if $X_2 \neq b$ and $Y_1 = c$ are compatible since they are restricted by the variable $Y$ within the existential. In this example they are not mutually exclusive and therefore compatible.
As said before two equality terms are compatible if they are not mutually exclusive. For example $(X = a)$ and $(X \neq b)$ are compatible, but $(X = a)$ and $(X \neq a)$ are not as $(X = a)$ and $(X = b)$ are neither.
In fact the test if the equalities of all the unit terms of concern are not mutually exclusive is a key feature here.
Additionally, we have to test if the equality term of the unit term satisfies the restrictions made by the constants contained in the query. In our example those are also satisfied.

If all tests succeed the query is answered as known to be true. Otherwise the test will fail and if there is no reasoning by cases allowed, the query will be answered as unknown.
While testing the equalities of a query it is important to be aware of the fact that it is not required to determine one specific value for the variable so that the current predicate holds. An existential only implies that there is a substitution that makes the predicate true.
Take a look at the following example where we apply reasoning by cases:

$$KB = \{(P(a) \vee Q(b)), (\neg P(a) \vee Q(a))\}$$
$$Query = \exists X.Q(X)$$

In particular that means that when we add $P(a)$ to the KB it follows that $Q(a)$ is in the KB and hence the query holds for the first predicate.
If we now add $Q(b)$ - as it is required by reasoning by cases - again the query is known to be true. So, when we apply reasoning by cases there is *no* connection between the equalities of the predicates that are used in reasoning by cases.

As said when we presented the implementation of reasoning by cases the test of a query is a subroutine in the entire algorithm.

### 4.5.5 The ∀-Quantifier

The ∀-quantifier is handled within the reasoning procedure $X$ in a similar way as the existential. But now *every* substitution of a variable by a constant of the set $H_k^+$ must fulfill the corresponding predicate. Hence, it is necessary to test every possible ground instance. It is obvious that this approach would involve a large set of ground instances to be tested.
But since we do not support inequalities it is not necessary to substitute every possible constant from the set $H_k^+$. Our implementation is based on the following properties.

**Lemma 13** *(Levesque, [38])*
*Let $\alpha$ be a sentence that contains no equalities and $*$ denotes a surjection from $\mathcal{C}$ to $\mathcal{C}$.*

$$\textit{Then } I \models \alpha^* \textit{ iff } I^* \models \alpha.$$

**Proof** This was shown and discussed in [38]. In general, the proof is based on the surjection $*$ while every constant that is contained in KB or $\alpha$ is mapped bijectively and the new constant $n^*$ is mapped to a constant $c_i$, so that $c_i \in KB$ or $c_i \in \alpha$. ∎

To make use of this result in our case the knowledge base nor the query may contain equality terms. Since we do not support inequalities and equality terms are in DNF we can generate a corresponding e-free knowledge base $\{\forall(c)\}$ that is equivalent to the original knowledge base, but contains no equalities at all (see Chapter 3).
In the following we use the notation of the e-free KB $\{\forall(c)\}$. Recall, that we use $KB \models_\epsilon \alpha$ as an abbreviation for $\varepsilon \cup KB \models \alpha$ to indicate the use of the standard model of equalities.

**Lemma 14** *Let KB be an e-free KB $\{\forall(c)\}$ and $\alpha$ a sentence that contains no equalities.*

$$\textit{Then } KB \models_\epsilon \alpha^X_{n^*} \Rightarrow KB \models_\epsilon \alpha^X_{c_i},\ c_i \in \mathcal{C}$$

**Proof** Let $I \models_\epsilon KB$. Then $I \models_\epsilon KB^*$. By Lemma 13 $I^* \models_\epsilon KB$ holds. Provided that $I^* \models_\epsilon \alpha^X_{n^*}$ holds, also $I \models_\epsilon (\alpha^X_{n^*})^*$ holds.

$$\text{Then } I \models_\epsilon \alpha^X_{c_i},\ c_i \in \mathcal{C}.$$

**Theorem 15** *Let KB be an e-free knowledge base* $\{\forall(c)\}$. $\alpha$ *is a sentence that contains no equality terms.*

$$\textit{Then } KB \models_\epsilon \forall X.\alpha \textit{ iff } KB \models_\epsilon \alpha^X_{n^*},\ n^* \textit{ a new constant.}$$

**Proof** "$\Rightarrow$": obvious. "$\Leftarrow$": By the assumption and Lemma 14 $KB \models_\epsilon \alpha^X_{c_i}, c_i \in \mathcal{C}$ and consequently, $KB \models_\epsilon \forall X.\alpha$. ∎

This result allows us to test if $\alpha$ is true for all possible values of the variable $X$ by applying only one single substitution.
Until now we assumed that $\alpha$ does not contain any equality terms. First we will add inequalities. For simplicity, we only examine the case of a single unary predicate and a single binary predicate, respectively. The general case of a disjunction of predicates of any arity follows by a similar argument. Note that inequalities are allowed in the query only.

**Lemma 16** *Let KB be an e-free KB* $\{\forall(c)\}$.

$$\textit{Then } KB \models_\epsilon \forall X.(X \neq a) \supset P(X) \textit{ iff } KB \models_\epsilon \forall X.P(X).$$

**Proof** "$\Leftarrow$": obvious. "$\Rightarrow$": Let $KB \models_\epsilon \forall(X \neq a) \supset P(X)$. Then $KB \models_\epsilon P(n^*)$, when $n^*$ is a new constant. By Lemma 15, $KB \models_\epsilon \forall X.P(X)$ follows. ∎

This shows that it is possible to handle a semi-restricted variable contained in a predicate in the same way as a free variable.
Now we allow besides inequalities also equalities and gain the following result.

**Lemma 17** *Let KB be an e-free KB* $\{\forall(c)\}$.
*Then* $KB \models_\epsilon \forall X.\forall Y.(X = a \wedge Y \neq b) \supset P(X,Y)$ *iff* $KB \models_\epsilon \forall X.\forall Y.(X = a) \supset P(X,Y)$.

**Proof** "$\Leftarrow$": obvious. "$\Rightarrow$": Let $KB \models_\epsilon \forall X.\forall Y.(X = a \wedge Y \neq b) \supset \alpha$, then $KB \models_\epsilon \forall Y.(y \neq b) \supset P(a,Y)$. Then $KB \models_\epsilon P(a,n^*)$, when $n^*$ is a new constant. By Lemma 15, $KB \models_\epsilon \forall X.\forall Y.P(a,Y)$. Consequently, $KB \models_\epsilon \forall X.\forall Y.(X = a) \supset P(X,Y)$. ∎

This means that if a variable is bound to a constant we test if the corresponding equality term in the knowledge base contains a matching equality for the variable. If an inequality is contained in the equality term of the query we bind the variable of concern to a new constant and then test if the corresponding equality matches.
Comprising, these observations allow us to implement the $\forall$-quantifier with a minimum of required substitutions.

### 4.5.6 The Combination of Quantifiers

Until now we only discussed the two quantifiers distinct from each other. In this section we will investigate, if the presented implementations for each of the quantifiers can be combined. Especially, we explore which connections the quantifiers introduce among the quantified variables when we combine the two quantifiers.
In our implementation we only support queries that contain both kind of quantifiers if they can be converted into the following format:

$$\forall X_1 \forall X_2 \ ... \forall X_n \exists Y_1 \exists Y_2 \ ... \ \exists Y_m \ \alpha, \ \alpha \text{ in DNF}$$

Note that this format of the query is necessary to allow an efficient handling of the quantifiers in our approach. Especially the way we implemented the $\forall$-quantifier does not allow a combination of quantifiers like $\exists X.\forall Y.P(X,Y)$. If we now simply apply our methods as introduced in the two sections before, we would test for every $X_1$ to $X_n$ that are contained in each of the disjunctive parts of $\alpha$ if they hold for every possible constant. In the same way we would test if the equality terms belonging to the variables $Y_1$ to $Y_m$ within a single disjunctive part of the query would match.
For example, suppose we have the following knowledge base:

$$KB = \{(Y = a \supset P(X,Y)), (Y = b \supset Q(X,Y)))\}$$

The query $\forall X \exists Y.(P(X,Y) \vee Q(X,Y))$ would be answered by $X$ as known to be true. The same answer is gained when using our implementation, because the variable $X$ has an empty equality term $(X|1|*|*)$ so that the new constant matches. Additionally, there exists an constant that substitutes $Y$ and fulfills the predicate.
Note that when the two predicates in the query would be connected by an conjunction the query would be answered with "unknown" even while the two predicates from the knowledge base are single unit terms, because then the existential would link the second arguments of each of the predicates with each other and would require them to be equal. The query $\forall X \exists Y.(P(X,Y) \wedge Q(X,Y))$ would be answered by $X$ as unknown. The same answer is gained when using our implementation, because then the existential would link the second arguments of each of the predicates with each other and would require them to be equal.
We implement the mentioned combinations of quantifiers while we make use of the results gained in the last section concerning the $\forall$-quantifier and we provide that existential variables that are used in predicates connected by conjunctions are not mutually exclusive.

## 4.6 Reasoning by Cases

### 4.6.1 Introduction

Since we introduced the idea of reasoning by cases in the second chapter already, we now describe how we implemented reasoning by cases in our approach.
As said in the second chapter the main problem with reasoning by cases is to determine the next clause to choose. There is no criterion that would allow us to make the "right" choice directly or at least it would be too time consuming to precompute the most useful clause. The only thing we can do is to restrict the search space slightly as described in the following section.

Recall, that in our implementation reasoning by cases is restricted to a user-defined level and does not depend on the structure of the query as suggested in *X*.

We will first of all present the criterion that we use to restrict the set of possible clauses that can be used at a specific application of reasoning by cases. At the same time we will see that there are various criterion that filter out specific possibilities, but can not be generally applied.
Additionally, we will discuss the fact that it is *not* possible to preprocess the knowledge base to gain better results concerning the set of possible clauses when more than one level of reasoning by cases is of concern. We will discuss this topic also in the section preprocessing.

Thereafter we present the algorithm that implements reasoning by cases in our approach.

### 4.6.2 The Criterion of Reasoning by Cases

In our implementation reasoning by cases is applied while making use of the following criterion.
Before we present the criterion first note that clauses are connected to each other if they share the same predicate. We use the absolute value of a predicate, hence the clauses $(P(X) \vee Q(x))$ and $(\neg P(X) \vee Q(x))$ are connected by $P$ and $Q$. Suppose the query would consist of the predicate $P$ only, then the clause $(\neg Q(X) \vee R(X) \vee S(X))$ would be *indirectly* connected to the query. The first two clauses are directly connected to the query.
The criterion allows us to pre-compute a conservative estimate of the disjunctive terms that are connected with each other [42]. This set is used when reasoning by cases is applied.

**Criterion** *Reasoning by Cases*
The set of clauses that will be used for reasoning by cases at any level will only contain the clauses that are connected by predicates directly or indirectly to the predicates of the query.

Note that this criterion requires the knowledge base to be consistent in the context of the reasoning procedure of concern. Hence, our first step when applying reasoning by cases is to determine all $proper^+$ terms in the knowledge base that contain the predicate from the query. Secondly, we determine all of the disjunctive terms that are connected to the first set of $proper^+$ terms by predicate. Note that this also includes clauses that are not directly related to one of the disjunctive terms of the first set. The following procedure represents the method how the possible set of clauses for reasoning by cases is determined.

---

**Initialize:**

- Determine all predicates from the query and store them in the variable $PredicateIDs_{new}$
- $PredicateIDs_{used}$, $PredicateIDs_{temp}$, and $TermIDs_{RbC}$ are empty variables

**Repeat**

- Identify all disjunctive terms that contain a predicate that is in $PredicateIDs_{new}$ or its complementary predicate is contained in $PredicateIDs_{new}$
- Store the identified terms in $TermIDs_{RbC}$ if they are not already contained
- Set $PredicateIDs_{used}$=$PredicateIDs_{new}$ and delete all predicates from $PredicateIDs_{new}$
- Determine all predicates contained in the newly identified terms and store them in $PredicateIDs_{temp}$
- Select only newly determined predicates by comparing the predicates stored in $PredicateIDs_{temp}$ and $PredicateIDs_{used}$ and store them in $PredicateIDs_{new}$

**Until** $PredicateIDs_{new}$ **contains no predicates**

---

After the application of this procedure the variable $TermIDs_{RbC}$ contains all disjunctive terms that are of concern when reasoning by cases is applied.

The procedure is initialized by storing the predicates contained in the query in $PredicateIDs_{new}$. The predicates from the query are used to determine the *directly* connected disjunctive terms. All further recursions of the procedure identify the *indirectly* connected disjunctive terms.
As the reader can observe we search for a predicate and the complement of a predicate at the same time. This proceeding is motivated by the fact that we can accomplish this task in one single SQL-Statement:

SELECT * FROM pTerm-pred WHERE predicateId IN ($PredicateIDs_{new}$)

The indirect way would look for the actual predicates only and then look in the following cycle only for the complementary version of the predicate. In the end both approaches will return the same set of disjunctive terms.
We use the variable $PredicateIDs_{used}$ to keep track of the predicates that have been already used to determine disjunctive terms. Then, it is possible to select only new predicates in the newly discovered disjunctive terms. At the same time this will terminate the procedure after all disjunctive terms for reasoning by cases are determined.

In Prolog this algorithm could be implemented by the following snippet of code:

```
% Detemine the terms that will be used by reasoning by cases
get_Clauses([],_,[]).
%
% PIDList holds the predicate identifiers (initial: Query)
get_Clauses(PIDList, PrevPIDList, [ReturnTIDs|ReturnTIDList]) :-
% create the list of the complementary predicates
maplist(times(-1),PIDList,InversePIDList),
append(PIDList,InversePIDList,CompletePIDList),
% get all corresponding disjunctive terms
getall_DisjunctiveTermIDs(CompletePIDList, ReturnTIDs),
% get all predicates that are contained in ReturnTermIDs
getall_PredicateIDs(ReturnTIDs, ActualPIDList);
% add to visited predicateIDsList
append(CompletePIDList,PrevPIDList, VisitedPIDList),
% delete all previous used predicateIDs from the current PIDlist
subtract(ActualPIDList, VisitedPIDList, NewPIDList),
% start the next cycle with the new predicates
get_Clauses(NewPIDList, VisitedPIDList, ReturnTIDList).
```

As said earlier we have to determine the set of clauses that will be used by reasoning by cases online. Consequently, the algorithm must be fast. In fact this is the reason why we can not take care of equalities for example. It is obvious that the observation of equality terms would restrict the set of possible clauses in a more *effective* way.
But we only use a predicate and its complement version to explore the connections among different clauses, because the analysis of the equality terms would be too time consuming.
There exist also other methods to restrict the set of clauses, but as far as we know they can not be applied efficiently. We will discuss some methods briefly in a later section. At this point we have chosen in favor of a large set of possible clauses and a fast method of determination.

Note that when all the clauses in a knowledge base are connected directly or indirectly to a given query *all* clauses of the entire knowledge base must be considered when reasoning by cases is applied. It is obvious that those kind of knowledge bases can not be handled efficiently with our implementation. A better criterion would improve this, but we think that those kind of knowledge bases can not be handled efficiently in general. Additionally, those knowledge bases do not belong to the field of application of concern.

We can not preprocess the set of possible clauses since we allow more than one level of reasoning by cases. At the first level of reasoning by cases a single predicate is added to the knowledge base and this might have essential effects on it. And since a subsequent level in reasoning by cases uses the knowledge base in the status the previous levels have changed the original knowledge base there is no way to preprocess the set of clauses. Although we will purpose a method to preprocess the knowledge base in a later section. But this method can only be used when exactly one level of reasoning by cases is of concern.

Finally, we presented a method in this section to precompute a conservative estimate of the set of clauses to be used by reasoning by cases in advance.

### 4.6.3 Implementation

In this section we present our implementation of reasoning by cases. Since unit propagation is a part of reasoning by cases the algorithm includes the two main features introduced by $X$. Therefore, the algorithm described here shows how we answer a query in general. The testing of a query is accomplished as described in an earlier section. In the following algorithm we will make use of those variables:

- $RbCLevel$: actual reasoning by cases level, the initial level is $-1$
- $MaxRbCLevel$: maximum reasoning by cases level (user-defined)
- $RbCTerms_i$: Set that contains the term identifiers that are of concern at the reasoning by cases level $i$

---

1. Test the query while using the actual knowledge base
2. If the test did not succeed for any of the queries apply reasoning by cases:
   (a) Increment the actual $RbCLevel$
   (b) If $RbCLevel > MaxRbCLevel$ then fail
   (c) Determine the set of clauses that will be used for reasoning by cases (as described in the section before), delete already used terms (marked) from the set and store it in $RbCTerms_{RbCLevel}$
   (d) Until not every term in $RbCTerms_{RbCLevel}$ is visited or not return 1 do the following:
      i. Add the actual predicate of the current term to the KB as unit term. If there is more than one equality term use one equality term that was not used before. Note that only one single ground instance can be added at once. Mark the entire term as used.
      ii. Apply *Unit Propagation*
      iii. Go to step 1. (test if query is implied by the KB and apply reasoning by cases again if necessary and possible)
      iv. If iii) succeeds (returns 1) proceed to the next predicate in the actual term and go to step a), else fail and if there are unused equality terms go to i) and else go to the next term in $RbCTerms_{RbCLevel}$ and undo all changes (e.g. unmark used terms, delete added unit terms and all changes done in the knowledge base) caused by this $RbCLevel$
      v. If step iv) is successful for every predicate of one term, then return 1
   (e) If step e) does not succeed for any of the terms in $RbCTerms_i$ then return 0
3. If the test succeeds for the query then return 1

Note that we will provide a detailed example in the following section. Since those steps sketch the algorithm briefly we go through it step by step now.
In step 1.) we apply the test that we have introduced in an earlier section. It checks if the given query is supported by the actual knowledge base or not.

If the query is not known to be true we apply reasoning by cases. This causes the increase of the actual RbC-Level which is simply necessary to be able to limit reasoning by cases by the user-defined maximum (see 2.a) and 2.b)). Note that we start with the initial RbC-Level $-1$.

In the section 2.c) we determine the set of clauses that will be of concern when we apply reasoning by cases. At this point we use the method presented in the section before. Every clause that is connected directly or indirectly by predicates to the predicates contained in the query will be included in the set $RbCTerms_{RbCLevel}$.
We prevent cycles in reasoning by cases simply by marking the term that has been used for the actual application of reasoning by cases. As can be seen in 2.d)iv) we unmark the term that was marked at the actual level again if we proceed to the next term.

The entire section 2.d) contains the essence of reasoning by cases. We take the first term from the previous generated set of term identifiers. Now we add the first predicate and the corresponding equality term in this chosen term to the database. In fact this is of course a unit term (2.d)i)).
At this point it is important to mention that we only add one single ground instance to the knowledge base. For example, suppose the chosen term to be the following one:

$$(X = a) \vee (X = b) \supset P(X) \vee Q(X)$$

Then we add at first the unit term $(X = a \supset P(X))$. If reasoning by cases does not succeed then, we do not proceed to the following term, but first try the other equality $X = b$. In general, we first add every possible ground instance of the actual term before proceeding to the next possible $proper^+$ term. We add a predicate that contains unrestricted variables by using equalities that are used by other $proper^+$ terms in the knowledge base and contain the identical predicate. If this does not succeed we use a "don't care"-symbol to generate a ground instance and keep track of an assignment to this variable so that the variable can have only *one* specific value during the actual reasoning process. Note that the variable might be assigned to a constant contained in the query, so that we take care of constants that are not used in the knowledge base but in the query only. This approach is similar to the implementation of the $\forall$-quantifier. It prevents the substitution of every

possible constant included in the entire knowledge base. If a $proper^+$ contains no unrestricted variables at all, we simply add each possible disjunctive equality term one by one if necessary.

Since we have a new unit term in our database we apply unit propagation again as described in some earlier section. This may cause that some other unit terms are created and our query could now be known to be true. This is why we test the query again (see 2.e)iii.)).
The return to step 1.) is in fact a recursion. If the test succeeds we go to the next predicate in the actual term. But if the test fails we apply reasoning by cases *again*. But now not on the original database but on the database that includes the changes from the previous reasoning by cases level. The depth of recursion is restricted by the maximum reasoning by cases level (2.b)).
If it is not possible - even with the highest reasoning by cases level - then we leave the actual term, undo all changes caused by the actual RbC-Level, go back one level and try all other possible terms at this level.
This proceeding is repeated until we again reach level zero. Then we proceed to the next term in the set of clause at RbC-Level 0, restart the entire process and proceed until there are no more possible terms at RbC-Level 0. In fact this proceeding is commonly known as backtracking.
Note that every single different term may additionally have several ground instances due to multiple equalities what of course causes an additional complexity.
At this point it should be obvious that the application of reasoning by cases is a complex process since there are not only possibilities in reasoning introduced by the set of clauses fulfilling the criterion, but additionally by different equalities.
Consequently, the reader should be aware of the fact that high levels of reasoning by cases can not be applied efficiently. And even small level of reasoning by cases may cause long answering times since the number of possibilities depends also on the trait of the knowledge base.
Think for example of a knowledge base where *every* clause is directly or indirectly connected to a given query. Then all clauses contained in the entire knowledge are of concern when reasoning by cases is applied. But this is a topic of the following chapter.

Undoing all changes for example includes deleting added unit terms and unmark the term that was involved in this actual application of reasoning by cases at the current level.
Note that we only undo changes applied by the actual level and not all changes. If we go back from RbC-Level 1 to 0 the database is again in its original state.

Be aware of the fact that we have to consider all possibilities at *each* level since we will test *all* of the terms that are actual in the current level (we again apply 2.d)). This includes that we sometimes go just one level back in reasoning by cases, go to the next term, test it and increment the level again.

Note that when we return to step 1. in 2.d)iii) we support the strategy of "depth-first" when reasoning by cases. This means that we go to the maximum RbC-Level each time when we add a new unit term (a single predicate) and the query is not tested successfully at any level before.
If we reached the maximum level and the query is not known to be true although we tried every possible term at each level we proceed to the next term in the set $RbCTerms_0$.
We have chosen the strategy of "depth-first" because of two main reasons:

- If we would use "breadth-first" we would always have to recreate the data that was achieved in the reasoning by cases levels before
- We assume only very small maximum RbC-Levels (normally 1 or 2)

We think that an improvement concerning the implementation of this part of reasoning by cases would be to use the strategy of breadth-first and a data structure that supports to keep data of different terms and levels of reasoning by cases distinct.
This approach would of course be more space consuming than our current implementation, but since the amount of generated clauses could be handled by a database and this approach would be more effective and efficient, we think that it would be at least useful when you would like to support higher maximum values for reasoning by cases than two or three.

If all predicates of one term of the first identified set turn out to support the query by applying reasoning by cases once or as often as required and allowed the query is known to be true.

In contrast, if we do not find any term that supports the query with all of its predicates at any allowed level of reasoning by cases the query is unknown (2.f)).

The step 3.) is used as direct return value when we do not need to apply reasoning by cases at all (step 1. succeeds) or as return value in one of the recursive calls.
In general, "return 1" as final return value states *known to be true* as "return 0" denotes *unknown*.

The presented algorithm implements the main part of $X$ since it includes the two main features reasoning by cases and unit propagation. Every kind of query will be answered by this algorithm.

Of course there are some features of $X$ left, but this algorithm belongs to the essential parts of the implementation.

## 4.7 Preprocessing of the Knowledge Base

While we already introduced all main features of our implementation we now present which kind of preprocessing takes place before any query is answered. As mentioned before preprocessing especially regards to the encoding of a given $proper^+$ knowledge base and the application of unit propagation.
In the following we will discuss both topics and we will additionally briefly discuss methods that could be used to enable a fast processing when we apply reasoning by cases.

First of all we have to encode a given $proper^+$ knowledge base. Actually this is done as described in the corresponding section. Note that this not only includes the encoding of each term of a $proper^+$ knowledge base, but also the encoding of predicates and constants. Since this is done during preprocessing we are not tied to the bounds of efficiency and therefore it is for example no problem to convert every equality term into DNF.
However, at this point we would like to mention a fact that we will discuss also in a later section. At the moment there exist no $proper^+$ knowledge bases at all. Hence, it could be even possible that knowledge bases are directly created in a given format (like our suggested one), so that nearly no encoding has to take place.
While this could be an advantage caused by the fact that there exist no $proper^+$ knowledge bases until now, the fact also causes a major problem: we do not have any opportunity to test our approach. But this topic will be of concern later on.

The most important feature of our preprocessing is the application of unit propagation on the entire knowledge base. This is of such importance, because it allows us to have a large number of unit terms in our knowledge base.
This is due to the fact that after we tested every unit term with every disjunctive term during preprocessing we will never again have to consider these unit terms when we apply unit propagation later on.
If a unit term does not succeed on a disjunctive term it will not succeed at any later test. If a unit term can be applied the resulting disjunctive or unit term is stored in the knowledge base and there is no need to redo this application.
Of course we will need to access the entire set of unit terms, but we do not have to consider all of the possible unit terms when we apply reasoning by

cases. Then we only have to take care of the *new* generated terms in the current reasoning process.
In fact, this is the reason why we have chosen three distinct tables that hold disjunctive terms, "old" unit terms and new unit terms. Preprocessing allows us to have a large number of unit terms in our original knowledge base since it does affect our reasoning procedure only slightly as we show in the next chapter.
As said before we need to search the table of unit terms during every reasoning process, but since we restrict ourself to have maximally $10^6$ unit terms this can be handled efficiently by the MySQL-database (see next chapter).

And since unit propagation is applied during preprocessing there exist queries that can be directly answered. Especially queries that would require simple applications of *Modus Ponens* can be answered immediately.
Suppose the following $proper^+$ terms contained in the original knowledge base:

$$P(X),\ \neg S(X),\ \neg P(X) \vee Q(X),\ \neg Q(X) \vee R(X) \vee S(X)$$

After encoding these terms and preprocessing the knowledge base the following result are achieved:

$$P(X),\ \neg S(X),\ \neg P(X) \vee Q(X),\ \neg Q(X) \vee R(X) \vee S(X),\ Q(X),\ R(X)$$

Hence, if we now ask $\exists X.R(X)$ we can directly answer that the query is known to be true, since the predicate is contained in the preprocessed knowledge base as unit term.
Consequently, queries that require the application of *Modus Ponens* can be answered instantly and therefore can be answered very efficiently (see next chapter).
Note that the application of unit propagation will cause the original knowledge base to *grow*. We will discuss this topic in the subsequent chapter.

Another kind of preprocessing could allow us to apply reasoning by cases efficiently at least if we restrict the level of reasoning by cases to be maximally 1.
For example, if you determine the clauses that are directly or indirectly linked to the query and this set does not contain any clause that has exactly two predicates there must be a clause that holds exactly the complement of the predicates contained in the query.
This is due to the fact, that further chaining in reasoning caused by unit propagation can only occur, when there exist clauses that contain exactly two predicates. Hence, if they do not exist and we allow only one level of reasoning by cases there must exist a clause that holds the inverted predicates of the query. Otherwise it is not possible that reasoning by cases succeeds.

Since this is a rather simple criterion you might wonder why we did not include it in our approach. The reason why we did not include this criterion among other possible criterion is that we support more than one level of reasoning by cases in general.
And then the connections between the clauses and the possibilities in chaining increases in a way that they can not be handled as efficient as with the criterion that we used. As said before we decided in favor of a criterion that can be applied efficiently.

All in all preprocessing in our implementation creates the foundation of our approach by encoding a given $proper^+$ knowledge base and allows us to answer queries that would only require the application of unit propagation instantly. In fact, one main method of reasoning introduced by $X$ is applied after we preprocessed the given knowledge base. Furthermore preprocessing and the chosen data structure enable us to handle round about $10^6$ unit terms without any major drawbacks concerning efficiency.
Additionally, we think that during preprocessing other tasks like a preparation for a later use of reasoning by cases could be applied to increase efficiency for special kind of queries.

## 4.8 Worst-Case Complexity

In this section we discuss the worst-case complexity of the presented implementation in terms of the number of applied unit propagations. In the following we assume that a single application of unit propagation requires linear time complexity in the number of $proper^+$ terms in total. This assumption is for instance based on the fact that we can assign a maximal length to a single $proper^+$ term and this length is very small compared to the size of the entire knowledge base. Additionally, in the propositional case [73] presents an algorithm with linear time complexity. In addition, we assume that every term is of concern when reasoning by cases is applied. Furthermore, we use an *e-free* KB $\{\forall(c)\}$ (see Chapter 3) during our observations. Recall, that this representation is equivalent to $proper^+$ knowledge bases $\{\forall(e \supset c)\}$ when $e$ in DNF and $e$ contains no inequalities.
We will use the following parameters:

$n = |KB| = |\mathcal{C}_{\mathcal{KB}}|$
$l =$ Maximal number of predicates contained in a disjunctive term
$k =$ Maximal number of variables in a term
$RbC_{Level} =$ Maximally allowed level of reasoning by cases

Now we determine the number of applied unit propagations to approximate the worst-case complexity. Thereby, we make use of Theorem 6 from Chapter 3. As it was shown in Chapter 3 an e-free knowledge base with $n$ terms results under the closure of unit propagation in a knowledge base whose size is maximally $n^{k+1}$ while $k$ denotes the maximal number of variables in each of the terms.
Therefore, we can assume that every single added unit term during reasoning by cases at the first level can only cause less than $n^{k+1}$ applications of unit propagation. Note that this requires besides other properties that every term is directly or indirectly connected to every other term in the knowledge base. Since there are $n$ terms and every term has maximally $l$ predicates and we approximate the number of argument values of a single predicate by the maximal number of variables contained in a term ($k$) we can not add more than about $n * n^k$ unit terms while we neglect $l$ since $l << n$. Mainly, this is due to the fact that every variable can be substituted by $n$ constants since $|\mathcal{C}| = n$. Recall, that we can only add one single ground instance during an application of reasoning by cases.
In total, we apply unit propagation $n^{2k+2}$-times at the first level of reasoning by cases.
Note that we neglect the time that is used to choose a term, to add an appropriate ground instance and to undo all changes when going back one level of reasoning by cases.

If we set $RbC_{Level} = 2$ we try for every added unit term at the first level of reasoning by cases to succeed while adding one other possible unit term which may cause again about $n^{k+1}$ applications of unit propagation for each added term. Note that we can again add about $n^{k+1}$ unit terms.
In general, the worst-case complexity is:

$$O((n^{2k+2})^{RbC_{Level}})$$

Note that the worst-case complexity is exponential in the number of the applied level of reasoning by cases, but not in the size of the knowledge base. Additionally, note that $k$ is a very small constant compared to $n$.

Since we only discussed the worst-case complexity until now, please note that the number of possible substitutions is much smaller and chaining in reasoning takes place only two or three times in the practical case. As mentioned before, we assumed in the discussion of the worst-case scenario that every term is connected to every other term like it is common in a SAT instance. But this is not the field of application of this reasoning procedure used here.

| *Example Knowledge Base* |
|---|
| $(X = a) \supset Q(X) \vee P(X, Y)$ |
| $(Y = d) \supset R(X) \vee P(X, Y)$ |
| $(X = a \wedge Y = d) \supset S(X) \vee P(X, Y)$ |
| $(X = a) \supset \neg Q(X) \vee \neg R(X) \vee \neg S(X)$ |

**Table 4.14:** An example knowledge base where all of the features of the introduced approach will be applied on

However, the complexity of the algorithm will be exponential in the user-defined level of reasoning by cases even with the just made assumption for the practical case. This is a consequence of the fact that high levels of reasoning by cases cause reasoning to get close to classical logical entailment which is intractable in general. Therefore we suggest small levels of reasoning by cases. As we will see in the following Chapter, the answering time of the algorithm also depends on the number of terms that are of concern during an application of reasoning by cases which corresponds to the observations made in the worst-case scenario.

## 4.9 A detailed Example

Since we described separately how $proper^+$ terms are encoded, unit propagation and reasoning by cases are applied, and how queries are evaluated in our implementation, we now turn to a detailed example to clarify how different pieces of the introduced approach fit together.
Note that when we apply unit propagation we will not denote every created disjunctive term to enable a better readability.

Suppose the example knowledge base as depicted in table 4.14.

Of course we first of all have to encode the knowledge base into the data structure on which our algorithm works on. The encoding requires two tables in essence as said before. Table 4.15 and 4.16 hold the corresponding encoding while the encoding of the single predicates is not shown; they are simply mapped to the numbers 1 to 4 according to their alphabetical order.

Preprocessing of the knowledge base will leave the $KB$ unaffected since there are no unit terms in the $KB$ at all. Therefore, the table "pTermpredU1" will contain no entries. Recall that preprocessing only applies to unit propagation.

Now we would like to answer the following query while we allow two steps of reasoning by cases

| *termId* | *predicateId* | *variables* | *rbcLevel* | *oldrbcLevel* | *new* |
|---|---|---|---|---|---|
| 1 | 2 | $X$ | 0 | 0 | 1 |
| 1 | 1 | $X, Y$ | 0 | 0 | 1 |
| 2 | 3 | $X$ | 0 | 0 | 1 |
| 2 | 1 | $X, Y$ | 0 | 0 | 1 |
| 3 | 4 | $X$ | 0 | 0 | 1 |
| 3 | 1 | $X, Y$ | 0 | 0 | 1 |
| 4 | $-2$ | $X$ | 0 | 0 | 1 |
| 4 | $-3$ | $X$ | 0 | 0 | 1 |
| 4 | $-4$ | $X$ | 0 | 0 | 1 |

**Table 4.15:** The table 'pTerm-pred' holding one part of the example knowledge base

| *termId* | *equalityId* | *equalities* | *updated* | *rbcLevel* |
|---|---|---|---|---|
| 1 | 1 | $X\|1\|1\|a - Y\|2\| * \|*$ | 0 | 0 |
| 2 | 1 | $X\|1\| * \| * - Y\|2\|1\|d$ | 0 | 0 |
| 3 | 1 | $X\|1\|1\|a - Y\|2\|1\|d$ | 0 | 0 |
| 4 | 1 | $X\|1\|1\|a$ | 0 | 0 |
| 4 | 2 | $X\|1\|1\|b$ | 0 | 0 |

**Table 4.16:** The table 'pTerm-equal' depicts the equalities of the example knowledge base

$$\exists X, Y.P(X, Y)$$

At this point we would like to mention again that $X$ in its standard definition does not support reasoning by cases at any higher level than one. The extension implemented here is founded on [37] as said before.

First the algorithm searches the table 'pTerm-predU1' if there is an unit term with the 'predicateId=1'. While this is not successful since the table is empty, the algorithm makes use of reasoning by cases, because it is allowed to by the user.

At this point we use the method to determine the possible set of disjunctive terms that can be used by reasoning by cases while we use the presented criterion. In our case this are the terms with the identifiers $1, 2, 3$ and $4$, since all clauses are directly or indirectly connected to the query which only contains the predicate $P$. In our example the fourth disjunctive term is indirectly linked to the query while all other terms are directly linked.
The algorithm always chooses the clause with the smallest term identifier ('termID') and so $(X = a) \supset Q(X)$ is added at first to the knowledge base, namely to the table "pTerm-predU2" (see table 4.17).

Note again that we do not support inequalities at this point. Suppose that the actual equality from the example would be not $(X = a)$ but $(X \neq a)$. Then the set of possible ground instances would be immense and it is not trivial to decide which of them to choose. Hence, there would exist too many possibilities of ground instances that could be added to the knowledge base and therefore this could not be implemented efficiently.
Please note that we add this clause to the table "pTerm-predU2". Only unit terms contained in the table "pTerm-predU2" are considered when unit propagation is applied within reasoning by cases. In addition the table 'pTerm-equal' is affected of course, but we will not show the changes made in that table here.

Now the algorithm applies unit propagation again what obviously affects $(X = a \vee X = b) \supset \neg Q(X) \vee \neg R(X) \vee \neg S(X)$. Consequently, the resulting term $(X = a) \supset \neg R(X) \vee \neg S(X)$ is added to the table that contains the disjunctive terms only (table 4.18).
At this point the algorithm will check again if there exists a unit term with the "predicateId=1" in the knowledge base. Again, this is not successful and since the algorithm is allowed to increase the level of reasoning by cases once again, it will add $R(X)$ as unit term next and apply unit propagation.

At this stage the application of unit propagation results in a new disjunctive term and a new unit term, namely:

$$X = a \supset \neg Q(X) \vee \neg S(X)$$
$$X = a \supset \neg S(X)$$

| *termId* | *predicateId* | *variables* | *rbcLevel* | *oldrbcLevel* | *new* |
|---|---|---|---|---|---|
| 5 | 2 | $X$ | 1 | 1 | 1 |

**Table 4.17:** The table 'pTerm-predU2' after adding the first predicate $Q(X)$ of the chosen clause in the beginning of reasoning by cases

Consequently, the execution of unit propagation proceeds since this unit term can be used to affect the term $(X = a \wedge Y = d) \supset S(X) \vee P(X, Y)$ for example and results again in a new unit term, namely $(X = a \wedge Y = d) \supset P(X, Y)$. Note that also other disjunctive terms are effected.

And now the algorithm will check again if there exists a unit term with the 'predicateId=1' in the table "pTerm-predU2" (see table 4.19).
This time this will succeed and since we do not need to check equalities while we only search for the existence of $P(X, Y)$, we proceed to the next and last predicate in the actual clause.
Hence, $(X = a \wedge Y = d) \supset P(X, Y))$ is added to the $KB$ and this of course implies that $\exists X, Y.P(X, Y)$ holds, since we only want to determine if $P$ exists with any arbitrary assignment of the variables $X$ and $Y$.

At this point we showed that $P(X, Y)$ holds when adding the first predicate of the first chosen clause. Furthermore, we have to check the second predicate of the clause, namely $(X = a) \supset P(X, Y)$; it is obvious that the addition of a corresponding ground instance will satisfy the query.
Accordingly, the algorithm answers that $\exists X, Y.P(X, Y)$ is known to be true.

Note that the use of our depth-first strategy and the allowance of two levels of reasoning by cases prevented the use of the fourth term. If we would restrict reasoning by cases to only one level the algorithm would still answer known to be true.
This is due to the fact that after trying every other clause without success the last clause supports the query by one single application of reasoning by cases.

## 4.10 Summary

In this chapter we presented an implementation of all main features introduced by $X$. We started the discussion of our implementation by restricting $proper^+$ terms to contain no inequalities for several reasons.
This restriction has major effects on the evaluation of a query and the application of reasoning by cases. For example quantifiers could be solved very efficiently and within reasoning by cases we can easily add single ground in-

| *termId* | *predicateId* | *variables* | *rbcLevel* | *oldrbcLevel* | *new* |
|---|---|---|---|---|---|
| 1 | 2 | $X$ | 0 | 0 | 0 |
| 1 | 1 | $X, Y$ | 0 | 0 | 0 |
| 2 | 3 | $X$ | 0 | 0 | 0 |
| 2 | 1 | $X, Y$ | 0 | 0 | 0 |
| 3 | 4 | $X$ | 0 | 0 | 0 |
| 3 | 1 | $X, Y$ | 0 | 0 | 0 |
| 4 | $-2$ | $X$ | 0 | 0 | 0 |
| 4 | $-3$ | $X$ | 0 | 0 | 0 |
| 4 | $-4$ | $X$ | 0 | 0 | 0 |
| 6 | $-3$ | $X$ | 1 | 1 | 1 |
| 6 | $-4$ | $X$ | 1 | 1 | 1 |

**Table 4.18:** The table 'pTerm-pred' after adding the unit term $(X = a) \supset Q(X)$ to the table 'pTerm-predU2' and applying unit propagation at the first level of reasoning by cases. Note the new disjunctive term.

| *termId* | *predicateId* | *variables* | *rbcLevel* | *oldrbcLevel* | *new* |
|---|---|---|---|---|---|
| 5 | 2 | $X$ | 1 | 1 | 0 |
| 7 | 3 | $X$ | 2 | 2 | 1 |
| 8 | $-4$ | $X$ | 2 | 2 | 1 |
| 9 | 1 | $X, Y$ | 2 | 2 | 1 |

**Table 4.19:** The table 'pTerm-predU2' after adding the unit term $R(X)$ and applying unit propagation at the second level of reasoning by cases

stances. Both aspects would not be possible with inequalities included.
Additionally, we could not think of any practical use of inequalities with regard to single domain constants, except of the use in a query. And since we allow inequalities to be contained in a query there are no practical restrictions caused.

While we presented the algorithms corresponding to unit propagation and reasoning by cases the usefulness of our encoding was clarified. Apart from the way we encode $proper^+$ terms the use of database features was essential, e.g.: fast search in large datasets and index join.

The application of reasoning by cases is as mentioned a complex process. We chose for a criterion that can be applied very fast. Hence, it is possible to efficiently determine the set of clauses that is of concern when reasoning by cases is applied.
The number of possible combinations of clauses caused by higher levels of reasoning by cases and different equality terms corresponding to one single $proper^+$ term is immense and therefore hard to implement efficiently. Therefore we suggested very small levels of reasoning by cases ($\leq 2$) and additionally $proper^+$ terms should not contain many disjunctive equality terms. This topic will be of concern in the next chapter.

Furthermore we showed the features that preprocessing introduces and which implications preprocessing has on the reasoning procedure itself. For instance, it allows to instantly answer queries that require simple applications of *Modus Ponens*.
Even more important for the case of reasoning by cases is the fact that preprocessing allows us to restrict the set of possible unit propagation pairs in a crucial way. This is due to the fact that all original unit terms have been considered already after preprocessing is applied.

The main contribution of our work is that all main features of $X$ can be handled by our implementation while allowing large datasets. We additionally support an efficient answering of queries based on the following features: unit propagation by employing database features, the efficient handling of quantifiers, and preprocessing of the knowledge base.

# Chapter 5

# Efficiency

## 5.1 Introduction

We begin this chapter by introducing a major problem concerning the generation of a test knowledge base.
The problem is that there do not exist any $proper^+$ knowledge bases at all. And as said before we think that there is a use for large first-order knowledge bases in the field of artificial intelligence, but until now they are not existent [36].
Consequently, we had no opportunity to take a given knowledge base and test our approach or to compare our results with other approaches. In contrast to the worldwide SAT competition [57] that involves thousands of competitors and test instances, large first-order knowledge base as suggested here are not of such concern.

Therefore, we cannot present as significant and precise test results as they are common in the context of the propositional case. We will discuss for example the relations between the size of the knowledge and the answering time, and answering times that are caused by queries that do not make use of reasoning by cases.
Additionally, we will see that the answering time depends immensely on the *characteristics* of the $proper^+$ terms contained in the knowledge base when reasoning by cases is applied.

A simple example of this fact is a large set of terms that is directly or indirectly connected to a predicate from the query. Then the set of clauses that will be used when reasoning by cases is applied is also large and consequently there exists a high number of possibilities for our implementation of reasoning by cases (see Chapter 4). Hence, the answering time will increase in a non-reasonable way.

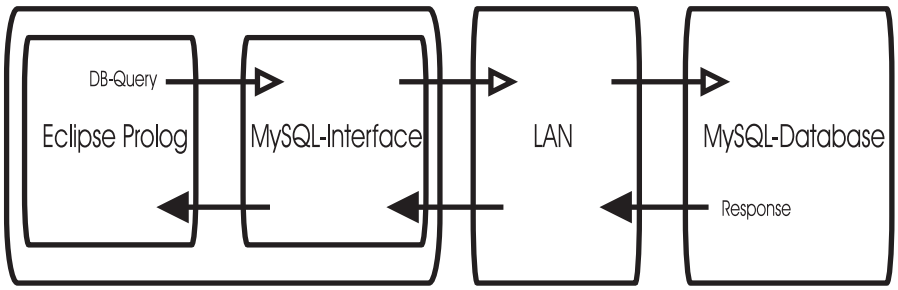


**Figure 5.1:** A brief overview on the flow of data when a query is send from $ECL^iPS^e$ PROLOG to the database and the answer to the query is send back.

Comprising, we will discuss how we generated a test knowledge base and which factors increase the answering time of the presented implementation and which circumstances support an efficient evaluation of the query.

## 5.2 Environment

We use a $1,7$ GHz dual processor system (512 MB RAM) as database server. The client ($1,7$ GHz, 128 MB RAM) is connected to the server by a 100 MBit/s local area network (LAN).

Furthermore, we use the MySQL version 4.01 [53] and the version 5.5 of $ECL^iPS^e$ PROLOG [21]. The used version of MySQL is public domain and can be downloaded freely at the given web address in the bibliography [53]. $ECL^iPS^e$ PROLOG is not public domain, but is available for free to universities and non-profit research institutions.

Additionally, we use a MySQL-interface to handle the database queries, which is available at [21] and was created by [32]. In fact the MySQL-interface consists of a C-interface to MySQL which is integrated in $ECL^iPS^e$ PROLOG.

The flow of data between $ECL^iPS^e$ PROLOG and the database is depicted in Figure 5.1. As can be seen in this figure a single query originating in PROLOG is first send to the MySQL-Interface, second to the LAN and finally to the MySQL-Database. The answer to the query has to take the same way back. Note that these costs introduced by the PROLOG/C-Interface, the MySQL-Interface and the client/server configuration are included in the answering times presented later on.

## 5.3 The Test Knowledge Base

As said before there do not exist any first-order knowledge bases that are of the size and type as suggested in our work [36].

Therefore we have to generate a test knowledge base on our own. As we know from the introduction it is even very hard to generate appropriate propositional instances. For example, instances created by random can be solved very efficiently with high probability. Consequently, these instances can not be used to test an approach in an appropriate way.
In the first-order case we discuss here, we have the additional problem that we need large datasets ($> 10^5$) to test our implementation. Note that the problem size in the propositional case consists of 1000 clauses maximally at most of the time [57].

Now suppose that you have to generate about $10^5$ $proper^+$ terms that are consistent and contain disjunctive terms that support queries like $\exists Y.(P(a,Y) \wedge Q(X,Y))$. Note that this also includes large set of constants and predicates.
It is of course possible to generate this set of terms by random, but the following two topics introduce major problems that have to be solved then:

1. Consistency of the entire knowledge base

2. Coherences in a set of terms

Recall, that we require consistency for our criterion of reasoning by cases.

After adding one new term the consistency of the whole knowledge base must be tested. Since we have more than $10^5$ terms the needed consistency test would be very complex and time consuming. For example, think of the following terms in the knowledge base:

$$(P(X))$$
$$(\neg P(b) \vee \neg Q(a)$$

Now suppose the next term to add would be the unit term $Q(a)$. Then we first had to apply unit propagation before we could determine that this term would cause the knowledge base to be inconsistent.

After generating this set of terms we still have to solve the second problem. Since the knowledge base is generated randomly we are not aware of the connections between the clauses. Consequently, it is hard to decide which query to ask.

The generation of an appropriate test knowledge base is not trivial and is not entirely solved in the propositional case. Especially, if a consistent knowledge base is generated successfully there is no information about the

*hardness* of this instance (see Introduction). Hence, it is not possible to determine how well an implementation works in general.

As shown in the first chapter many researchers are working on the satisfiability problem and therefore there exist well known test instances and results that can be used to test the efficiency of a new algorithm.
Since large first-order knowledge bases as we suggest them are very seldom in AI [36] we can not take advantage of any knowledge bases created before.

Our test knowledge base consists of terms generated in the following two ways:

- Unit terms are generated by random
- Disjunctive terms are clauses of SAT instances (from [57])

Since every unit term in our database holds a single predicate that is not contained anywhere else in this table we simply use the actual unique term identifier with some offset as predicate identifier. We choose a single equality term by random by using a single constant from a set of constants. Recall that predicates and constants are mapped to numbers.

We use clauses that contain three literals each from SAT instances to generate disjunctive terms. Hence, every created disjunctive term consists of three predicates and a equality term generated by random.

But *none* of these terms will be of concern directly when a query is answered, because we add incomplete information about individuals, rules and facts manually for every query we want to ask. In fact, we scatter the specific problem instances over the entire dataset and then ask the corresponding query.

In other words, we take a small set of $proper^+$ terms that have specific and known internal connections and add those terms to the entire knowledge base.

Comprising, we can not present a test knowledge base that can be used in its entire size. We simply generate a test knowledge base that consists of a large number of $proper^+$ terms, but only a small set will be of concern when answering a specific query.
Nevertheless, the entire set of terms must be considered to determine the corresponding terms of the actual problem instance.

## 5.4 Test Results

In this section we present the results of our approach applied on test knowledge bases of the type that was discussed in the previous section.

### 5.4.1 Preprocessing

Since preprocessing itself is done offline and is therefore not of concern when determining the efficiency of our approach we have no corresponding test results.

This is also due to the fact that the type of the generated knowledge base has nearly no connection between disjunctive terms and unit terms. Hence, the application of unit propagation would be without an effect at most of the time. Consequently, test results would not be representative. Note that the *result* of preprocessing will be of concern in the next section.

Furthermore, the encoding of $proper^+$ term is not applied since we generate terms in the suggested format directly.

At this point we only would like to discuss the following disadvantage caused by preprocessing. The growth of the knowledge base when unit propagation is applied (see Chapter 3) has a great impact during preprocessing since *every* disjunctive term and unit term is of concern when unit propagation is applied. While we discussed the theoretical case in Chapter 3 we now turn to the practical case and make several assumptions concerning the characteristics of the knowledge base.

Suppose that we have 10.000 disjunctive terms and 100.000 unit terms. Every disjunctive term contains two predicates. Then the growth of the knowledge base would be of minor concern, because every successfully applied unit propagation would generate a new *unit* term only. Consequently, only the number of unit terms would increase. This has no drawback to efficiency as we will see later on.

But if the disjunctive terms contain more than two predicates the number of disjunctive terms will increase in addition. For example, consider the following set of terms:

$$(P(X) \vee Q(X) \vee R(X))$$
$$\neg P(a),\ \neg Q(a)$$

After the application of unit propagation the set contains the following terms since we are not allowed to delete any disjunctive terms if they are not redundant (Chapter 3):

$$(P(X) \vee Q(X) \vee R(X))$$
$$(Q(a) \vee R(a))$$
$$(P(a) \vee R(a))$$
$$\neg P(a),\ \neg Q(a),\ R(a)$$

Note the growth in the number of disjunctive terms. When a disjunctive term contains more than two predicates than the number of disjunctive term

| | *2 Predicates* | *3 Predicates* | *more than 3* |
|---|---|---|---|
| *Number of Disjunctive Terms* | 80% | 15% | 5% |

**Table 5.1:** The number of disjunctive terms in the knowledge base with regard to the number of predicates contained

increases by the number of unit propagations successfully applied.
Additionally note that the number of unit propagations can be bigger than the number of predicates included in a term. For instance, the reader can observe this if we add the unit terms $\neg P(b)$ and $\neg Q(b)$ to the set of terms from above.

We assume that disjunctive terms that contain only two predicates are mainly used in the knowledge bases that are of concern here. Nevertheless, there remains a considerable growth of disjunctive terms.

Assume the percentages depicted in the Table 5.1 considering the number of predicates contained in the disjunctive terms in a knowledge base. If we additionally assume that on every disjunctive term that contains more than two predicates, unit propagation can be applied twice the number of terms will be multiplied by a factor about 3.
For example, if we have 10.000 disjunctive terms in total, then there are 2.000 terms that contain more than two predicates with regard to the assumption made. In the following we also assume that there are only three predicates in these disjunctive terms.

If we now apply unit propagation once on each of these terms the number of disjunctive terms doubles since the original disjunctive term remains and the new disjunctive term is added. Hence, we have now 12.000 disjunctive terms in total. When we apply unit propagation again the number of original terms is doubled again and 4.000 new unit term are generated in addition.

The generation of unit terms is due to the fact that the 2.000 disjunctive terms that were created by the first application of unit propagation contain only two predicates (see last example). And since we assume that we can apply unit propagation twice on each original disjunctive term there must be a predicate contained in the new generated disjunctive term where unit propagation can be applied on successfully.
Hence, a unit term is created since the new generated disjunctive term contains only two predicates. The same holds for the other generated disjunctive terms and consequently another 2.000 unit terms are generated.

In total we have 14.000 disjunctive terms after we have applied preprocessing. We will see in the next section that this kind of growth will not introduce any major drawbacks concerning efficiency even if the number of

| *Query* | *100.000 unit terms* | *1.000.000 unit terms* |
|---|---|---|
| $(X = a \wedge R(X))$ | $9msec$ | $10msec$ |
| $(X \neq a \wedge R(X))$ | $10msec$ | $10msec$ |
| $\exists X.R(X)$ | $9msec$ | $10msec$ |

**Table 5.2:** The answering times of the queries while using 100.000 unit terms in the first case and 1.000.000 in the second

disjunctive terms would double or triple.
Note that this discussion here is mainly based on assumptions and empirical results. Additionally, note that when we will refer to a number of terms to be contained in the knowledge base we always refer to the number of terms *after* preproccesing.
Additionally, we will discuss the positive results that are gained by preprocessing in the next section.

### 5.4.2 Answering Queries without Reasoning by Cases

We begin this section by discussing results that are achieved when no reasoning by cases is used. In the last chapter we mentioned that queries that require no reasoning by cases at all can be answered instantly due to preprocessing. These queries include cases that require simple applications of *Modus Ponens* (see previous chapter).
For instance, suppose the following set of terms to be contained in the knowledge base *before* preprocessing:

$$(X = a \vee X = b \supset \neg P(X) \vee Q(X))$$
$$(X = a \vee X = b \supset \neg Q(X) \vee R(X) \vee S(X))$$
$$P(X), \neg S(X)$$

After preprocessing the database holds the unit term $(X = a \vee X = b \supset R(X))$ among others.
The Table 5.2 holds sample queries and the corresponding answering times. The first test knowledge base contains 100.000 unit terms and the second 1.000.000 unit terms. Note that the number of disjunctive terms plays no role in this context.

Note the small differences between the answering times concerning the different queries and the different sizes of the knowledge base. All stated queries will be answered with known to be true. Most notably these results confirm that the handling of the existential quantifier is accomplished very efficiently by our implementation.

| *Query* | *10.000/100.000* | *100.000/1.000.000* |
|---|---|---|
| $P(a, Y)$ | $56msec$ | $61msec$ |
| $\forall X.\exists Y.P(X, Y)$ | $56msec$ | $61msec$ |
| $(P(X, Y) \wedge Q(X, Y))$ | $67msec$ | $74msec$ |
| $\forall X.\exists Y.(P(X, Y) \wedge Q(X, Y))$ | $68msec$ | $76msec$ |

**Table 5.3:** The answering times of the queries while using 10.000 disjunctive terms in the first case and 100.000 in the second.

The very small difference caused by the different number of unit terms in the knowledge base is due to the fact that databases can handle millions of datasets very efficiently [23]. Note that no reasoning at all takes place except the evaluation of the query.

In consequence these results confirm that queries that require simple applications of *Modus Ponens* or unit terms only can be answered efficiently. This is mainly due to the encoding scheme of $proper^+$ terms and preprocessing.

### 5.4.3 Answering Queries while using Reasoning by Cases

Suppose the following two terms to be contained in the knowledge base:

$$P(X, a) \vee P(X, b)$$
$$(Y = a \vee Y = b \supset Q(X, Y))$$

Now we ask the queries as stated in Table 5.3. Note that we now have 10.000 disjunctive terms in the first case and 100.000 disjunctive terms in the second. Recall that it was assumed in [40] that the number of disjunctive terms is 10% of the entire number of terms in the knowledge base. Additionally note that the predicates $P$ and $Q$ are contained nowhere else in the entire knowledge base except in the terms stated above.

All queries require reasoning by cases, but no unit propagation is applied. Note again the efficient handling of the quantifiers. In this scenario it is very important to note that there exists only *one* clause that can be used when reasoning by cases is applied. This implies that it is not necessary to choose from a set of clauses when reasoning by cases is applied.

In the next scenario we will show that the number of clauses that are used when reasoning by cases is applied causes a major drawback to efficiency. First of all we present the terms that are of concern in this test case:

$$(X = a \wedge Y = a \supset Q(X) \vee P(X, Y))$$
$$(Y = a \supset R(X) \vee P(X, Y))$$
$$(Y = b \supset S(X) \vee P(X, Y))$$
$$(X = a \supset \neg Q(X) \vee \neg R(X) \vee \neg S(X))$$

Additionally, we have the query $\exists Y.P(a, Y)$. Recall, that this query can be answered correctly already when allowing only one level of reasoning by cases (see previous chapter).

Note that every application of reasoning by cases includes several applications of unit propagation depending on the allowed level. Furthermore the test of the query is applied multiple times (refer to the example in the last chapter).

This query is answered in $280msec$ when allowing one level of reasoning by cases, and considering 10.000 disjunctive terms and 100.000 unit terms are contained in the knowledge base. If we allow two levels of reasoning by cases then the answering time is $210msec$.

The reason why the answering time of the query that allows two levels of reasoning by cases is faster than the one that supports only one level is due to the fact that we use a depth-first strategy in our approach when applying reasoning by cases. If we use the first clause and two levels of reasoning by cases are allowed, the query is known to be true and no further reasoning has to be accomplished.
But if we only allow one level of reasoning by cases we have to go through the first three clauses in the set and then succeed when using the last clause with reasoning by cases. This causes the differences in the answer times here.
Note that the clauses that are determined for the use with reasoning by cases are sorted by there term identifier and so it is possible to fix the order of clauses to be used (see chapter 4).

The answering time of $210msec$, when two levels of reasoning by cases are supported, shows that unit propagation is implemented efficiently since the evaluation of the query requires the testing of the query 5 times ($\approx 10msec$ each) and unit propagation itself is applied 4 times (see also Chapter 4).
Note that 5 tests of the query require about $50msec$ since testing a query only involves unit terms (refer to the results from the last section). Hence, one application of unit propagation requires less than $40msec$ since unit propagation is only a subprocess when reasoning by cases is applied. As we will see later on the time used for unit propagation will only increase in a reasonable way when 100.000 disjunctive terms are of concern.

To show that the number of disjunctive terms that are of concern when reasoning by cases is applied cause a major drawback concerning efficiency we will add disjunctive terms that are indirectly connected to the query.

For example, we add the disjunctive term $(R(X) \vee T(X))$ while the predicate $T$ is nowhere else contained in the knowledge base. And since we add the disjunctive term while using a smaller term identifier than the other terms of concern have, this disjunctive term will be used in an application of reasoning by cases at first.

Additionally, unit propagation can be applied successfully when adding the first predicate of the clause. When a ground instance of $R(X)$ is added as unit term then this has an effect on the disjunctive term $(X = a \supset \neg Q(X) \vee \neg R(X) \vee \neg S(X))$. Then the query is tested again, but without success.

Consequently, the algorithm will proceed to the next clause in the set if there is no further level of reasoning by cases allowed. If a further level of reasoning by cases is allowed then the query is known to be true with regard to the first predicate used in the current clause. This is due to the fact that at the next level of reasoning by cases again every clause may be chosen from the determined set except the actual one.
For example, the term $(X = a \wedge Y = a \supset Q(X) \vee P(X, Y))$ is chosen next for reasoning by cases at the second level. Since $(X = a \supset \neg Q(X) \vee \neg S(X))$ is now contained in the knowledge base and adding a ground instance of $Q(X)$ as unit term will create the new unit term $\neg S(X)$ the query is supported. The same holds of course for the second predicate $P$ of the current term.

But since the predicate $T$ will *not* support the query at any level of reasoning by cases the disjunctive term $(R(X) \vee T(X))$ will never support the query. As we could see, clauses of this kind cause several applications of unit propagation and the test of a query is applied multiple times.
So, we guarantee that all features of reasoning are accomplished for every single added disjunctive term. At the same time we prevent that those kind of clauses can support the query by using an unique predicate $(T)$.

In this test we will not only add one term of this kind, but up to sixty terms. Every single newly added term will be considered *before* the term is reached that supports the query.

The result can be seen in Figure 5.2. While we have 4 terms in the original set, we first add 4, then 12, 28, and finally 60 terms. Note that *none* of the terms will support the query at any level of reasoning by cases, but they are all considered *before* the original set of terms is of concern. In the knowledge base there are 10.000 disjunctive terms and 100.000 unit terms contained.

As we can see in the figure the answering times increases with the number of terms that are considered when reasoning by cases is applied. The effect is amplified by the number of levels of reasoning by cases that are allowed. This is explainable by the fact that each level of reasoning by cases reconsiders all possible terms again except the actual used ones (see chapter 4).

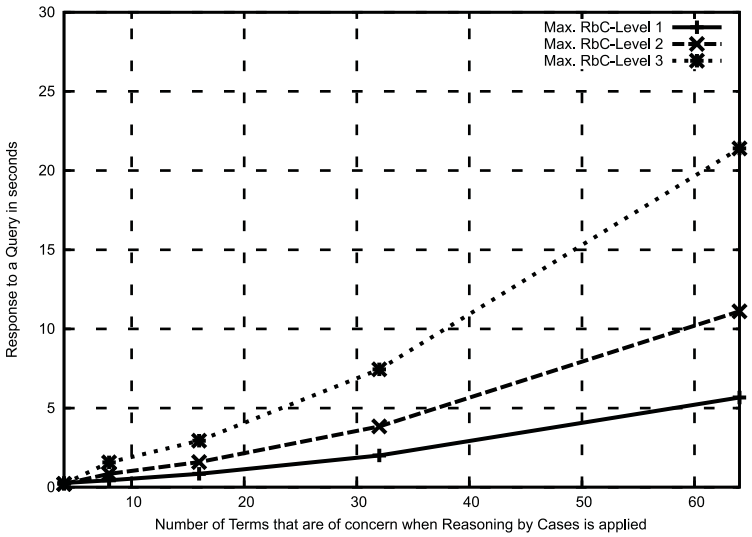


**Figure 5.2:** The influence of the number of clauses considered when reasoning by cases is applied on the answering time to a query. (10.000 Disjunctive Terms/100.000 Unit Terms)

If reasoning by cases fails for the actual term it goes to the next level of reasoning by cases if it is allowed to and otherwise it will try all other possible terms at this level to support the query before it returns to the previous level again. In fact this corresponds to the backtracking property that was explained in the last chapter.

Note that the answering time when we use reasoning by cases with a maximal level of two is only faster than the answering time when allowing only one level if the number of terms is only four.

The problem is the criteria used when reasoning by cases is applied. Since it is only tested if a predicate is directly or indirectly connected to the predicate in the query the set of terms that is of concern grows very fast. The size of growth depends on the structure or characteristic of the knowledge base. Think for example of SAT instances where every literal is directly or indirectly connected to every other literal in the instance. Consequently, if the instance would hold 1000 clauses *every* single clause would be considered during one application of reasoning by cases.

Note that our approach is not able to solve such kind of instances, because of the high number of clauses that have to be considered at each level of reasoning by cases and especially the fact that such kind of instances require very high levels of reasoning by cases. Recall, that the solution of the combinatorial puzzle introduced in Chapter 2 requires 8 levels of reasoning

| *Nr. of RbC-Terms* | *RbC-Level 1* | *RbC-Level 2* | *RbC-Level 3* |
|---|---|---|---|
| 4 | 0,28 | 0,21 | 0,21 |
| 8 | 0,44 | 0,85 | 1,57 |
| 16 | 0,85 | 1,59 | 2,94 |
| 32 | 2,01 | 3,84 | 7,45 |
| 64 | 5,67 | 11,11 | 21,40 |

**Table 5.4:** The answering times in seconds to the query $\exists Y.P(a, Y)$ depending on the number of terms that are considered when reasoning by cases is applied and the allowed level of reasoning by cases.(10.000 Disjunctive Terms / 100.000 Unit Terms)

by cases.

Consequently, our approach is not able to answer queries efficiently if the predicates of the query are connected directly or indirectly to a set that contains more than 16 to 32 disjunctive terms. As can be seen in the Figure 5.2 and the corresponding Table 5.4 the query is answered in $0,85sec$ when only allowing one level of reasoning by cases and $1,59sec$ when two levels are allowed and if there are 16 terms of concern when reasoning by cases is applied.

We think that answering times at about 1 second can be called efficient in our case. As can be seen from the figure the answering time to a query increases dramatically with the number of terms that are of concern with reasoning by cases. Especially, the amplified answering times when allowing a maximal reasoning by cases level of three suggest an exponential growth of answering times with regard to the user-defined level of reasoning by cases and the number of disjunctive terms that are used when reasoning by cases is applied. This result corresponds to the discussed worst-case complexity in Chapter 4.

Comprising, these results show that reasoning by cases with a maximal level of 2 causes a major drawback when the characteristic of the knowledge base contains sets of terms that contain more than 32 connected disjunctive terms.

### 5.4.4 The Size of the Knowledge Base

Since we already gave some test results concerning the size of the knowledge base we now present further results in detail.

As we said in the previous chapter the chosen data structure supports a large set of unit terms (more than 1.000.000) in the original knowledge base

| *Nr. of RbC-Terms* | *10.000/100.000* | *10.000/1.000.000* | *100.000/1.000.000* |
|---|---|---|---|
| 4 | 100% $(0,28s)$ | +11% $(0,31s)$ | +27% $(0,36s)$ |
| 8 | 100% $(0,44s)$ | +9% $(0,48s)$ | +20% $(0,53s)$ |
| 16 | 100% $(0,85s)$ | +4% $(0,88s)$ | +15% $(0,98s)$ |
| 32 | 100% $(2,01s)$ | +6% $(2,14s)$ | +19% $(2,38s)$ |
| 64 | 100% $(5,67s)$ | +4% $(5,88s)$ | +23% $(6,99s)$ |

**Table 5.5:** Comparison of the answering times corresponding to the number of terms (disjunctive terms / unit terms) contained in the knowledge base while allowing only one level of reasoning by cases

since those terms are not of concern when reasoning by cases is applied.
At this point we present the corresponding results. In Figure 5.3 the test from the last section is repeated in a knowledge base that contains 10.000 disjunctive terms and 1.000.000 unit terms. As you can observe the difference to the case where only 100.000 unit terms where of concern is negligible. Hence, the reasoning procedure is nearly independent of the number of unit terms.

We also said in the section "Preprocessing" in this chapter that the growth of disjunctive terms caused by the application of unit propagation during preprocessing does not cause any major drawbacks.
Figure 5.4 supports this statement. Now the knowledge base contains 100.000 disjunctive terms and 1.000.000 unit terms. As you can observe the large number of disjunctive term has only a slight impact on the answering times. This is again mainly due to the fact that databases can handle datasets of this size very efficiently [23]. Note, that this also implies that the implementation of unit propagation works efficiently even when 100.000 disjunctive terms must be considered.

In the Table 5.5 we summarize the results concerning the topic of this section by comparing the answering times presented here with the times from the last section. We compare all answering times measured at the reasoning by cases level 1.

## 5.5 Summary

In this chapter we presented a method for generating test instances and described the difficulties that arise when generating a test knowledge base. Unit terms are generated by random and we use SAT instances to generate disjunctive terms. Small number of $proper^+$ terms are added to the knowledge

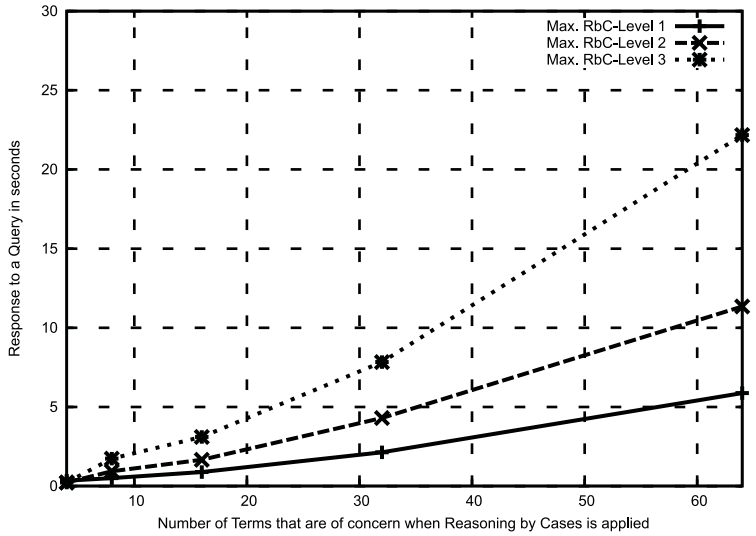


**Figure 5.3:** The influence of the number of clauses considered when reasoning by cases is applied on the answering time to a query. (10.000 Disjunctive Terms/1.000.000 Unit Terms)

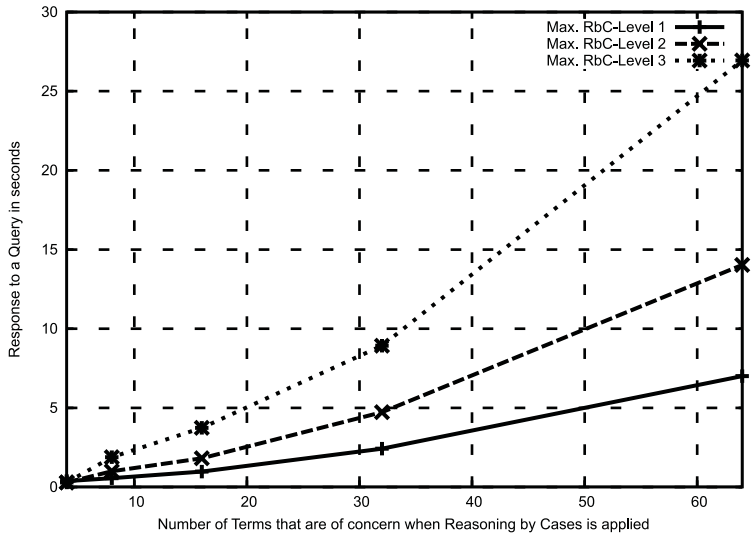


**Figure 5.4:** The influence of the number of clauses considered when reasoning by cases is applied on the answering time to a query. (100.000 Disjunctive Terms/1.000.000 Unit Terms)

base which contain specific rules or incomplete knowledge. This enabled us to ask the corresponding queries.

We discussed the disadvantage caused by preprocessing, namely the growth of the number of disjunctive terms caused by the application of unit propagation. For the practical case we could show under several assumptions that the number of disjunctive terms that hold more than two predicates cause the size of the knowledge base to double or triple. But additionally we presented results that confirmed that this kind of growth has no major drawback concerning efficiency.

As said in the last chapter we confirmed in this chapter that queries which require no reasoning by cases can be answered very efficiently. The same holds for queries that make use of reasoning by cases when the number of disjunctive terms that are of concern is relatively small ($< 30$).
At the same time we could show empirically that the implementation of unit propagation works efficiently. Even if 100.000 disjunctive terms are contained the implementation introduces no major drawback.

In contrast, we could show that the number of disjunctive terms that is of concern when reasoning by cases is applied has an major influence on the answering time to a query. In fact, if the set of disjunctive terms contains more than about 30 terms, the evaluation of a query is not efficient anymore. In consequence, the implementation of reasoning by cases must be improved. Especially, the criterion that determines the set of disjunctive terms that is of concern when reasoning by cases is applied must be improved to reduce the size of the set.

Furthermore, we showed that the size of the knowledge base has only a slight impact on the performance of our approach. Especially, the number of unit terms increases the answering times to a query in a negligible way only.

# Chapter 6

# Summary and Discussion

In this chapter we are going to provide a summary and a critical assessment of the work that was done during this thesis. Furthermore, we will discuss directions for future research.

## 6.1 Summary

In this thesis, we investigated and implemented a deductive and logical sound reasoning procedure that is able to handle incomplete first-order knowledge bases that contain disjunctive information.

First, we introduced the deductive reasoning procedure that is of concern in this work and then we examined the properties of the reasoning procedure itself. We could show that one of the main features, namely unit propagation, introduced by the reasoning procedure causes an exponential growth of the equality terms if equalities are represented in DNF. In addition, we could show that during an application of unit propagation only redundant terms can be deleted. Hence, the application of unit propagation is not as unproblematic as in the propositional case. In addition, we observed that the use of *inequality* causes the major drawback concerning complexity in our implementation; therefore inequalities were no longer supported.

The fact that we did not support inequalities had a major impact on the implementation. Especially, the handling of the $\forall$-quantifier was simplified in a crucial way since we could show that we do not have to substitute every possible constant to determine if the $\forall$-quantifier holds.

Furthermore, we implemented all main features of the reasoning procedure while using an encoding scheme for $proper^+$ terms and employing database features to enable an efficient handling of huge datasets. Afterwards we discussed the efficiency of our implementation and showed that we

achieved efficient implementations concerning the evaluation of queries (e.g., quantifiers) and the application of unit propagation even if about $10^6$ terms are contained in the knowledge base.

In contrast, we could show that the application of reasoning by cases can cause the answering times to a query to increase dramatically. Beside the user-defined reasoning by cases level the answering times depend on the *characteristic* of the knowledge base. Namely, the number of disjunctive terms that are connected to each other by predicates plays a major role in this context.

## 6.2 Critical Assessment

As said before it was possible to determine that the growth of equality terms was caused by the use of inequality when we use DNF to represent equalities. This was one of the main reasons to *exclude* inequalities from the entire reasoning procedure. As a downside, if we want to include inequalities later on, perhaps in a limited form, this would require substantial revision of the implementation.

While the implementation of the reasoning procedure concerning the representation of $proper^+$ terms and unit propagation caused only minor problems, the implementation of reasoning by cases was very complex and, ultimately, could not be accomplished in a satisfying way. The reason for this difficulty is originated in the theoretical definition of reasoning by cases given in the reasoning procedure. The choice of the clause that is used for reasoning by cases is non-deterministic in the definition. The criterion we used simply determined which disjunctive terms contained in the entire knowledge base are directly or indirectly connected to the query. Depending on the characteristic of the knowledge base the number of terms can be too large for our approach to stay efficient. Already small numbers (about 30) cause the answering time to a query to increase dramatically.

In addition, we did not discuss the soundness and completeness of our implementation with regard to the original reasoning procedure in every case. For instance, the format of the query is restricted so that we do not support queries such as $\forall((X = a \wedge X \neq a) \supset P(X))$.

The implementation does also not support every feature of equality terms. Especially, we can not handle equality terms of the following type: $(X = Y \wedge Y \neq Z)$.

Furthermore, the architecture of the implementation can be improved since there exist several interfaces such as the Prolog-C and the C-MySQL interface that connect our program to the database. For instance, it would be

possible to decrease the number of interfaces and therefore increase efficiency.

Although we tried to establish an appropriate test environment to determine the efficiency of our implementation it remains to be seen if the results would be similar under real-world conditions. Recall, that there was no appropriate test knowledge base available. As in the propositional case the characteristic of the test instance as a major influence on the test results.

Additionally, the presented implementation is a prototype that is only suitable to examine the feasibility of the deductive reasoning procedure and to analyse its efficiency in general. In particular, the user interface needs to be improved for the use by others.

## 6.3 Future Work

In this section we provide a brief outlook on future research. We begin this outlook by presenting some proposals to improve the introduced implementation. Additionally, we will propose an improvement of the given reasoning procedure.

There exist many essential improvements that can be made concerning the implementation since it was the first attempt at all to implement the given reasoning procedure. For instance, the criterion used when applying reasoning by cases. An enhanced criterion could decrease the answering time to queries in an essential way, since the answering time increases with the number of disjunctive terms used when reasoning by cases is applied. We think, that this improvement would require an updated data structure or a graph that holds the necessary information to determine the clauses which are used with reasoning by cases.
Besides improving the approach by advanced algorithms there also exist several possibilities to increase the performance by using the given infrastructure in a more effective way. For example, the use of database features can be enlarged and rectified (e.g., the use of nested SQL-queries).
Furthermore, a discussion concerning soundness and completeness of every part of our implementation is necessary. Also the set of possible queries must be extended.

Additionally, there is a lot of research necessary to enable an accurate and appropriate test environment to verify the performance of a given approach. While the research activities in the propositional case are very intensive the research activities that deal with huge incomplete first-order knowledge bases that contain disjunctive information are very seldom. Consequently, it is very hard to determine the efficiency of an implementation. Recall, that this problem is not yet entirely solved in the propositional case. We ap-

proximated the worst-case complexity by determining the maximal number of applied unit propagations which resulted in a worst-case complexity of $O((n^{2k+2})^{RbC-Level})$ when $|KB| = |\mathcal{C}_{\mathcal{KB}}| = n$, $RbC - Level$ is the user-defined reasoning by cases level and $k$ denotes the maximal number of free variables contained in one single term. In the practical case the exponent $RbC - Level$ remains, but the base $n^{2k+2}$ is decreased in an essential way. For instance, the facts that knowledge bases in our field of application consist of no terms that are connected by a predicate to every other possible term contained in the knowledge base and it is not necessary to substitute every variable by every possible constant decrease the polynomial.

While there exist several possibilities to advance the presented implementation, we also think that the given deductive reasoning procedure can be extended. One possible extension would for instance affect the return value "unknown" of the reasoning procedure. The information content of this return value is small, although an immense number of reasoning operations are executed most of the time when a query is answered. Therefore, we suggest a return value like "unknown, but ..." to take advantage of the reasoning that was applied and the corresponding results.
For example, suppose the following terms to be contained in the knowledge base:

$$P(X) \vee \neg Q(X) \vee R(X) \vee T(X)$$
$$Q(X) \vee \neg R(X)$$
$$\neg Q(X) \vee \neg T(X)$$
$$R(X) \vee \neg T(X)$$

Then it would be possible to answer the query $\exists X.P(X)$ with "unknown, but $P(X) \vee \neg Q(X)$ and $P(X) \vee R(X)$ are implied by the KB". Note, that we do not use the expression "known to be true", because only single literals are known to be true.
This approach would increase the information content of the negative response to a query in an essential way. The suggested extension has its origin in a brief discussion with Craig Boutilier [8].

# Bibliography


[1] F. Afrati, S. S. Cosmadakis, M. Yannakakis, *On Datalog vs. Polynomial Time*, Proceedings of the Tenth ACM PODS Conference, pages 13-25, 1991

[2] A. R. Anderson and N.D. Belnap, *Entailment, The Logic of Relevance and Necessity*, Princeton University Press, 1975

[3] H. Andreka, I. Hodkinson, and I. Nemeti, *Modal Languages and Bounded Fragments of Predicate Logic*, ILLC Research Report ML-96-03, University of Amsterdam, 1996

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, *The Description Logic Handbook*, Theory, Implementation and Applications, ISBN: 0521781760, Cambridge University Press, 2003

[5] R. Bayardo Jr., R. C. Schrag, *Using CSP Look-Back Techniques to Solve Real-World SAT Instances*, Proceedings of the 14th National Conference on Artificial Intelligence, pages 203-208, 1997

[6] B. Beckert, J. Posegga, *Lean tableau-based deduction*, Technical Report, Universitaet Karlsruhe, 1994

[7] A. Blake, *Canonical Expressions in Boolean Algebra*, Ph.D. thesis, University of Chicago, 1938

[8] Personal Communication: C. Boutilier, University of Toronto, Toronto, cebly@cs.toronto.edu

[9] I. Bratko, *PROLOG, Programming for Artificial Intelligence*, third edition, Addison Wesley, 2001

[10] S.A. Cook, *The complexity of theorem proving procedures*, Proceedings of the 33rd Annual ACM Symposium on the Theory of Computation, pages 151-158, 1971

[11] J. Calmetz, G. Bittencourty, K. Homannz, A. Lulayz, *MANTRA: A Multi-Level Hybrid*, In T. Pequeno, F. Carvalho (Eds.), Proceedings of the XI Brazilian Symposium on Artificial Intelligence, Fortaleza, pp. 493-506, 1994

[12] J. M. Crawford, D. W. Etherington, *A Non-Deterministic Semantics for Tractable Inference*, Proceedings of the AAAI-98, pages 286-291, 1998

[13] Cycorp, *The Cyc Knowledge Server*, http://www.cyc.com/, Austin, Texas

[14] M. Dalal, *Anytime Families of Tractable Propositional Reasoners*, Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics, pages 42-45, 1996

[15] M. Dalal, *Semantics of an Anytime Family of Reasoners*, Proceedings of the Twelfth European Conference on Artificial Intelligence, pages 360-364, 1996

[16] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schoening, *A deterministic* $(2-2/(k+1)^n)$ *algorithm for k-SAT based on local search*, paper to be published, 2002

[17] E. Dantsin, G. Gottlob, T. Etter, A. Voronkov, *Complexity and Expressive Power of Logic Programming*, Proceedings 12th Annual IEEE Conference on Computational Complexity, 1997

[18] M. Davis, H. Putnam, *A computing procedure for quantification theory*, Journal of the Association for Computing Machinery, pages 201-215, 1960

[19] DIMACS, *Satisfiability Suggested Format*, Center for Discrete Mathematics and Theoretical Computer Science, 1993

[20] J. M. Dunn, *Intuitive Semantics for First-Degree Entailments and Coupled Trees*, Philosophical Studies 29, pages 149-168, 1976

[21] $ECL^iPS^e$, *The* $ECL^iPS^e$ *Constraint Logic Programming System*, Imperial College London, http://www-icparc.doc.ic.ac.uk/eclipse/

[22] J. Eder, *Logic and Databases*, Advanced Topics in Artificial Intelligence, pages 95-103, 1992

[23] R. Elmasri, S. B. Navathe, *Fundamentals of Database Systems*, Addison-Wesley, ISBN: 0201542633, 4th edition, 2000

[24] A. Frisch, *Inference without chaining*, Proceeding of the IJCAI-87, Morgan Kaufmann, San Francisco, pages 515-519, 1987

[25] H. Gallaire, J. Minker, J-M. N. Nicolas, *Logic and Databases: A deductive approach*, ACM Computing Surveys, 16(2):153-185, 1984

[26] I.P. Gent and T. Walsh, *The search for Satisfaction*, Internal Report, Department of Computer Science, University of Strathclyde, 1999

[27] I.P. Gent and T. Walsh, *The Hardest Random SAT Problems*, Proceedings of German Conference on AI, KI-94, 1994

[28] I.P. Gent and T. Walsh, *An empirical analysis of search in GSAT*, Journal of Artificial Intelligence Research,1:23-57, 1993

[29] G. Gottlob, N. Leone, F. Scacello, *Robbers, marshals and guards: Game theoretic and logical characterizations of hypertree width*, In Proceedings of the 20th ACM Symposium on Principles of Database Systems, 2001

[30] E.A. Hirsch and A. Kojevnikov, *Unit Walk: A new SAT Solver that uses local search guided by unit clause elimination*, Steklov Institute of Mathematics St. Petersburg, PDMI reprint, 2001

[31] H. Kautz, B. Selman, *Planning as Satisfiability*, Proceedings of the 10th ECAI, pages 359-363, 1999

[32] Personal Communication: T. H. Kolbe, University of Bonn, Bonn, tk@ikg.uni-bonn.de, via email

[33] K. Konolige, *A Deduction Model of Belief*, Brown University Press, 1986

[34] E. Koutsoupias, C.H. Papadimitriou, *On the greedy algorithm for satisfiability*, Information Processing Letters, 43, pages 53-55, 1992

[35] G. Lakemeyer, *Models of Belief for Decidable Reasoning in Incomplete Knowledge Bases*, Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, 1990

[36] G. Lakemeyer, H. Levesque, *Evaluation-Based Reasoning with Disjunctive Information in First-Order Knowledge Bases*, In the Proceedings of KR-02, pages 14-23, 2002

[37] G. Lakemeyer, H. Levesque, *A Logic of limited Belief for Evaluation-Based Reasoning*, unpublished note, 2003

[38] H. Levesque, *A Formal Treatment of Incomplete Knowledge Bases*, Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, 1981

[39] H. Levesque, *A Logic of Implicit and Explicit Belief*, Proceedings of the AAAI-84, pages 198-202, 1984

[40] H. Levesque, *A Completeness Result for Reasoning with Incomplete First-Order Knowledge Bases*, In the Proceedings of KR-98, 1998

[41] H. Levesque, G. Lakemeyer, *The Logic of Knowledge Bases*, MIT Press, 2001

[42] Personal Communication: H. Levesque, University of Toronto, Toronto, hector@cs.toronto.edu

[43] C.M. Li, M. Anbulagan, *Look-ahead versus look-back for satisfiability problems*, in Proceedings of the International Conference on Principles and Practice of Constraint Programming, 1997

[44] C. M. Li, M. Anbulagan, *Heuristics Based on Unit Propagation for Satisfiability Problems*, International Joint Conference on Artificial Intelligence (IJCAI), pages 315-326, 1997

[45] Y. Liu, *Towards Tractable Reasoning in Classical Knowledge Bases*, Depth Paper, University of Toronto, Toronto, 2001

[46] Y. Liu, H. Levesque, *A Tractability Result for Reasoning with Incomplete First Order Knowledge Bases* , Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, 2003

[47] Personal Communication: Y. Liu, University of Toronto, Toronto, yliu@cs.toronto.edu

[48] J. McCarthy, *Programs with Common Sense*, In Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory, pages 77-84, London, U.K. Her Majesty's Stationery Office, 1959

[49] J. Minker, *Logic and databases: A 20 year retrospective*, in Logic in Databases, International Workshop LID 96, San Miniato, Italy, July 1-2, 1996, Proceedings, Vol. 1154 of Lecture Notes in Computer Science,Springer, pp. 3-57, 1996

[50] R. C. Moore, *The Role of Logic in Knowledge Representation and Common Sense Reasoning*, in Proceedings of the Second National Conference of the American Association for Artificial Intelligence, Pittsburgh, pages 428-433, 1982

[51] P. Morris, *The Breakout method for escaping from local minima*, Proceedings of the 11th National Conference on AI, American Association for Artificial Intelligence, 1993

[52] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansky, *Determining computational complexity from characteristic phase transitions*, Nature, Vol 400, pages 133-137, 1999

[53] MySQL, *MySQL Open Source Database*, Version 4.01, http://www.mysql.com

[54] C.H. Papadimitriou, *On selecting a satisfying truth assignment*, Proceedings of the Conference on the Foundations of Computer Science, pages 163-169, 1991

[55] A. Patel-Schneider, *A decidable first-order logic for knowledge representation*, Proceedings of the IJCAI-85, Los Angeles, CA, pages 455-458, 1985

[56] R. Reiter, *On Closed-World Databases*, In H. Gallaire and J.Minker, editors, Logic and Data Bases, pages 55-76. Plenum Press, New York, 1978

[57] SAT Competition Problems, *SATLIB - The Satisfiability Library*, http://www.satlib.org/

[58] U. Schoening, *A probabilistic algorithm for k-SAT and constraint satisfaction problems*, Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, FOCS99, pages 410-414, 1999

[59] U. Schoening, *Logik für Informatiker*, Spektrum Akademischer Verlag, 5th edition, 2000

[60] R. Schuler, U. Schoening, O. Watanabe, *An improved randomized algorithm for 3-SAT*, Technical Report TR-C146, Dep. of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2001

[61] B. Selman, H. Levesque, *A new method for Solving Hard Satisfiability*, Proceedings of the 10th National Conference on AI, pages 440-446. American Association for Artificial Intelligence, 1992

[62] B. Selman, H. Kautz, *An empirical study of greedy local search for satisfiability testing*, Proceedings of the 11th National Conference on AI, pages 46-51, American Association for Artificial Intelligence, 1993

[63] B. Selman, H. Kautz, B. Cohen, *Noise Strategies for Improving Local Search*, Proceedings of the 12th National Conference on AI, pages 337-343. American Association for Artificial Intelligence, 1994

[64] B. Selman, H. Levesque, D. Mitchell, *Generating Hard Satisfiability Problems*, Artificial Intelligence, vol. 81, no. 1-2, pages 17-29, 1996

[65] A. Silberschatz, H. F. Korth, S. Sudarshan, *Database System Concepts*, McGraw-Hill Education, ISBN: 0071217622, 2001

[66] B.C. Smith, *Reflection and Semantics in a Procedural Language*, MIT Laboratory for Computer Science, Technical Report MIT-TR-272, 1982

[67] W.M. Spears, *A neural network algorithm for boolean satisfiability problems*, Proceedings of the 1996 International Conference on Neural Networks, pages 1121-1126, 1996

[68] W.M. Spears, *Simulated Annealing for hard satisfiability problems*, D.S. Johnson and M.A. Trick, editors, In Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, pages 533-558, 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, American Mathematical Society.

[69] W.M.K. Trochim, *Deductive and Inductive Thinking*, http://trochim.human.cornell.edu/kb/dedind.htm, 2000

[70] J.D. Ullmann, *A Database Approach to Knowledge Representation*, Proceedings of the 13th National Conference of the American Association for Artificial Intelligence. AAAI Press, MIT Press, 1996

[71] J.D. Ullmann, *Assigning an appropriate meaning to database logic with negation*, Computers as our better Partners, pages 216-225, World Scientific Press, 1994

[72] J. Zhang, H. Zhang, *Combining local search and backtracking techniques for constraint satisfaction*, Proceedings of 13th National Conference on Artificial Intelligence, pages 369-374, American Association for Artificial Intelligence, AAAI Press/The MIT Press, 1996

[73] H. Zhang, M. Stickel, *An efficient algorithm for Unit Propagation*, Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics, 1996

[74] H. Zhang, M. Stickel, *Implementing the Davis-Putnam method*, in I. Gent, H. van Maaren, and T. Walsh, editors, SAT 2000, pages 309-326, IOS Press, 2000