Faculty of Computer Science

MCL Master's Thesis

Finite Herbrand Models for Monadic Clauses with Unary Function Symbols

Author:

Muhammad Zahid Zia

Supervisor:

Prof. Dr.-Ing. Franz Baader

Advisor:

Dr.-Ing. Stefan Borgwardt

March, 2016

Technische Universität Dresden

Author: **Muhammad Zahid Zia**

Matrikel-Nr: **4026829**

Title: **Finite Herbrand Models for Monadic Clauses with Unary Function Symbols**

Degree: **International MSc Program in Computational Logic**

Date of Submission:

Declaration

Hereby I certify that the thesis has been written by me. Any help that I have received in my research work has been acknowledged. Additionally, I certify that I have not used any auxiliary sources and literature except those cited in the thesis.

Muhammad Zahid Zia

Abstract

Decidability of *First Order Logic (FOL)* is a key and active research area in logic and computer science. As *FOL* is undecidable in its general form, a lot of research is focused on finding classes of *FOL* which are decidable and studying their properties. In this thesis, we work on one such decidable fragment that only allows monadic predicate and function symbols (with the exception of one constant). We also allow just one variable in the formulas.

Research in Logic generally does not concentrate on the finiteness of interpretations. Yet, for any realistic and practical knowledge representation purpose we only have finitely many resources to represent the real world. Therefore we focus on finding the finite Herbrand models for the above mentioned class. We also provide a terminating decision procedure for the existence of a finite Herbrand model for this restricted fragment of *FOL*. In the end we provide some optimizations and variants of the original algorithm for better performance on certain problems.

Acknowledgements

I like to express my gratitude to all people who have helped me in making this thesis possible.

First I would like to thank my adviser Dr.-Ing. Stefan Borgwardt for his kind support and infinite patience. This thesis is a product of his thoughtful guidance. I would also like to thank my supervisor Prof. Dr.-Ing. Franz Baader for his support and Prof. Sebastian Rudolph for accepting to read and co-supervise this thesis. I want to show my gratitude to all the faculty members and staff of International Center for Computational Logic at TU Dresden for giving me a wonderful and enriching experience.

Last but not least I want to thank all my family for their unwavering support especially my mother, my father and my wife. I want to thank all my friends too for making my stay in Dresden very memorable and precious.

Contents

Chapter 1

Introduction

There is a lot of research done for finding a decidable fragment of *First Order logic (FOL)* retaining enough expressive power to solve a particular class of problems. Some of the well studied decidable fragments of *FOL* are the monadic predicate calculus [3, 4] which only allows unary predicate symbols without functions and equality, effectively propositional logic also known as the Bernays-Schönfinkel class which contains *FOL* formulas that, when written in prenex normal form, have an $\exists^*\forall^*$ quantifier prefix and do not contain any function symbols, and Ackermann's class [5] where prenex formulas have a prefix of the form $\exists^*\forall\exists^*$. All of these classes have been found to be decidable. The class of *FOL* formulas that we want to study in this thesis is an extension of formulas used by monadic predicate calculus, but we also allow unary function symbols as well and a constant.On the other hand we do not allow the equality predicate and we restrict the formulas to clausal form in which only one variable can appear.

Another emerging area of interest in the past two decades is the use of logic for data and knowledge management. Datalog [6], a counterpart of Prolog, was devised for data processing and tries to extend normal factual data with the help of logic rules to discover and deduce inherent knowledge within the data. Under model-theoretic semantics, a Datalog program can be considered as an *FOL* theory where Datalog rules can be seen as function-free *FOL* Horn clauses. This ensures the decidability of Datalog and also ensures minimality of the models of Datalog programs which is very important for developing efficient algorithms for query systems. This is why in this thesis we are not only interested in the decidability of the fragment, we are also interested in developing an efficient algorithm for reasoning task over the logic. So we are not interested in just any model for the logic but in a more particular question, which is the existence of a finite Herbrand model for a set of clauses. This question has importance in finite model theory as well

as in database theory.

Datalog in its general form is not very expressive, but there are multiple extensions for *Datalog* created to increase the expressiveness, like *Datalog*± [8], which extends Datalog with existential quantification, the equality predicate, and the truth constant false, while preserving not only decidability, but also tractability of query answering in data complexity by using syntactic restrictions. Adding just the existential quantifier without any restrictions like guardedness proposed in [9] can make the models infinite and query answering undecidable [10], and similarly for adding equality and false.

As the topic of this thesis is also related to one such decidable fragment of *FOL*, it would be beneficial to compare the expressiveness of this fragment with already well-known decidable fragments like *Datalog*. We give the clausal formalism for our fragment of interest later, but most importantly we do not restrict our clauses to Horn clauses, which means that we can have disjunctions in the head, which improves expressiveness greatly. For example, one cannot express the sentence "If the tire is flat, either the valve is leaky or the tube is punctured" in *Datalog* or *Datalog*±, but one can express this as the following *monadic clause*.

$$flatTire(x) \rightarrow leakyValve(x) \vee punctured Tube(x)$$

But this does not mean that monadic clauses are more expressive than Datalog, as Datalog rules do not restrict the use of n-ary predicates, which can express relationships between different elements of the interpretation. For example, one can express graph reachability as the following *Datalog* rule:

$$reachable(x) \wedge edge(x, y) \rightarrow reachable(y),$$

but this is not possible in monadic clauses, as binary predicates are not allowed.

A similar problem has already been addressed in [1] for a subset of the fragment that we will focus on. They show that deciding the existence of finite Herbrand models is ExpTime-complete for a set of propagation rules (anti-Horn monadic clauses). We will show for a larger fragment of *FOL* where we allow conjunctions on the left-hand side of the clause.

Chapter 2

Monadic Clauses

2.1 Preliminaries

In this section we list some notions which we will use later in this thesis. We assume the reader to be familiar with basic concepts of logic.

Definition 2.1.1 (Interpretation)**.**
An *interpretation* $\mathcal{I}$ for a *FOL* language $\mathcal{L}(\mathcal{P}, \mathcal{F}, \mathcal{V})$ consists of a non-empty set $\mathcal{D}$ and a mapping $.^{\mathcal{I}}$, which satisfies the following conditions:

- Every n-ary function symbol $f/n \in \mathcal{F}$ is mapped to an n-ary function $g^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$.
- Every n-ary relation symbol $P/n \in \mathcal{P}$ is mapped to an n-ary relation $P^{\mathcal{I}} \subseteq \mathcal{D}^n$.

$\mathcal{D}$ is called the domain of the interpretation.

Definition 2.1.2 (Herbrand Interpretation)**.**
A Herbrand interpretation is an interpretation $\mathcal{I}$ where

- The domain of $\mathcal{I}$ consists of all the ground terms over the set of functions. This domain is also called Herbrand universe.
- All the ground terms are mapped onto themselves, i.e $t^{\mathcal{I}} = t$.

A Herbrand interpretation is finite if each predicate is interpreted by a finite subset of the potentially infinite Herbrand universe.

2.2 Monadic Clauses

In this section, we introduce the fragment of *first order logic* that we will focus on for our work and also some of the new notions required. We work with a fragment of first order logic constructed over the signature of finitely many unary predicates $\mathcal{P}$, finitely many unary function symbols $\mathcal{F}$, one constant a, and one variable x.

We use the symbols $P, Q, P_1, Q_1, P_2, \ldots$ for predicates and $f, g, \ldots$ for functions. The atomic formulas are of the form $P(t)$, where $P \in \mathcal{P}$ and t is a term over the function symbols in $\mathcal{F}$, the constant a and the variable x. All the ground terms over this signature are of the form $f_1(\cdots f_n(a)\cdots)$ where $\{f_1, \ldots, f_n\} \subseteq \mathcal{F}$. The length of the term is the number of function symbols used to construct the term.

We consider only the clausal form of the fragment which consists of *monadic clauses* as defined below.

Definition 2.2.1 (Monadic Clause)**.**
A *monadic clause* is a clause of the following form.

$$P_1(t_1) \wedge \cdots \wedge P_n(t_n) \rightarrow Q_1(t_{n+1}) \vee \cdots \vee Q_m(t_{n+m})$$

for $P_1, \ldots, P_n, Q_1, \ldots, Q_m \in \mathcal{P}$ and terms $t_1, \ldots, t_{n+m}$ over $\mathcal{F}$ and x. The variable x is universally quantified.

A finite Herbrand interpretation $\mathcal{H}$ is a model for a set $\mathcal{C}$ of clauses if it satisfies every clause in $\mathcal{C}$. Note that n or m can be zero. In the case that n is zero, the empty conjunction on the left-hand side of the clause becomes a $\top$ (representing the truth value true), which means that clause becomes a positive clause. In the case that m is zero, the empty disjunction on the right-hand side becomes $\bot$ (representing the truth value false) making the clause negative. It also should be noted that in case of a positive clause all the atoms on the right-hand side must be ground; otherwise there would be no finite Herbrand model for the clause set.

2.2.1 Normalized Clause Sets

To reduce the complexity of clauses, we flatten them by replacing the terms of length higher than 1. We extend the notion of normalized set of clauses from [1] for our generalized monadic clauses.

Definition 2.2.2 (Normalized clause set)**.**
A finite set $\mathcal{C}$ of monadic clauses is normalized if there exists a set $\mathcal{D}(\mathcal{C}) \subseteq \mathcal{P} \times \mathcal{F}$ such that

- for every $(P, f) \in \mathcal{D}(\mathcal{C})$, we have $P^f \in \mathcal{P}$ and the two clauses $P^f(x) \to P(f(x))$ *(increasing clause)* and $P(f(x)) \to P^f(x)$ *(decreasing clause)* are contained in $\mathcal{C}$, and
- all clauses in $\mathcal{C}$ expect the ones mentioned above must be flat, i.e., of the form

  $$P_1(t_1) \wedge \cdots \wedge P_n(t_n) \to Q_1(t_{n+1}) \vee \cdots \vee Q_m(t_{n+m})$$

  where t_i can be either the variable x or the constant a, with $i \in \{1, \ldots, n+m\}$.

The reason we want to create this normalized set is that it allows us to separate terms of different length, because after normalization if we want to check if a term satisfies a flat clause, we only need to care about that one term. And the only link between terms of different lengths are the increasing and decreasing clauses.

A terminating procedure is given in [1] to transform an arbitrary finite set of propagation rules into to an equivalent normalized set. We extend this procedure for a finite set of monadic clauses. Let $\mathcal{C}$ be a finite set of monadic clauses. We transform it into a normalized set $\mathcal{C}'$.

- We initialize $\mathcal{C}' := \mathcal{C}$ and the set $\mathcal{D}(\mathcal{C}')$ as follows.
  If for $(P, f) \in \mathcal{P} \times \mathcal{F}$ there is a unique Q such that $Q(x) \to P(f(x))$ and $P(f(x)) \to Q(x)$ are in $\mathcal{C}'$ and the same does not hold for any other pair in $\mathcal{P} \times \mathcal{F}$ and Q, then we rename Q to P^f and add (P, f) to $\mathcal{D}(\mathcal{C}')$.
- While $\mathcal{C}'$ is not yet normalized, we choose a clause

  $$c = P_1(t_1) \wedge \cdots \wedge P_n(t_n) \to P_{n+1}(t_{n+1}) \vee \cdots \vee P_{n+m}(t_{n+m})$$

  such that there is still a term $t_i = f(t_i')$ for some $f \in \mathcal{F}$ and term t_i'.
  If (P_i, f) is already in $\mathcal{D}(\mathcal{C}')$, we simply replace $P_i(t_i)$ with $P_i^f(t_i')$. Otherwise, we add the new predicate P_i^f to $\mathcal{P}$, add (P_i, f) to $\mathcal{D}(\mathcal{C}')$, extend the clause set $\mathcal{C}'$ with two clauses $P_i(f(x)) \to P_i^f(x)$ and $P_i^f(x) \to P_i(f(x))$, and replace the atom $P_i(f(t_i))$ in the clause c with $P_i^f(t_i')$.

The above mentioned process terminates, as the clause set $\mathcal{C}$ is finite so each term occurring in $\mathcal{C}$ is also of some finite length. As we reduce the length of the term in each step, the process terminates after finitely many steps.

Lemma 1. *A set $\mathcal{C}$ of monadic clauses has a finite Herbrand model iff the normalized set $\mathcal{C}'$ has a finite Herbrand model.*

Proof. We prove the claim by induction on the process of transformation. In the initialization step, we only do renaming of predicates which does not affect the existence of a finite Herbrand model. For the second part of the process, it is enough to show the claim for one replacement step.
If $\mathcal{H}$ is a finite Herbrand model of $\mathcal{C}'$ before the replacement, we can extend $\mathcal{H}$ for the new predicate P_i^f as $P_i^{f\mathcal{H}} := \{w(a) \mid f(w(a)) \in P_i^{\mathcal{H}}\}$ where w is some word over $\mathcal{F}$. It is clear under this extended interpretation $\mathcal{H}'$ that the atom $P_i(f(w(a)))$ is true in $\mathcal{H}'$ iff $P_i^f(w(a))$ is. And thus it satisfies all the clauses where $P_i(f(t_i))$ was replaced by $P_i^f(t_i')$ and also the increasing $(P_i^f(x) \to P_i(f(x)))$ and decreasing $(P_i(f(x)) \to P_i^f(x))$ clauses added during this step.
Now let there be a finite Herbrand model $\mathcal{H}$ of $\mathcal{C}'$ after the replacement. As it satisfies the increasing and decreasing clauses for every $(P_i, f) \in \mathcal{D}(\mathcal{C})$, we have that $P_i^{f\mathcal{H}} = \{w(a) \mid f(w(a)) \in P_i^{\mathcal{H}}\}$, so $P_i(f(w(a)))$ is true in $\mathcal{H}$ iff $P_i^f(w(a))$ is, which means that if we change back $P_i^f(t_i')$ to $P_i(f(t_i))$, then $\mathcal{H}$ still satisfies the clause. □

Let us take a look at an example to have a bit more clarity on the transformation process.

Example 2.2.1.

Consider the following simple set of monadic clauses.

$$\begin{aligned}\mathcal{C}_1 := \{&\top \to P_1(f(a)) \vee P_2(f(g(a))),\\ &P_1(f(x)) \to P_3(x),\ P_3(x) \to P_1(f(x)),\\ &P_4(x) \wedge P_2(f(x)) \to P_4(f(x))\}\end{aligned}$$

To normalize this set using the above mentioned algorithm, we start by initializing the set $\mathcal{C}_1'$ with $\mathcal{C}_1$. Now as we can note that there is $(P_1, f) \in \mathcal{P} \times \mathcal{F}$ and the predicate P_3 which satisfies the condition in initialization step, so we rename P_3 to P_1^f, add (P_1, f) in $\mathcal{D}(\mathcal{C}_1')$.
We then go on with the second step of the process as there is no other candidate satisfying this condition. We choose the first clause and term $t = f(a)$ as there is the function symbol f and term $t' = a$ as (P_1, f) is already in $\mathcal{D}(\mathcal{C}_1')$ so we just replace $P_1(f(a))$ with $P_1^f(a)$.Then we choose the first clause again but for term $t = f(g(a))$ as there is the function symbol f and term $t' = g(a)$ such that $t = f(t')$. We add P_2^f to $\mathcal{P}$,

add (P_2, f) to $\mathcal{D}(\mathcal{C}_1')$, add the increasing and decreasing clauses $P_2^f(x) \to P_2(f(x))$ and $P_2(f(x) \to P_2^f(x)$ respectively, and replace $P_2(f(g(a)))$ with $P_2^f(g(a))$ in the clause. And we repeat the step for (P_2^f, g) and (P_4, f) until $\mathcal{C}_1'$ is normalized.

After completion the set $\mathcal{C}_1'$ and $\mathcal{D}(\mathcal{C}_1')$ would be as follows.

$$\begin{aligned}
\mathcal{C}_1' := \{&\top \to P_1^f(a) \vee P_2^{fg}(a),\\
&P_1(f(x)) \to P_1^f(x), P_1^f(x) \to P_1(f(x)),\\
&P_4(x) \wedge P_2^f(x) \to P_4^f(x),\\
&P_2(f(x)) \to P_2^f(x), P_2^f(x) \to P_2(f(x)),\\
&P_2^f(g(x)) \to P_2^{fg}(x), P_2^{fg}(x) \to P_2^f(g(x)),\\
&P_4(f(x)) \to P_4^f(x), P_4^f(x) \to P_4(f(x))\}\\
\mathcal{D}(\mathcal{C}_1') = \{&(P_1, f), (P_2, f), (P_2^f, g), (P_4, f)\}
\end{aligned}$$

2.2.2 Reduced Clause Sets

Even after normalization we can still have *mixed* clauses, i.e. having both ground and non-ground atoms. To make the algorithm a bit more simple, we get rid of all the ground atoms from the clause set by non-deterministically guessing a set of predicates for a finite Herbrand model such that for every predicate P in that set the grounded atom $P(a)$ is in the model, and for every predicate not in the set the grounded atom is not in the model.

Definition 2.2.3 (Reduced clause set)**.**
For a set $\mathcal{C}$ of normalized monadic clauses and a set $\mathcal{X}_a \subseteq \mathcal{P}$ of predicates, the reduced set $\mathcal{C}_{\mathcal{X}_a}$ of clauses is defined as the set of all the clauses of $\mathcal{C}$ after replacing all occurrences of $P(a)$ in all the clauses of $\mathcal{C}$ with *true* if $P \in \mathcal{X}_a$ and (replacing all occurrences of $P(a)$ in every clause of $\mathcal{C}$) with *false* if $P \notin \mathcal{X}_a$.

We also remove the trivial clauses after the reduction process. Note that the reduced set can contain an empty clause if there is a clause in $\mathcal{C}$ of the form

$$P_1(a) \wedge \cdots \wedge P_n(a) \to Q_1(a) \vee \cdots \vee Q_m(a)$$

with $P_i \in \mathcal{X}_a$ and $Q_j \notin \mathcal{X}_a$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. But this means that the choice of the set $\mathcal{X}_a$ is wrong(or $\mathcal{C}$ has no finite Herbrand Model).

Lemma 2. *A set $\mathcal{C}$ of normalized monadic clauses has a finite Herbrand model iff for a set $\mathcal{X}_a$ of predicates the reduced clause set $\mathcal{C}_{\mathcal{X}_a}$ has a finite Herbrand model $\mathcal{H}$ with $a \in P^{\mathcal{H}}$ iff $P \in \mathcal{X}_a$ for all $P \in \mathcal{P}$.*

Proof. If $\mathcal{H}$ is a finite Herbrand model for $\mathcal{C}_{\mathcal{X}_a}$, then $\mathcal{H}$ is a model for all clauses in $\mathcal{C}$ without any ground atom as these clauses are not changed in $\mathcal{C}_{\mathcal{X}_a}$. Now for the *mixed* and *ground* clauses let us consider the following clause of $\mathcal{C}$:

$$c = P_1(t_1) \wedge \cdots \wedge P_n(t_n) \rightarrow P_{n+1}(t_{n+1}) \vee \cdots \vee P_{n+m}(t_{n+m})$$

We have two cases, that is a ground atom can occur either on the right-hand side of the clause or left-hand side. Let c' be the clause reduced from c.

For the first case, let for some $1 \leq i \leq n$ the atom $P_i(t_i)$ be ground, i.e. $t_i = a$. Now there can be again two cases either $P \in \mathcal{X}_a$ which means it is replaced by $\top$ in c' but as $\mathcal{H}$ satisfies the c' it also satisfies c because as $a \in P_i^{\mathcal{H}}$ so the atom $P_i(t_i)$ is true and does not affect the truth value of the premise of clause. And if $P \notin \mathcal{X}_a$ then $P_i(t_i)$ is false under $\mathcal{H}$ and c is satisfied by $\mathcal{H}$ trivially, as this makes the premise of the clause false.

For the second case, let for some $n < i \leq n + m$ the atom $P_i(t_i)$ be ground, i.e. $t_i = a$. Here we also have the same two cases, i.e. either $P \in \mathcal{X}_a$ and the atom $P_i(t_i)$ is replaced by true in c', which makes the whole clause true, so it is satisfied by $\mathcal{H}$ trivially. As $a \in P_i^{\mathcal{H}}$, the atom $P_i(t_i)$ is true under $\mathcal{H}$, thus c is also satisfied by $\mathcal{H}$. For the other case if $P \notin \mathcal{X}_a$, then it is replaced by false in c', but as $\mathcal{H}$ satisfies the updated clause, this means that either the premise of the clause is not satisfied, in which case $\mathcal{H}$ also satisfies c as the premise is not changed during reduction. But if the premise is true, then there must exists$j \neq i$ where $n < j \leq n + m$ such that atom $P_j(t_j)$ is true under $\mathcal{H}$ hence c is also satisfied by $\mathcal{H}$ as this makes the consequence of c true.

For the other direction let $\mathcal{H}$ be a finite Herbrand model for $\mathcal{C}$. We choose $\mathcal{X}_a = \{P \mid a \in P^{\mathcal{H}}\}$. We show that $\mathcal{H}$ is a model for the reduced clause set $\mathcal{C}_{\mathcal{X}_a}$. We again consider the clause c, the reduced clause c' and all the cases for a ground atom.

For the first case, let for some $1 \leq i \leq n$ the atom $P_i(t_i)$ be ground, i.e. $t_i = a$. Now there can be two cases. Either the ground atom $P_i(a)$ is true under $\mathcal{H}$ i.e. $a \in P_i^{\mathcal{H}}$. Then by choice of $\mathcal{X}_a$ we know that $P \in \mathcal{X}_a$ and $P_i(a)$ will be replaced by $\top$ in c' which does not affect the truth value of the premise of the clause so as c is satisfied under $\mathcal{H}$ so is c'. And if $P_i(a)$ is false under $\mathcal{H}$ we know $P \notin \mathcal{X}_a$ then $P_i(a)$ is replaced by false making the whole premise false, thus making c' a tautology under any interpretation, in particular $\mathcal{H}$.

For the second case, let for some $n < i \leq n + m$ the atom $P_i(t_i)$ be ground, i.e. $t_i = a$. Here we have again the same two cases, i.e. either $P_i(a)$ is true under $\mathcal{H}$ i.e. $a \in P_i^{\mathcal{H}}$ thus, we know that $P \in \mathcal{X}_a$ and $P_i(a)$ will

be replaced by $\top$ in c' which makes the whole clause true, so it is satisfied by $\mathcal{H}$ trivially. If $P_i(a)$ is false under $\mathcal{H}$ we know $P \notin \mathcal{X}_a$, and it is replaced by false in c', which does not change the truth value of so if $\mathcal{H}$ satisfies the c, it also satisfies c'.

□

Example 2.2.2.

Consider the following normalized set of monadic clauses.

$$\begin{aligned}\mathcal{C} := \{&\top \to P_1(a) \vee P_2(a),\ P_2(a) \wedge P_3(x) \to P_4(a)\\ &P_1(a) \wedge P_3(x) \to P_4(x) \vee P_5(x),\ P_2(a) \to P_6(a) \vee P_7(a),\\ &P_4(x) \wedge P_2(x) \to P_7(x)\}\end{aligned}$$

Let $\mathcal{X} = \{P_1, P_6\}$ be the set of predicates for which we want to reduce $\mathcal{X}$. After reduction the reduced set would be as follows.

$$\mathcal{C}_\mathcal{X} := \{P_3(x) \to P_4(x) \vee P_5(x),\ P_4(x) \wedge P_2(x) \to P_7(x)\}$$

Note that the first two clauses and the fourth were removed as they became trivially true after replacement.

2.3 Possibilities

We will use the notion of possibility for a term t and a set $\mathcal{X}$ of predicates as a set $\mathcal{Y}$ of predicates for which we need to add t in all the predicates of $\mathcal{Y}$ under the interpretation $\mathcal{I}$ if t is already in all the predicates of $\mathcal{X}$ under $\mathcal{I}$, to make $\mathcal{I}$ a model.

Example 2.3.1.

Consider the following flat monadic clauses.

$$\begin{aligned}\mathcal{C}_2 := \{&\top \to P_1(a) \vee P_2(a), P_2(x) \to P_4^f(x)\\ &P_1(x) \to P_4(x) \vee P_5(x), P_1(x) \to P_6^g(x) \vee P_7^g(x),\\ &P_5(x) \wedge P_1(x) \to P_6(x)\}\end{aligned}$$

For $\mathcal{X} = \{P_1, P_5\}$ one possibility could be $\{P_1, P_5, P_7^g, P_6\}$, and $\{P_1, P_5, P_4, P_6\}$ another one.

Chapter 3

Finding Finite Herbrand Models

In this chapter we present an algorithm to decide the existence of a finite Herbrand model for a set of monadic clauses.

3.1 Exploring Possibilities

The idea of the following algorithm is that we want to explore all minimal finite models. Normalization of clauses helps us isolate different terms because we can devise a strategy to treat the clauses which deal with a particular term (the flat clauses) and the clauses dealing with terms of different length (increasing and decreasing clauses) differently. As we are interested in finite Herbrand models, we explore the potential models for all the ground terms recursively, starting from the shortest ground term a as that is the only constant in our logic. Then we increase the length of the term for some function symbol with each recursive step in the algorithm. We calculate a set of possibilities $\mathcal{L}$ at each level representing parts of all the potential models. We expand the possibilities in $\mathcal{L}$ first using the flat clauses and then check if we need to add terms of higher length in the potential model using the increasing clauses. Let us take a look at an example on how we want to explore the potential model.

Example 3.1.1.

Consider the following (simple) set of reduced monadic clauses

$$\begin{aligned}\mathcal{C}_3 := \{&A(x) \to P(x) \vee Q(x),\ P(x) \to R^f(x),\\ &A(x) \wedge P(x) \to S(x),\ R(x) \to T(x),\\ &Q(x) \to U^g(x),\ U(x) \to V(x),\end{aligned}$$

$$A(f(x)) \to A^f(x),\ A^f(x) \to A(f(x)),$$
$$R(f(x)) \to R^f(x),\ R^f(x) \to R(f(x)),$$
$$T(f(x)) \to T^f(x),\ T^f(x) \to T(f(x)),$$
$$U(g(x)) \to U^g(x),\ U^g(x) \to U(g(x))\}$$

with the set

$$\mathcal{D}(\mathcal{C}_3) = \{(A, f), (R, f), (T, f)\}$$

and

$$\mathcal{X}_a = \{A^f\}$$

And we want to explore all the potential models for $\mathcal{C}_3$. We know from the selection of $\mathcal{X}_a$ that $a \in A^{f^{\mathcal{I}}}$, where $\mathcal{I}$ is a potential model. We initialize the set $\mathcal{L}_a$ of possibilities with $\mathcal{X}_a$, i.e. $\mathcal{L}_a = \{\{A^f\}\}$. As this possibility needs to satisfy the increasing clause $A^f(x) \to A(f(x))$, this enforces that $f(a) \in A^{\mathcal{I}}$ and we start looking for possibilities for $\mathcal{X}_{f(a)} = \{A\}$. We initialize the set of possibilities as:

$$\mathcal{L}_{f(a)} = \{\{A\}\}$$

We expand $\mathcal{L}_{f(a)}$ for the first clause $A(x) \to P(x) \vee Q(x)$, because for an interpretation $\mathcal{I}$ to be the model of this clause then for any term $t \in A^{\mathcal{I}}$ either $t \in P^{\mathcal{I}}$ or $t \in Q^{\mathcal{I}}$. We update the set of possibilities as follows:

$$\mathcal{L}_{f(a)} = \{\{A, P\}, \{A, Q\}\}$$

Then we go on expanding the first possibility in $\mathcal{L}_{f(a)}$ for the second, third and fifth clause of $\mathcal{C}_3$ and update $\mathcal{L}_{f(a)}$ as

$$\mathcal{L}_{f(a)} = \{\{A, P, R^f, S\}, \{A, Q, U^g\}\}$$

Now we do not need to expand $\mathcal{L}_{f(a)}$ any more for the flat clauses. But considering the potential model $\mathcal{I}$ for the first possibility, as we have $f(a) \in R^{f^{\mathcal{I}}}$ and to satisfy the increasing clause $R^f(x) \to R(f(x))$ we need to start looking for possibilities for $\mathcal{X}_{f(f(a))} = \{R\}$. We repeat the procedure by initializing $\mathcal{L}_{f(f(a))}$ as $\{\{R\}\}$ and expanding the possibilities using flat clauses. After expansion for the clause $R(x) \to T(x)$, the set looks as follows:

$$\mathcal{L}_{f(f(a))} = \{\{R, T\}\}$$

There is no possibility in $\mathcal{L}_{f(f(a))}$ with a complex predicates, hence we do not need to increase the length of the term so our recursion stops and returns. On the return we do need to be careful about the decreasing clauses because if for any interpretation $\mathcal{I}$ to be the model it also needs to satisfy the decreasing clauses i.e. for any $(P, f) \in \mathcal{D}(\mathcal{C}_3)$ and for all ground terms t, if $f(t) \in P^{\mathcal{I}}$, then $t \in P^{f^{\mathcal{I}}}$ because of the decreasing clause $T(f(x)) \to T^f(x)$. And in our example, as we have the predicate T in the only possibility in $\mathcal{L}_{f(f(a))}$,

and $(T, f) \in \mathcal{D}(\mathcal{C}_3)$, so we need to expand the possibility $\{A, P, R^f, S\}$ with T^f, and update $\mathcal{L}_{f(a)}$ as:

$$\mathcal{L}_{f(a)} = \{\{A, P, R^f, S, T^f\}, \{A, Q, U^g\}\}$$

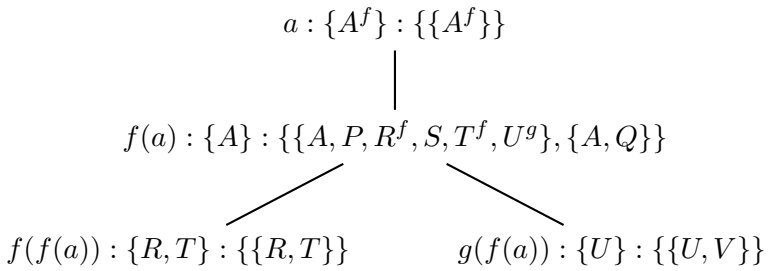
We now start expanding the second possibility in $\mathcal{L}_{f(a)}$ but, and we expand the possibilities for term $g(f(a))$ and the set $\mathcal{X}_{g(f(a))} = \{U\}$ similarly resulting in the following set of possibilities.

$$\mathcal{L}_{g(f(a))} = \{\{U, V\}\}$$

After that we do not need to expand anymore for the term $f(a)$ and return to expanding the possibilities for a. But as there is no $(P, f) \in \mathcal{D}(\mathcal{C}_3)$ for which we need to expand $\mathcal{L}_a$ our search for potential models ends with two possible models of $\mathcal{C}_3$ as follows:

$$\mathcal{H}_1 := \{A^f(a), A(f(a)), P(f(a)), R^f(f(a)), S(f(a)), T^f(f(a)), R(f(f(a))), T(f(f(a)))\}$$
$$\mathcal{H}_2 := \{A^f(a), A(f(a)), Q(f(a)), U^g(f(a)), U(g(f(a))), V(g(f(a)))\}$$

Consider the following diagram to illustrate the recursive exploration of possibilities. Each node is of the form $t : \mathcal{X}_t : \mathcal{L}_t$ where t is the ground term for which is being explored.



3.2 Finding Loops

As we are interested in finiteness of the models, we need to detect infinite loops and stop the recursive search. A model is infinite if there is a loop within the set of clauses with an ever increasing term length.

Example 3.2.1.

One of the simplest example to enforce an infinite model would be a set of monadic clauses representing the set of natural numbers as follows:

$$\mathcal{C}_4 := \{\top \to Natural(a),\ Natural(x) \to Natural(succ(x))\}$$

where a is representing the number 0. It is clear that $\mathcal{C}_4$ only has one infinite model $\mathcal{H}$ where $Natural^{\mathcal{H}} = \{a, succ(a), succ(succ(a)), \dots\}$ represents the infinite set of natural numbers. This happens because there

is a loop in the second clause and for every natural number i.e. $t \in Natural^{\mathcal{H}}$, we have to add the successor $succ(t)$ of the number in $Natural^{\mathcal{H}}$.

In our algorithm we try to detect these kind of loops. To do this we keep track of the working set $\mathcal{W}$, which stores all the sets $\mathcal{X}_t$ for all the terms t that we are exploring in the current recursive branch. Consider the example above, Following is the normalized set for $\mathcal{C}_4$:

$$\begin{aligned}\mathcal{C}_4' := \{&\top \to Natural(a),\\ &Natural(x) \to Natural^{succ}(x),\\ &Natural(succ(x)) \to Natural^{succ}(x),\\ &Natural^{succ}(x) \to Natural(succ(x))\}\end{aligned}$$

with the set

$$\mathcal{D}(\mathcal{C}_4) = \{(Natural, succ)\}$$

We want to explore the potential finite Herbrand models for $\mathcal{C}_4$, so we start with the term a and the set $\mathcal{X}_a = \{Natural\}$ and, because of the second clause in $\mathcal{C}_4$, we need to expand the possibility for the predicate $Natural^{succ}$ and update $\mathcal{X}_a = \{Netural, Natural^{succ}\}$. As we want to satisfy the increasing clause we need to start the search for possibilities for term $succ(a)$ and the set $\mathcal{X}_{succ(a)} = \{Natural\}$ and we need to expand the possibility with $Natural^{succ}$ again which makes $\mathcal{X}_{succ(a)} = \{Natural, Natural^{succ}\}$ and we need to again make an other recursive call for term $succ(succ(a))$ and the cycle continues infinitely. But to detect and stop, we keep track of all the $\mathcal{X}_t$s in the current working branch and stop whenever the same set is produced again in the same working branch. So in our algorithm we will not continue infinitely because we will add $\mathcal{X}_a$ in the working set $\mathcal{W}$ before looking for possibilities for term $succ(a)$ and, as $\mathcal{X}_a = \mathcal{X}_{succ(a)}$, we stop the process and mark $\{Natural, Natural^{succ}\}$ as a bad possibility(which is described in the next section) and as this was the only possibility in $\mathcal{X}_a$ we decide that there is no finite model for $\mathcal{C}_4$.

3.3 Bad Possibilities

One other problem that we need to deal with while expanding the possibilities is that we do not want to get stuck in a cycle where a possibility is identified as a bad possibility and then the same one is added in the next cycle of saturation again by some clause.

Example 3.3.1.

Consider the following set of monadic clauses:

$$\mathcal{C}_5 := \{\top \to P(a),$$
$$P(x) \to Q(x) \vee R(x),$$
$$R(x) \to \bot\}$$

In this example when we want to look at the possibilities for term a we start with $\mathcal{X}_a = \{\{P\}\}$ and then we expand $\mathcal{X}_a$ for the second clause and update $\mathcal{X}_a$ as

$$\mathcal{X}_a = \{\{P, Q\}, \{P, R\}\}$$

but because of the third clause in $\mathcal{C}_4$ we know that possibility $\{P, R\}$ is bad as R leads to $\bot$, and is removed from $\mathcal{X}_a$. But as we still have P in the first possibility, we can expand this possibility again for the second clause to add a new possibility $\{P, Q, R\}$ updating $\mathcal{X}_a$ as.

$$\mathcal{X}_a = \{\{P, Q\}, \{P, Q, R\}\}$$

We know that this possibility will also be identified as bad because of R but we will get stuck in an infinite cycle of removing this possibility and adding it again. That is why we keep track of all the possibilities identified as bad so that they might never be produced again.

Following is the recursive algorithm to get possibilities.

Algorithm 1: getPossibilities(t , $\mathcal{X}$, $\mathcal{W}$, $\mathcal{C}$).

Input : A ground term t, a set $\mathcal{X}$ of predicates, current working set $\mathcal{W}$ of sets of predicates and a normalized set $\mathcal{C}$ of *MFOL+* clauses

Output: A set of possibilities for t

$\mathcal{L}_0 \leftarrow \{\mathcal{X}\}, k \leftarrow 0, \mathcal{B} \leftarrow \emptyset$

repeat

$k \leftarrow k + 1$

$\mathcal{L}_k \leftarrow \mathcal{L}_{k-1}$

for all $P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow Q_1(x) \vee \cdots \vee Q_m(x) \in \mathcal{C}$ **do**

for all $\mathcal{Y} \in \mathcal{L}_k$ such that $\{P_1, \ldots, P_n\} \subseteq \mathcal{Y}$ and there is no $1 \leq i \leq m$ such that $Q_i \in \mathcal{Y}$ **do**

$\mathcal{L}_k \leftarrow ((\mathcal{L}_k \setminus \{\mathcal{Y}\}) \cup \{\mathcal{Y} \cup \{Q_l\} \mid l \in \{1, \ldots, m\}\}) \setminus \mathcal{B}$

$\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathcal{Y}\}$

for all $\mathcal{Y} \in \mathcal{L}_k$ and $f \in \mathcal{F}$ with $\mathcal{Y}_f = \{P \mid (P, f) \in \mathcal{D}(\mathcal{C}), P^f \in \mathcal{Y}\}$ such that $\mathcal{Y}_f \neq \emptyset$ **do**

$\mathcal{L}_k \leftarrow \mathcal{L}_k \setminus \{\mathcal{Y}\}$

if $\mathcal{Y}_f \notin \mathcal{W}$ **then**

$\mathcal{L}_{f(t)} \leftarrow$ getPossibilities$(f(t), \mathcal{W} \cup \{\mathcal{Y}_f\}, \mathcal{C})$

$\mathcal{L}_k \leftarrow (\mathcal{L}_k \cup \{\mathcal{Y} \cup \{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{Z}\} \mid \mathcal{Z} \in \mathcal{L}_f\}) \setminus \mathcal{B}$

if $\{\mathcal{Y}\} \notin \mathcal{L}_k$ **then**

$\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathcal{Y}\}$

until $\mathcal{L}_k = \mathcal{L}_{k-1}$

return $\mathcal{L}_k$

The main algorithm that decides the existence of a finite Herbrand model is very straight forward. It non-deterministically choses a set $\mathcal{X}_a$ of predicates and makes the first call to *getPossibilities* for the term a, the chosen set $\mathcal{X}_a$, the initial working set $\mathcal{W} = \{\mathcal{X}_a\}$ and the set $\mathcal{C}_{\mathcal{X}_a}$ of monadic clauses reduced for $\mathcal{X}_a$. The algorithm returns true if the possibilities set $\mathcal{L}_a$ returned by *getPossibilities* contains the set $\mathcal{X}_a$ (meaning the choice for $\mathcal{X}_a$ was correct and there is a finite Herbrand model for $\mathcal{C}$) and false otherwise.

Following is the main algorithm.

Algorithm 2:

Input : A normalized set $\mathcal{C}$ of Monadic clauses

Output: True if the set $\mathcal{C}$ has a finite model and false otherwise

(non-deterministically) guess a set $\mathcal{X}_a$ of predicates.

$\mathcal{L}_a \leftarrow$ getPossibilites$(a, \mathcal{X}_a, \{\mathcal{X}_a\}, \mathcal{C}_{\mathcal{X}_a})$

if $\mathcal{X}_a \in \mathcal{L}_a$ **then**

  return true

else

  return false

3.4 Proofs

3.4.1 Termination

Lemma 3. *Algorithm 2 always terminates.*

Proof. Algorithm 2 consists of a non deterministic choice of the set $\mathcal{X}_a$ and then calling *getPossibilities* with $\mathcal{X}_a$ and the reduced set $\mathcal{C}_{\mathcal{X}_a}$ to get the set of possibilities $\mathcal{L}_a$ and checking if $\mathcal{X}_a \in \mathcal{L}_a$. From the Definition 1.1 it is trivial to see that the procedure for calculating the reduced clause set $\mathcal{C}_{\mathcal{X}_a}$ is terminating for any set of clauses and a set of predicates in particular $\mathcal{C}$ and $\mathcal{X}_a$. For the termination proof of Algorithm 1 the only thing that we need to prove is the termination of *getPossibilities*.

The procedure *getPossibilities* is recursive in nature, so for termination of the procedure we need to prove that there are only finitely many recursive calls and all the loops in a call are terminating. Every call to the procedure has the set $\mathcal{W}$ of sets of predicates which capture the working predicate sets in the current recursive call. The procedure does not make a new recursive call if the set $\mathcal{Y}^f$ is already part of the working set $\mathcal{W}$. As there can only be $\mid 2^{\mathcal{P}} \mid$ such sets in $\mathcal{W}$ before a set is repeated, this means there can only be maximum of $\mid 2^{\mathcal{P}} \mid$ recursive calls of *getPossibilities*.

We also need to prove that there are no infinite loops in any of the recursive calls to *getPossibilities*. The only loop that is of concern is the outermost loop to check the saturation of the set $\mathcal{L}_k$ of possibilities. All the inner loops are bounded by the set of predicates $\mathcal{P}$, the set $\mathcal{L}_k$ or the set of functions $\mathcal{F}$, and all of

these sets are finite, hence all these loops are bound to terminate after finite time.

It is not that obvious to see that the outermost loop will always terminate after finite time as the procedure adds new possibilities as well as remove the bad possibilities from the set $\mathcal{L}_k$. But we keep track of all the bad possibilities that are removed and they are never added to $\mathcal{L}_k$ even if they are discovered again. Which means that the set $\mathcal{L}_k \cup \mathcal{B}$ is always increasing. As the sets $\mathcal{L}_k$ and $\mathcal{B}$ are disjoint and there are only $\mid 2^{\mathcal{P}} \mid$ possibilities it is clear that after at most $2 \mid 2^{\mathcal{P}} \mid$ loops $\mathcal{L}_k \cup \mathcal{B}$ will saturate (and in particular $\mathcal{L}_k$). □

3.4.2 Correctness

Lemma 4. *(Soundness) If Algorithm 2 returns true, then $\mathcal{C}$ has a finite Herbrand model.*

Proof. We create the finite Herbrand model $\mathcal{H}$ by using a variant of *getPossibilities* which does not add all the possibilities for a term but, instead returns one of the possibilities. As Algorithm 1 returns true, there exists a set $\mathcal{X}_a$ of predicates such that the first call to *getPossibilities* returns the set $\mathcal{L}_a$ where $\mathcal{X}_a \in \mathcal{L}_a$ so we choose the possibility $\mathcal{X}_a$ in the first call. We create the model by collecting all the predicates P in the chosen possibility such that $t \in P^{\mathcal{H}}$ for all the recursive calls of *getPossibilities*. From the termination condition of every call to *getPossibilites* we know that the chosen possibility $\mathcal{Y}$ is fully saturated i.e. there is no flat clause $P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow Q_1(x) \vee \cdots \vee Q_m(x) \in \mathcal{C}$ such that $\{P_1, \ldots, P_n\} \subseteq \mathcal{Y}$ and $\{Q_1, \ldots, Q_n\} \not\subseteq \mathcal{Y}$ and also for all $f \in \mathcal{F}$ with $\mathcal{Y}_f = \{P \mid (P, f) \in \mathcal{D}(\mathcal{C}), P^f \in \mathcal{Y}\}$ there exists a possibility $\mathcal{Z} \in \mathcal{L}_f$ such that $\{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{Z}\} \subseteq \mathcal{Y}$. We know that the model is finite, as by Lemma 1 the Algorithm returns after a finite time. □

Lemma 5. *(Completeness) If $\mathcal{C}$ has a* finite Herbrand model*, then Algorithm 2 returns true.*

Proof. Let $\mathcal{H}$ be a finite Herbrand model for $\mathcal{C}$. We choose the set $\{P \mid a \in P^{\mathcal{H}}\}$ as $\mathcal{X}_a$. Now to prove that the initial call to *getPossibilities* will return the set $\mathcal{L}_a$ with $\mathcal{X}_a \in \mathcal{L}_a$, we consider an alternate procedure for *getPossibilities* as defined below where we remove the working set and we also add a non-deterministic choice of the possibility $\mathcal{Y}$ instead of the loop in the original algorithm. The working set $\mathcal{W}$ in the original algorithm only ensures the termination of the algorithm which is not of concern for this proof, we assume that the $\mathcal{H}$ is minimal which means we can drop $\mathcal{W}$ without affecting the structure of the algorithm. Also by adding the non-deterministic choice for the possibility $\mathcal{Y} \in \mathcal{L}_k$ we ensure that the algorithm always picks

the right possibility for the model $\mathcal{H}$. Thus our claim holds for the original algorithm if the set returned by the *getPossibilities*$(a, \mathcal{X}_a, \mathcal{C}_{\mathcal{X}_a})$ contains the possibility $\mathcal{X}_a$.

Following is the alternate procedure for *getPossibilities*.

Algorithm 3: getPossibilities(t , $\mathcal{X}$, $\mathcal{C}$).

Input : A term t of sets of predicates, a set $\mathcal{X}$ of predicates and a normalized set $\mathcal{C}$ of Monadic clauses

Output: A set of possibilities $\mathcal{L}$

1: $\mathcal{L}_0 \leftarrow \{\mathcal{X}\}, k \leftarrow 0$
2: **repeat**
3: $k \leftarrow k + 1$
4: $\mathcal{L}_k \leftarrow \mathcal{L}_{k-1}$
5: */* Saturation step */*
6: **for all** $P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow Q_1(x) \vee \cdots \vee Q_m(x) \in \mathcal{C}$ **do**
7: **for all** $\mathcal{Y} \in \mathcal{L}_k$ such that $\{P_1, \ldots, P_n\} \subseteq \mathcal{Y}$ and there is no $1 \leq i \leq m$ such that $Q_i \in \mathcal{Y}$ **do**
8: $\mathcal{L}_k \leftarrow (\mathcal{L}_k \setminus \{\mathcal{Y}\}) \cup \{\mathcal{Y} \cup \{Q_l\} \mid l \in \{1, \ldots, m\}\}$
9: **choose** $\mathcal{Y} \in \mathcal{L}_k$
10: **for all** $f \in \mathcal{F}$ with $\mathcal{Y}_f = \{P \mid (P, f) \in \mathcal{D}(\mathcal{C}), P^f \in \mathcal{Y}\}$ such that $\mathcal{Y}_f \neq \emptyset$ **do**
11: */* Incrementation Step */*
12: $\mathcal{L}_f \leftarrow$ getPossibilities$(\mathcal{Y}_f, f(t), \mathcal{C})$
13: */* Decrementation Step */*
14: $\mathcal{L}_k \leftarrow (\mathcal{L}_k \setminus \{\mathcal{Y}\}) \cup \{\mathcal{Y} \cup \{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{Z}\} \mid \mathcal{Z} \in \mathcal{L}_f\}$
15: **until** $\mathcal{L}_k = \mathcal{L}_{k-1}$
16: **return** $\mathcal{L}_k$

We define the *call stack* **S** as a stack of frames, where each frame is a tuple of the form $(\mathcal{X}, t, \mathcal{L}, l)$, which represents a function call to *getPossibilites*, where $\mathcal{X}$ and t are the parameters of the call, $\mathcal{L}$ is a local variable representing the current $\mathcal{L}_k$, and l is the current execution step within the call. The frame at the top of **S** represents the last recursive call of the procedure. For example a call stack after n recursive calls to the procedure is as follows.

$$(\mathcal{X}_0, t_0, \mathcal{L}_0, l_0), (\mathcal{X}_1, t_1, \mathcal{L}_1, l_1), \ldots, (\mathcal{X}_n, t_n, \mathcal{L}_n, l_n)$$

where $(\mathcal{X}_n, t_n, \mathcal{L}_n, l_n)$ is the top frame in this case. Every step in the procedure can update the local

variables of the top frame, push a new frame onto $\mathbf{S}$ for a new call of the procedure, or pop a frame from $\mathbf{S}$ in case the procedure returns. From now onward we will refer to call stack as stack.

We define the *execution* of a procedure $\mathbf{P}$ as a sequence of above mentioned stacks $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \ldots, \mathbf{S}_m$, where each stack $\mathbf{S}_{i+1}$ is the result of execution of the l^{th} step of $\mathbf{P}$ on stack $\mathbf{S}_i$ where l is the from the top frame of $\mathbf{S}_i$ and $1 \leq i \leq m-1$.

We already know that the original algorithm terminates from Lemma 1. To prove that there exists a possibility for $\mathcal{X}_a$ when the first call returns we prove for the alternate algorithm that for every frame $(\mathcal{X}, t, \mathcal{L}, l)$ of a stack in the execution of $getPossibilites(\mathcal{X}_a, a, \mathcal{C}_{\mathcal{X}_a})$ there always exists a possibility $\mathcal{Y} \in \mathcal{L}$ such that $t \in P^{\mathcal{H}}$ for all $P \in \mathcal{Y}$, in particular for the first frame $(\mathcal{X}_a, a, \mathcal{L}, l)$.

We prove this by induction over the execution $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \ldots, \mathbf{S}_m$ of the procedure call $getPossibilites(\mathcal{X}_a, a, \mathcal{C}_{\mathcal{X}_a})$. For the base case, the call stack $\mathbf{S}_1$ contains the only frame $(\mathcal{X}_a, a, \{\mathcal{X}_a\}, 1)$. The claim is trivially true as we know by construction of $\mathcal{X}_a$ that $a \in P^{\mathcal{H}}$ for all $P \in \mathcal{X}_a$.

For the purpose of simplicity we assume the initialization of local variables $\mathcal{L}_0$ and k are done when the frame for a procedure call is added to the stack. The rest of the procedure can be divided into three steps. The first step is *saturation*, where only the local variable $\mathcal{L}_k$ is updated by saturating the set of possibilities with flat clauses. The second step is *incrementation*, where a new non empty set $\mathcal{Y}_f = \{P \mid (P, f) \in \mathcal{D}(\mathcal{C}), P^f \in \mathcal{Y}\}$ is found for a possibility $\mathcal{Y} \in \mathcal{L}_k$ and a function symbol f such that there is no shortcut calculated for $\mathcal{Y}^f$, and we make a recursive call to *getPossibilities* with the set $\mathcal{Y}_f$ and term $f(t)$. i.e adding new frame corresponding to it. The third step *decrementation* is to saturate the possibility $\mathcal{Y} \in \mathcal{L}_k$ if such $\mathcal{Y}_f$ is found by creating new possibilities after adding the new predicates of the form P^f for every possibility calculated in $\mathcal{R}_l$ for $\mathcal{Y}_f$. The *incrementation* and *decrementation* steps are performed for all the function symbols $f \in \mathcal{F}$ for the chosen possibility $\mathcal{Y}$.

For the induction step we make the case distinction for all three steps of the procedure. For the first case let l be the saturation step. This step only updates the local variables, specifically $\mathcal{L}$ for the possibility $\mathcal{Y}$ if there is a clause of the form $P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow Q_1(x) \vee \cdots \vee Q_m(x)$ in the reduced clause set $\mathcal{C}$ such that $\{P_1, \ldots, P_n\} \subseteq \mathcal{Y}$ and $Q_j \notin \mathcal{Y}$ for $j \in \{n+1, \ldots, n+m\}$. The algorithm removes $\mathcal{Y}$ and adds all the new possibilities $\mathcal{Z} = \{\mathcal{Y} \cup \{Q_l\} \mid l \in \{1, \ldots, m\}\}$ into $\mathcal{L}$. As we know from induction hypothesis that there is a $\mathcal{Y} \in \mathcal{L}$ such that $t \in P^{\mathcal{H}}$ for all $P \in \mathcal{Y}$, and for $\mathcal{H}$ to satisfy the clause $P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow Q_1(x) \vee \cdots \vee Q_m(x)$ there must exist a $j \in \{1, \ldots, m\}$ such that $t \in Q_j^{\mathcal{H}}$, and we also know that for j the possibility $\mathcal{Y}' = \mathcal{Y} \cup \{Q_j\}$ is part of $\mathcal{Z}$ added into $\mathcal{L}$, after the saturation step there

exists a possibility $\mathcal{Y}' \in \mathcal{L}$ such that $t \in P^{\mathcal{H}}$ for all $P \in \mathcal{Y}'$.

For the second case, let l be the incrementation step. This step creates a new call to the procedure if there is no shortcut available in $\mathcal{R}$ and, otherwise does not change anything. A new frame $(\mathcal{Y}_f, f(t), \{\mathcal{Y}_f\}, 0)$ is added into the stack $\mathbf{S}_{i+1}$ for the possibility $\mathcal{Y}$ with $\mathcal{Y}_f = \{P \mid (P, f) \in \mathcal{D}(\mathcal{C}), P^f \in \mathcal{Y}\}$. Now as for the model $\mathcal{H}$, we know that $t \in P^{f\mathcal{H}}$ for all $P \in \mathcal{Y}_f$ and there is an increasing clause of the form $P^f(x) \to P(f(x))$ for every P^f, therefore $f(t) \in P^{\mathcal{H}}$ for every $P \in \mathcal{Y}_f$. Which proves our claim, as the stack $\mathbf{S}_{i+1}$ contains the possibility $\mathcal{Y}_f$ such that $f(t) \in P$ for all predicates $P \in \mathcal{Y}_f$.

For the third case, let l be the decrementation step. In this step the procedure call returns with the set of possibilities $\mathcal{L}_f$ calculated for the $\mathcal{Y}_f$. And the procedure removes $\mathcal{Y}$ and adds all the possibilities in $\{\mathcal{Y} \cup \{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{Z}\} \mid \mathcal{Z} \in \mathcal{L}_f\}$. We know from the induction hypothesis that before the recursive call returns there must exist a possibility $\mathcal{Z}$ in the returned set $\mathcal{L}_f$ of the recursive call frame such that $f(t) \in P^{\mathcal{H}}$ for all $P \in \mathcal{Z}$. Now the algorithm updates the possibility $\mathcal{Y}$ by adding $\{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{Z}\}$, but as we know $f(t) \in P^{\mathcal{H}}$ for all $P \in \mathcal{Z}$ and there exists a decreasing clause of the form $P(f(x)) \to P^f(x)$ for all $P \in \mathcal{Z}$, then for the model $\mathcal{H}$ to satisfy these clauses we must have $t \in P^{f\mathcal{H}}$ for all $P \in \mathcal{Z}$ if $(P, f) \in \mathcal{D}(\mathcal{C})$, which proves the claim that after the execution of this step there exists the possibility $\mathcal{Y}' = \mathcal{Y} \cup \{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{Z}\}$ in $\mathcal{L}$ such that $t \in P^{\mathcal{H}}$ for all predicates $P \in \mathcal{Y}'$. And this concludes our induction step. □

Chapter 4

Optimizations and Variants

4.1 Shortcuts

We provide some optimizations to improve the best case of the algorithm by reusing some of the computation. We use the notion of shortcuts to save the possibilities calculated for some set $\mathcal{X}$ of predicates and for some term t in *getPossibilites*. As we are dealing with non ground clauses after the reduction, in most cases we can reuse the possibilities calculated for $\mathcal{X}$ with other terms as well.

Example 4.1.1.

Consider the following set of reduced clauses:

$$\begin{aligned}\mathcal{C}_6 := \{&P(x) \to Q^f(x), P(x) \to R^g(x),\\ &Q(x) \to S(x) \vee T(x), S(x) \to P(x),\\ &R(x) \to Q^g(x)\}\end{aligned}$$

with $\mathcal{D}(\mathcal{C}_6) = \{(Q, f), (R, g), (Q, g)\}$

The algorithm starts with $\mathcal{X}_a = \{P\}$ and expands the possibilities set $\mathcal{L}_a$ for the flat clauses as

$$\mathcal{L}_a = \{\{P, Q^f, R^g\}\}$$

Then we make another call to the procedure looking for $\mathcal{X}_{f(a)} = \{Q\}$ and expanding the set $\mathcal{L}_{f(a)}$ of possibilities using the flat clauses as

$$\mathcal{L}_{f(a)} = \{\{Q, S, P\}, \{Q, T\}\}$$

The algorithm returns as these are fully saturated. Then the algorithm starts looking for possibilities for

$\mathcal{X}_{g(a)} = \{R\}$ and expands the possibilities to

$$\mathcal{L}_{g(a)} = \{\{R, Q^g, P\}\}$$

And then algorithm has to find the possibilities for $\mathcal{X}_{g(g(a))} = \{Q\}$ but as it is the same as $\mathcal{X}_{f(a)}$ the possibilities calculated by the algorithm would be the same as $\mathcal{L}_{f(a)}$

$$\mathcal{L}_{g(g(a))} = \{\{Q, S, P\}, \{Q, T\}\} = \mathcal{L}_{f(a)}$$

we could have saved this computation if we had saved already calculated possibilities for $\mathcal{X}_{f(a)}$.

We propose to save the computed set of possibilities for a set of predicates as long as they are valid, because we can not always reuse previously calculated possibilities as there might be the case that some possibility is removed because of a loop found due to the working set $\mathcal{W}$ but that possibility might be not bad for some other working set $\mathcal{W}'$.

Example 4.1.2.

Consider the following set of reduced clauses:

$$\begin{aligned}\mathcal{C}_7 := \{&P(x) \to Q^f(x), P(x) \to R^g(x),\\ &Q(x) \to S^g(x) \vee T(x), S(x) \to Q^h(x),\\ &R(x) \to S^g(x)\}\end{aligned}$$

with $\mathcal{D}(\mathcal{C}_7) = \{(Q, f), (R, g), (S, g), (Q, h)\}$

The algorithm selects $\mathcal{X}_a = \{P\}$ and expands the possibilities $\mathcal{L}_a$ for flat clauses as

$$\mathcal{L}_a = \{\{P, Q^f, R^g\}\}$$

Then the algorithm makes a recursive call to *getPossbilities* looking for possibilities for $\mathcal{X}_{f(a)} = \{Q\}$ and expanding its possibilities using the flat clauses as

$$\mathcal{L}_{f(a)} = \{\{Q, S^g\}, \{Q, T\}\}$$

Then the algorithm call again *getPossbilities* for $\mathcal{X}_{g(f(a))} = \{S\}$ because of S^g in the first possibility and expands its possibilities as:

$$\mathcal{L}_{g(f(a))} = \{\{S, Q^h\}\}$$

As the only possibility $\mathcal{L}_{g(f(a))}$ has Q^h in it, algorithm needs to look for the possibilities for $\mathcal{X}_{h(g(f(a)))} = \{Q\}$ again but then it has found the loop as $\mathcal{X}_{h(g(f(a)))}$ is part of the working set $\mathcal{W}$ so algorithm removes the possibility $\{S, Q^h\}$ from $\mathcal{L}_{g(f(a))}$. As this is the only possibility in $\mathcal{L}_{g(f(a))}$ we remove the possibility $\{Q, S^g\}$

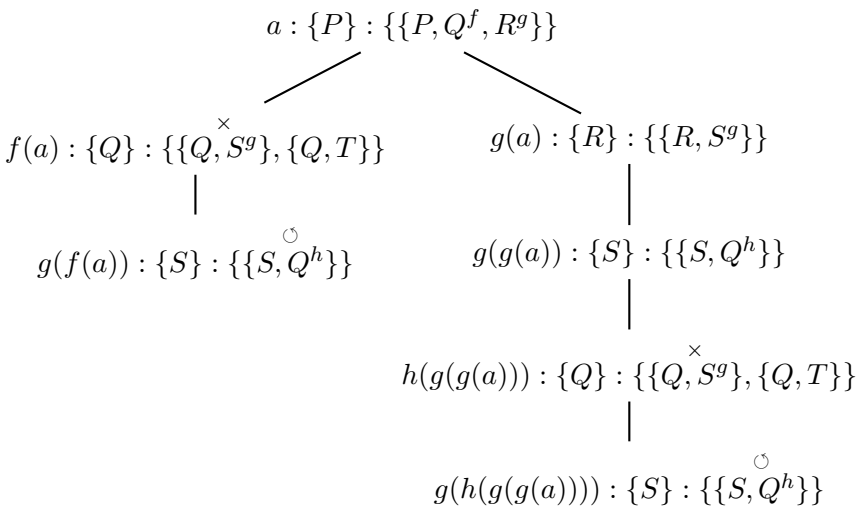
from $\mathcal{L}_{f(a)}$ as well so the only good possibility left in $\mathcal{L}_{f(a)}$ is $\{Q, T\}$ which is fully saturated and hence we return. Now algorithm searches possibilities for $\mathcal{X}_{g(a)} = \{R\}$ and adds S^g to update the set of possibilities as

$$\mathcal{L}_{g(a)} = \{\{R, S^g\}\}$$

The algorithm then looks for possibilities for $\mathcal{X}_{g(g(a))} = \{S\}$ which is the same as $\mathcal{X}_{g(f(a))}$ for which the algorithm did not find any good possibilities because of a loop. But now if we allow the algorithm to use already calculated possibilities for $\mathcal{X}_{g(f(a))}$ algorithm will return false deciding that there is no finite Herbrand model for $\mathcal{C}_7$ which is wrong because there is a finite Herbrand model i.e.

$$\mathcal{H} = \{P(a), Q^f(a), R^g(a), Q(f(a)), T(f(a)), R(g(a)), S^g(g(a)),$$
$$S(g(g(a))), Q^h(g(g(a))), Q(h(g(g(a)))), T(h(g(g(a))))\}$$

Consider the following diagram to illustrate the recursive execution of *getPossibilities* without using shortcuts. Each node represents the final values of the local variables $t : \mathcal{X}_t : \mathcal{L}_t$, the symbol $\times$ on top of a possibility means it was calculated and removed and the symbol $\circlearrowleft$ means the possibility was removed because the loop was found in the call.



We describe below an optimized algorithm for *getPossibilities* with the set $\mathcal{R}$ of shortcuts. We propose to save the set $\mathcal{L}$ of possibilities for a set $\mathcal{X}$ of predicates as a tuple $(\mathcal{X}, \mathcal{L}_{Heads}, \mathcal{L})$ in the set $\mathcal{R}$, where $\mathcal{L}_{Heads}$ is the set of all the sets of predicates for which the loop has been found during the calculation of this shortcut. To illustrate consider Example 5.1.2 again, we were not able to use the shortcut calculated for $\{S\}$ because of the loop due to $\{Q\}$ so we save the shortcut for $\{S\}$ as $(\{S\}, \{\{Q\}\}, \emptyset)$ and we remove this shortcut from $\mathcal{R}$ before returning from the call of *getPossibilities* with $\mathcal{X} = \{Q\}$. We remove the

shortcut $(\mathcal{X}', \mathcal{L}'_{Heads}, \mathcal{L}')$ before returning from a call with $\mathcal{X}$ if $\mathcal{X} \in \mathcal{L}'_{Heads}$ because we know that we have removed some possibilities for $\mathcal{X}'$ due to the loop originating from $\mathcal{X}$ and which might be a valid in some other part of the model.

Following is the new algorithm for *getPossibilities*.

Algorithm 4: getPossibilities($\mathcal{X}, \mathcal{R}, \mathcal{W}, \mathcal{C}$).

Input : A set $\mathcal{X}$ of predicates, a set $\mathcal{R}$ of shortcuts, current working set $\mathcal{W}$ of sets of predicates and a normalized set $\mathcal{C}$ of *MFOL+* clauses

Output: A set of updated shortcuts $\mathcal{R}$

$\mathcal{L}_0 \leftarrow \{\mathcal{X}\}, k \leftarrow 0, \mathcal{B} \leftarrow \emptyset, \mathcal{L}_{Heads} \leftarrow \emptyset$

repeat

$k \leftarrow k + 1$

$\mathcal{L}_k \leftarrow \mathcal{L}_{k-1}$

for all $P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow Q_1(x) \vee \cdots \vee Q_m(x) \in \mathcal{C}$ **do**

for all $\mathcal{Y} \in \mathcal{L}_k$ such that $\{P_1, \ldots, P_n\} \subseteq \mathcal{Y}$ and there is no $1 \leq i \leq m$ such that $Q_i \in \mathcal{Y}$ **do**

$\mathcal{L}_k \leftarrow ((\mathcal{L}_k \setminus \{\mathcal{Y}\}) \cup \{\mathcal{Y} \cup \{Q_l\} \mid l \in \{1, \ldots, m\}\}) \setminus \mathcal{B}$

if $\mathcal{Y} \notin \mathcal{L}_k$ **then**

$\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathcal{Y}\}$

for all $\mathcal{Y} \in \mathcal{L}_k$ and $f \in \mathcal{F}$ with $\mathcal{Y}_f = \{P \mid (P, f) \in \mathcal{D}(\mathcal{C}), P^f \in \mathcal{Y}\}$ such that $\mathcal{Y}_f \neq \emptyset$ **do**

$\mathcal{L}_k \leftarrow \mathcal{L}_k \setminus \{\mathcal{Y}\}$

if $\mathcal{Y}_f \notin \mathcal{W}$ **then**

if $(\mathcal{Y}_f, \mathcal{L}^f_{Heads}, \mathcal{R}_l) \notin \mathcal{R}$ **then**

$\mathcal{R} \leftarrow$ getPossibilities$(\mathcal{Y}_f, \mathcal{R}, \mathcal{W} \cup \{\mathcal{Y}_f\}, \mathcal{C})$

$\mathcal{L}_{Heads} \leftarrow \mathcal{L}_{Heads} \cup \mathcal{L}^f_{Heads}$ with $(\mathcal{Y}_f, \mathcal{L}^f_{Heads}, \mathcal{R}_l) \in \mathcal{R}$

$\mathcal{L}_k \leftarrow (\mathcal{L}_k \cup \{\mathcal{Y} \cup \{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{Z}\} \mid \mathcal{Z} \in \mathcal{R}_l, (\mathcal{Y}_f, \mathcal{L}'_{Heads}, \mathcal{R}_l) \in \mathcal{R}\}) \setminus \mathcal{B}$

$\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathcal{Y}\}$

else

$\mathcal{L}_{Heads} \leftarrow \mathcal{L}_{Heads} \cup \{\mathcal{Y}_f\}$

until $\mathcal{L}_k = \mathcal{L}_{k-1}$

$\mathcal{R} \leftarrow \mathcal{R} \setminus \{(\mathcal{X}', \mathcal{L}'_{Heads}, \mathcal{L}'_k) \mid (\mathcal{X}', \mathcal{L}'_{Heads}, \mathcal{L}'_k) \in \mathcal{R}, \mathcal{X} \in \mathcal{L}'_{Heads}\}$

return $\mathcal{R} \cup \{(\mathcal{X}, \mathcal{L}_{Heads}, \mathcal{L}_k)\}$

As this new algorithm *getPossibilities* returns the set $\mathcal{R}$ of shortcuts instead of set $\mathcal{L}$ of possibilities. We need to update the condition in the main algorithm as well. Instead of checking if $\mathcal{X}_a$ is part of the return set $\mathcal{L}_a$, we check if there is a shortcut $(\mathcal{X}_a, \mathcal{L}_a)$ calculated for $\mathcal{X}_a$ in the returned set $\mathcal{R}$ of shortcuts. Following is the updated main algorithm.

Algorithm 5:

Input : A normalized set $\mathcal{C}$ of Monadic clauses

Output: True if the set $\mathcal{C}$ has a finite model and false otherwise

(non-deterministically) guess a set $\mathcal{X}_a$ of predicates.

$\mathcal{R} \leftarrow$ getPossibilites$(a, \mathcal{X}_a, \{\mathcal{X}_a\}, \mathcal{C}_{\mathcal{X}_a})$

if $\mathcal{X}_a \in \mathcal{L}_a$ with $(\mathcal{X}_a, \mathcal{L}_a) \in \mathcal{R}$ **then**

  return true

else

  return false

4.2 Monadic Horn Clauses

In this section we consider the Horn case of our monadic clauses because there might be certain scenarios where we want to have the least model property in our logic like in database theory, or we do not need the expressiveness of disjunctions on right-hand of the clauses.

As we only allow one predicate on the right-hand side of the clauses, we do not have multiple possibilities for a set of predicates, we only have one possibility which we will call a set of consequences for this section. We can have a much simpler algorithm to decide the existence of a finite Herbrand model for a set of monadic Horn clauses. As there is just one set of consequences for any set of predicates, we can always reuse it whenever we need to find the consequences of the same set of predicates. The shortcuts are simple too, now they are just pairs of two sets $\mathcal{X}$ of predicates (reasons) and $\mathcal{L}$ of predicates (consequences). We do not need to keep track of loop heads for shortcuts, because as soon as we find a loop we know that there is no finite Herbrand model for the clause set, and we return by adding an empty shortcut. We keep on returning empty shortcuts until we get to the main procedure with an empty shortcut for $\mathcal{X}_a$ which means the main procedure returns false. For this Horn case we do not need to change the main algorithm apart from changing the condition $\mathcal{X}_a \in \mathcal{L}_a$ to $\mathcal{X}_a = \mathcal{L}_a$ as $\mathcal{L}_a$ is a set of consequences.

We provide the following simpler and efficient algorithm *getPossibilities* for monadic Horn clauses.

We do not formally prove that this new algorithm is only exponential in the size of clauses but it is easy to see that you can have only exponentially many recursive calls because of the working set $\mathcal{W}$ and the termination condition. And within a recursive call, the algorithm can make recursive calls only polynomially many times for each function symbol. The total number of recursive calls are still exponential because we save the calculated consequences in $\mathcal{R}$ and we need to calculate them only once.

Algorithm 6: getPossibilities($\mathcal{X}$, $\mathcal{R}$, $\mathcal{W}$, $\mathcal{C}$).

Input : A set $\mathcal{X}$ of predicates, a set $\mathcal{R}$ of shortcuts, current working set $\mathcal{W}$ of sets of predicates and a reduced set $\mathcal{C}$ of monadic Horn clauses

Output: A set of updated shortcuts $\mathcal{R}$

$\mathcal{L}_0 \leftarrow \{\mathcal{X}\}, k \leftarrow 0$

repeat

$k \leftarrow k + 1$

$\mathcal{L}_k \leftarrow \mathcal{L}_{k-1}$

for all $P_1(x) \wedge \cdots \wedge P_n(x) \rightarrow P(x) \in \mathcal{C}$ such that $\{P_1, \ldots, P_n\} \subseteq \mathcal{L}_k$ **do**

$\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{P\}$

for all $f \in \mathcal{F}$ with $\mathcal{Y}_f = \{P \mid (P, f) \in \mathcal{D}(\mathcal{C}), P^f \in \mathcal{L}_k\}$ such that $\mathcal{Y}_f \neq \emptyset$ **do**

if $\mathcal{Y}_f \in \mathcal{W}$ **then**

return $\mathcal{R} \cup \{(\mathcal{X}, \emptyset)\}$

if $(\mathcal{Y}_f, \mathcal{R}_l) \notin \mathcal{R}$ **then**

$\mathcal{R} \leftarrow$ getPossibilities$(\mathcal{Y}_f, \mathcal{R}, \mathcal{W} \cup \{\mathcal{Y}_f\}, \mathcal{C})$

if $\mathcal{R}_l \neq \emptyset$ with $(\mathcal{Y}_f, \mathcal{R}_l) \in \mathcal{R}$ **then**

$\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{P^f \mid (P, f) \in \mathcal{D}(\mathcal{C}), P \in \mathcal{R}_l\}$

else

return $\mathcal{R} \cup \{(\mathcal{X}, \emptyset)\}$

until $\mathcal{L}_k = \mathcal{L}_{k-1}$

return $\mathcal{R} \cup \{(\mathcal{X}, \mathcal{L}_k)\}$

Optionally we can also make the algorithm deterministic if we do not allow to have mixed clauses with both x and a in the normalized clauses. We can then assume that the normalized set $\mathcal{C}$ has only one positive clause of the form $\top \rightarrow A(a)$, where A is a special predicate we add in $\mathcal{P}$ which only occurs apart for this positive clause, alone on the left hand side of ground clauses. If there is no such predicate we add a new one A, add the clause $\top \rightarrow A(a)$ to $\mathcal{C}$ and replace $\top$ by $A(a)$ in every other positive clause. It is clear that this change does not affect the existence of a finite Herbrand model for $\mathcal{C}$. This remove the need

of non deterministic choice of $\mathcal{X}_a$ as well as the reduction of clauses.

We can modify the main algorithm as follows.

Algorithm 7:

Input : A normalized set $\mathcal{C}$ of Monadic Horn clauses

Output: True if the set $\mathcal{C}$ has a finite model and false otherwise

$\mathcal{L}_a \leftarrow$ getPossibilites$(a, \{A\}, \{\{A\}\}, \mathcal{C}_{\mathcal{X}_a})$

if $\mathcal{L}_a \neq \emptyset$ **then**

return true

else

return false

4.3 Multiple Constants

Restricting to only one constant curtails the expressiveness of monadic clauses. We can remove this restriction and extend the algorithms given in Chapter 4 to use multiple constants instead of the constant a. We define the set $\mathcal{F}^0$ as the set of finitely many constant symbols (nullary function symbols) and the terms are created over a finite set $\mathcal{F}$ of unary functions, $\mathcal{F}^0$ and the variable x.

For a set $\mathcal{C}$ of clauses with multiple constants from $\mathcal{F}^0$, in order to decide the existence of a finite Herbrand model. We guess the sets $\mathcal{X}_c$ of predicates for every constant $c \in \mathcal{F}^0$. And reduce the clause set $\mathcal{C}$ for all the guessed sets $\mathcal{X}_c$ for every $c \in \mathcal{F}^0$ iteratively to get the reduced set $\mathcal{C}'$ of clauses. As after the reduction we have eliminated the ground parts of the clauses, the clauses contain only x, so they cannot talk about terms having different constants. So the parts of the model involving a constant a are independent of those for another constant b where $a, b \in \mathcal{F}^0$ and $a \neq b$. We can decide the existence of a finite Herbrand model for $\mathcal{C}$ iff we find for every constant $c \in \mathcal{F}^0$ that there exists a finite Herbrand model for $\mathcal{C}'$ and the guessed set $\mathcal{X}_c$ using the algorithm proposed in Chapter 3. If we do not find a finite Herbrand model for any of the guessed sets $\mathcal{X}_c$, then we return false because we know that either the guess was wrong or there is not finite Herbrand model for $\mathcal{C}$.

As we look for loops for every constant separately, we do not need to update the procedure *get-Possibilities* defined in Chapter 3. Updating the main procedure as follows would be enough to incorporate multiple constants.

Algorithm 8:

Input : A normalized set $\mathcal{C}$ of Monadic clauses

Output: True if the set $\mathcal{C}$ has a finite model and false otherwise

(non-deterministically) guess a set $\mathcal{X}_c$ of predicates for every constant $c \in \mathcal{F}^0$.

calculate $\mathcal{C}'$ a reduced set of $\mathcal{C}$ for all the guessed sets $\mathcal{X}_c$

for all $c \in \mathcal{F}^0$ **do**

$\mathcal{L}_c \leftarrow$ getPossibilites$(c, \mathcal{X}_c, \{\mathcal{X}_c\}, \mathcal{C}')$

if $\mathcal{X}_c \notin \mathcal{L}_c$ **then**

return false

return true

Chapter 5

Conclusion

We proposed *monadic clauses* a novel decidable fragment of *first order logic* in Chapter 2 and gave a comparison with some of the other decidable fragments. We also provided a terminating algorithm in Chapter 3 for deciding the existence of finite Herbrand Model for *monadic clauses*. In Chapter 4 we provided some optimizations and special cases of the algorithm to better suit some problems. In contrast to the algorithm in [1], the presented algorithm does not blindly compute all possibilities, but works in a goal-oriented way by calling getPossibilities only for sets $\mathcal{X}$ that are needed to construct a finite Herbrand model.

5.1 Future Work

As future work, we propose exploring real world problems which can be solved using this fragment of logic efficiently. We can also study the use of *monadic clauses* for knowledge representation and also for knowledge extension of relational databases as a counterpart to *Datalog*. On the theoretical side, it would be interesting to know the precise complexity of our problem, i.e., whether it is ExpTime-complete as for anti-Horn clauses [1], or harder.

Bibliography

[1] Borgwardt, S., Morawska B. *Finding Finite Herbrand Models*, LTCS-Report 11-04, TU Dresden(2011), see http://lat.inf.tu-dresden.de/research/reports.html.

[2] Baader, F., Narendran, P.: *Unification of concept terms in description logics.* J. Symb. Comput. 31(3), 277-305 (2001)

[3] Heinrich Behmann, Beiträge zur Algebra der Logik, insbesondere zum Entscheidungsproblem, in Mathematische Annalen (1922)

[4] Löwenheim, L. (1915) "Über Möglichkeiten im Relativkalkül," Mathematische Annalen 76: 447-470. Translated as "On possibilities in the calculus of relatives" in Jean van Heijenoort, 1967. A Source Book in Mathematical Logic, 1879-1931. Harvard Univ. Press: 228-51.

[5] Ackermann, W., Hilbert, D. *Grundzüge der Theoretischen Logik.* Springer Berlin Heidelberg, Volume 27, 1949. Print-ISBN:978-3-642-52790-6; English translation Principles of Mathematical Logic, R. E. Luce, ed., (Chelsea Publishing Company, New York).

[6] Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer (1990)

[7] Gottlob, G., Orsi, G., Pieris, A., Simkus, M.: Datalog and its extensions for semantic web databases. In: Eiter, T., Krennwallner, T. (eds.) Reasoning Web 2012. LNCS, vol. 7487, pp. 54-77. Springer, Heidelberg (2012)

[8] Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/-: A family of logical knowledge representation and query languages for new applications. In: Proc. of LICS, pp. 228-242 (2010)

[9] Andréka, H., van Benthem, J., Németi, I.: Modal languages and bounded fragments of predicate logic. J. of Philosophical Logic 27(3), 217-274 (1998)

[10] Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. In: Proc. of ICALP, pp. 73-85 (1981)