Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

A Linear Algebraic Approach to Logic Programming

Author:
Yaniv Aspis

Supervisor:
Krysia Broda

Submitted in partial fulfillment of the requirements for the MSc degree in Computing (Machine Learning) of Imperial College London

September 2018

Abstract

Logic Programming is a declarative programming paradigm, where knowledge about the world is encoded into a computer system by symbolic rules. Traditionally, inference based on these rules has been done by symbolic manipulation. In this work we explore how inference from logic programs can be done by linear algebraic algorithms instead. We build upon previous work on matrix characterization of Horn propositional programs and show how it can be modified to realize a fully differentiable deductive process. We then consider normal programs and describe how the concept of program reducts can be realized linear algebraically. A new algorithm for non-monotonic deduction, based on linear algebraic reducts and differentiable deduction, is then presented. Finally, we show how to extend these results to First-Order Predicate Logic.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr Krysia Broda, for her excellent guidance and constant support throughout this project. I would also like to thank Professor Alessandra Russo for many sessions of helpful advice and commentary. I thank my family for their unwavering support and encouragement throughout my studies.

Contents

Chapter 1

Introduction

Logic Programming has long played a central role in Artificial Intelligence. Logic programs allow us to encode knowledge about the world in a way that is understandable to a computer system and yet is also readable by human beings. From such a knowledge base, an intelligent system can infer further rules and facts about the world, which can in turn support future decision making by itself or by end users.

Traditionally, inference from logic programming has been achieved by means of symbolic manipulation. Prolog, for instance, is a popular language for that purpose. Its interpreter answer queries by means of matching symbols (oftern through a process called unification). First, a goal term (symbol) is matched with the head of a clause in the program. Then, the body terms of said clause become new goals that must be proved (Bratko, 2001). Answer Set Programming is another paradigm of logic programming, which focuses instead on searching for (stable) models of a program. Answer Set solvers often combine symbolic deduction with a cominatorial search over the space of possible models (see, for instance, Gebser et al. (2012)).

Linear Algebra has also played a major role in Artificial Intelligence. Statistical machine learning is often based on linear algebraic operations, describing observations (samples) as elements of a vector space, and then learning functions over such vector spaces for inference purposes. In recent years, Deep Neural Network models have been highly successful in learning distributions over data by means of layering linear mappings with non-linear operations (Liu et al., 2017). Yet, logic programming is one area of AI where linear algebraic approaches have not received much attention.

There are several reasons to consider linear algebra as a mathematical framework for logic programming. First, the success of Deep Learning has accelerated the development of GPU-based hardware for AI. Such hardware is especially designed for efficient computation of operations such as matrix multiplication, which can be easily parallelized. Developing algorithms that take advantage of these operations could greatly improve the efficiency of logic programming.

Second, there are many linear algebraic operations that may be beneficial in the development of more efficient algorithms. Examples include dot products, outer products, projections, matrix inversion, matrix decompositions such as SVD and Schur, to name a few.

Finally, in recent years there has been increased interest in bridging the gap between symbolic AI and neural networks (Besold et al., 2017). Combining neural networks with logic programs has several appealing advantages. Logic programs allow for inference in the domain of small sample size - a domain where deep learning currently finds difficult to learn in. Logic programs also allow for easy transfer learning and inclusion of prior knowledge. A linear algebraic framework for logic programs is a natural first step towards neural-symbolic integration to overcome these problems.

While work on a linear algebraic approach for logic programming has so far been limited, two important contributions have been made recently. Sato (2017a) proposes to encode the truth value of predicates in high-order tensors, and shows that it leads to efficient computation of transitive closure relation. Sakama et al. (2017) provide a characterization of Horn, disjunctive and normal programs using matrix and third-order tensors. In our work, we build upon these ideas and provide three main contributions of our own.

First, we adjust the deductive algorithm by Sakama et al. (2017) to realize a completely differentiable process of deduction for Horn programs, which could prove useful in realizing the goal of neural-symbolic integration.

Second, we take a different approach from Sakama et al. (2017) towards deduction in normal programs. While their approach used an exponential-size third-order tensor to allow exact computation of all stable models of a program, we instead provide a linear algebraic characterization of program reducts for stable semantics (Gelfond and Lifschitz, 1988). Combining the differentiable deductive process with a matrix-based characterization of reducts allows us to develop an entirely new algorithm for non-monotonic reasoning, based on the Newton-Raphson method for root search (Parker and Chua, 1989).

Third, we build upon both the works of Sato (2017a) and Sakama et al. (2017) to show how a full linear algebraic characterization of First-Order Predicate logic can be achieved.

This report is structured as follows: Chapter 2 provides background information about logic programming, high-order tensors and work by other author on linear algebraic logic programming. Chapter 3 describes a restriction that we impose on all programs considered in this work. Proofs are provided that this restriction is without loss of generality. Chapter 4 deals with the monotonic case for logic programming, and expands the work of Sakama et al. (2017) to a differentiable form of deduction by means of root search.

Chapter 5 moves on the non-monotonic case, where a linear algebraic characterization of program reducts is given. The differentiable formulation of the deductive process is then shown to be applicable here as well. Chapter 6 studies several simple programs and illustrates how deduction as root search works in practice.

Chapter 7 introduces a new formulation of First-Order logic programs in terms of high-order tensors. Chapter 8 surveys related work. Chapter 9 discusses possible of extensions of this work to the case of web-scale knowledge bases. Finally, chapter 10 provides a summary of this work and a look at future research directions.

Chapter 2

Background

In this chapter, we provide background information needed to understand the rest of this work. This work deals with theoretical issues in logic programming and how they may be realized in a linear algebraic framework, so a through discussion of logic programming follows.

We begin with the monotonic case of Horn propositional logic programs, their syntax and semantics. We then describe how non-monotonic deduction is formulated in logic programming using negation-as-failure and normal programs. There is less general agreement to the semantics of normal programs, and we describe here several common approaches. We then proceed from the propositional case of logic programs to First-Order Predicate Logic, which is more often used when writing programs. Since some parts of our work involve high-order tensors, we provide some basic definitions, before turning our attention to how previous works have dealt with providing a linear algebraic characterization of logic programming.

2.1 Logic Programming

Logic Programming is a declarative form of programming, which aims to turn formal logic into computable programs. In logic programs, knowledge about a problem domain is formalized into a set of clauses of the form:

$$h \leftarrow b_1, b_2, ..., b_n \tag{2.1}$$

where h_1 is called the head of the clause and can be thought of as being a symbol defined by the clause. $b_1, b_2, ..., b_n$ are atoms (or literals) that consist of the body of the clause. The clause can then be read as "h is true if all of $b_1, b_2, ..., b_n$ are true". We use the symbol $\leftarrow$ for implication, and commas is a shorthand shorthand notation for 'logical and' (also written as $\wedge$). A special kind of clause is a fact, written as:

$$h \leftarrow \tag{2.2}$$

which states that h is unconditionally true. Finally, a program can also have constraints. These are often used to filter out 'undesirable' situations, and are expressed as clauses with $\bot$ as their head. However when we explicitly write a constraint we omit $\bot$. So, for instance, if we would like to rule out both p and q being true at the same time, we write:

$$\leftarrow p, q \tag{2.3}$$

A program can be thought of as a knowledge base. Typically a user is interested in querying against such a knowledge base. For instance, we may ask if a specific atom h can be inferred from the program, or if the program is even consistent to begin with, meaning non of the integrity constraints are broken. Traditionally, symbolic interpreters are used for such tasks.

2.1.1 Horn Propositional Programs

In propositional logic, each atom expresses a proposition, which can be either true or false. For instance, if the atom p stands for the statement "Socrates is a man" and the atom q stands for "Socrates is mortal", then a statement like "If Socrates is a man then Socrates is mortal" can be encoded as:

$$q \leftarrow p \tag{2.4}$$

Formally, consider an alphabet $\Sigma = \{\bot, \top, p_1, p_2, ..., p_N\}$. $\bot$ and $\top$ are two special symbols that are always in our alphabet. $\bot$ is a symbol representing 'False' while $\top$ represents 'True'. Other symbols are called *atoms* or *propositional variables*.

Definition 2.1.1. A *Horn clause* or *Horn rule* is a sentence over $\Sigma \cup \{\leftarrow\} \cup \{,\}$ of the form:

$$h \leftarrow b_1, b_2, ..., b_n \tag{2.5}$$

Where $h, b_1, ..., b_n \in \Sigma$

We often denote a rule such as above with the letter r. h is called the head of the clause and we denote $head(r) = h$. The set of atoms $\{b_1, b_2, ..., b_n\}$ is called the body of the rule and is denoted by $body(r)$.

Definition 2.1.2. A *fact* is a Horn clause r where $body(r) = \{\top\}$.

Definition 2.1.3. A *definite clause* is a Horn clause r where $head(r) \neq \bot$. If $head(r) = \bot$ the Horn clause is called a *constraint*.

Definition 2.1.4. A *Horn program* is a set of Horn clauses. If the program contains only definite clauses, it is instead called a *definite program*.

So far we have been describing the syntax of Horn programs. We now move on to describing the semantics of programs, that is, the truth-values of atoms in the program.

Definition 2.1.5. The *Herbrand Base* of a program P, denoted B_P, is the set of atoms that appear in the program, including $\bot$ and $\top$.

For simplicity, we often assume $\Sigma = B_P$. That is, our alphabet contains only the atoms that are needed for the program.

Definition 2.1.6. An *interpretation* I of a program P is a subset of B_P that contains $\top$.

So we always have $\top \subseteq I \subseteq B_P$. I can be thought of as an assignment of truth values to the atoms in the Herbrand base. When writing I explicitly we usually omit $\top$. Hence the "empty interpretation" is semantically identified with $\{\top\}$.

Definition 2.1.7. An interpretation I is *inconsistent* if $\bot \in I$. Otherwise I is *consistent*.

Since $\bot$ is semantically identified with 'False', inconsistent interpretations are not desirable. Instead, we look for interpretations that 'follow' the rules of the program.

Definition 2.1.8. An interpretation I is said to *satisfy* a Horn clause r if $body(r) \subseteq I$ implies $head(r) \in I$. If a consistent interpretation I satisfies all of the clauses of a program P, we say it is a *model* of P [1]. A *satisfiable* program is one that has a model. Otherwise, it is *unsatisfiable*.

We are interested in models of a program P. Van Emden and Kowalski (1976) showed that a definite program always has a model.

Definition 2.1.9. A model M of program P is called *minimal* if no proper subset of M is a model of P.

If a Horn program is satisfiable, then it has a single minimal model (Van Emden and Kowalski, 1976). All other models therefore contain this model.

Definition 2.1.10. The minimal model of a satisfiable Horn program P is called the *Least Herbrand Model* of P and is denoted $LHM(P)$.

Definition 2.1.11. Given a program P, the *Immediate Consequence* operator T_P is defined over consistent interpretations as follows:

$$\begin{gathered} T_P : 2^{B_P} \to 2^{B_P} \\ T_P(I) = \big\{head(r) | r \in P,\, body(r) \subseteq I\big\} \end{gathered} \tag{2.6}$$

[1] Note that a constraint is satisfied by I if its body is false in I.

In essence, $T_P(I)$ is everything that can be deduced from I in one step. T_P can be concatenated to achieve multiple step deduction:

$$\begin{aligned} T_P^0(I) &= I \\ T_P^{n+1}(I) &= T_P(T_P^n(I)) \end{aligned} \tag{2.7}$$

Note that we assume our programs always (implicitly) contain the clause $\top \leftarrow \top$, since without such a clause the output of T_P may not contain $\top$ and hence would not be an interpretation. Van Emden and Kowalski (1976) showed that for a large enough n, $T_P^n(\emptyset)$ is a fixed point of the Immediate Consequence operator, and that fixed point coincides with the Least Herbrand Model. Therefore, to compute all the consequences of a knowledge base, it is sufficient to compute the powers of T_P up to the fixed point.

2.2 Non-Monotonic Logic Programming

In realistic cases we often need a way of talking about a negation of an atom. In classical logic, the negation of an atom p is often denoted $\neg p$ and is understood to mean "p is false" (i.e. p can inferred to be false). From the point of view of logic programming, it is difficult to show a statement to not be true. Our knowledge of the world may simply not be sufficient enough to decide. Instead, in logic programming we introduce the notion of *negation-as-failure*. Under this viewpoint, if p cannot be shown to be true from our knowledge base, it is assumed to be false. This is sometimes referred to as the "closed world assumption". To differentiate from classical negation, we write a negation-as-failure literal as $\text{not}\, p$.

Definition 2.2.1. A *normal clause* or *normal rule* is a sentence over $\Sigma \cup \{\leftarrow\} \cup \{,\} \cup \{\text{not}\}$ of the form:

$$h \leftarrow b_1, b_2, ..., b_n, \text{not}\, c_1, \text{not}\, c_2, ..., \text{not}\, c_m \tag{2.8}$$

Where $h, b_1, b_2, ..., b_n, c_1, c_2, ..., c_m \in \Sigma$

The atoms $b_1, b_2, ..., b_n$ are referred to as the "positive body" and are denoted $body^+(r) = \{b_1, b_2, ..., b_n\}$. Similarly, the atoms $c_1, c_2, ..., c_m$ are known as the "negative body" and we denote $body^-(r) = \{c_1, c_2, ..., c_m\}$.

Definition 2.2.2. A *normal program* is a set of normal clauses.

Definition 2.2.3. An interpretation I is said to *satisfy* a normal clause r if $body^+(r) \subseteq I$ and $body^-(r) \cap I = \emptyset$ implies $head(r) \in I$.

A Horn clause is a special case of a normal clause where $m = 0$. Hence, Horn and definite programs are a special case of normal programs.

We are still interested in models of the program, namely, those interpretations that satisfy all the clauses of the program. However, with the introduction of negation, our knowledge base is no longer monotonic - introduction of new facts can invalidate previous acquired knowledge.

Example. Consider the following three programs:

$$p \leftarrow \text{not } q \tag{2.9}$$

$$\begin{aligned} &p \leftarrow \text{not } q \\ &q \leftarrow \end{aligned} \tag{2.10}$$

$$\begin{aligned} &p \leftarrow \text{not } q \\ &q \leftarrow \text{not } p \end{aligned} \tag{2.11}$$

For the first program, a query for p should return "True" since q cannot be deduced from the program. In the second program, however, q has been added as a fact, so a query for p should now return "False". p, that was once thought to be true, became false given new information.

The situation is made even worse in the third program. Given the third program, if we were to query for the truth value of p, we would not be able to return an answer. This is because deducing the truth value of p requires deducing the truth-value for q first, which requires deducing the truth value of p again. We therefore enter an infinite loop (in Prolog this is referred to as "Floundering"). The underlying cause of this problem is that our program no longer has a single minimal model. Instead there are two: $\{p\}$ and $\{q\}$. Neither of these models stands out as special. To resolve this problem, we would have to consider different kinds of semantics other than minimality.

2.2.1 Supported Models

The immediate consequence operator can be extended to the case of normal programs:

$$\begin{gathered} T_P : 2^{B_P} \to 2^{B_P} \\ T_P(I) = \big\{head(r) | r \in P,\ body^+(r) \subseteq I,\ body^-(r) \cap I = \emptyset\big\} \end{gathered} \tag{2.12}$$

Unlike the case of definite programs, computing the powers of $T_P(\emptyset)$ is not guaranteed to arrive at a fixed point. What are the fixed points then?

Definition 2.2.4. We say a model M of program P is *supported* if for every $p \in M$ there exists a rule $r \in P$ such that $head(r) = p$, $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$.

In supported models, every atom is 'explained' by some rule of the program.

Proposition 2.2.1. *Let P be a normal program. $T_P(M) = M$ if and only if M is a supported model.*

Proof. Suppose $M = T_P(M)$. Then for every rule $r \in P$, if $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$ then $head(r) \in M$. Hence M is a model. Suppose now that $p \in M$. Since $M = T_P(M)$ then there exists a rule $r \in P$ such that $head(r) = p$, $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$. Hence M is supported.

Now, suppose M is a supported model. Since M is a model, $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$ implies $head(r) \in M$. Hence $T_P(M) \subseteq M$. Suppose $p \in M$. Then there exists a rule $r \in P$ such that $head(r) = p$, $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$, therefore $p \in T_P(M)$. Thus $M = T_P(M)$. □

The concept of supported can induce models that are not intuitively desirable. For instance, for the following program:

$$\begin{aligned} p &\leftarrow q \\ q &\leftarrow p \end{aligned} \tag{2.13}$$

Supported models are $\emptyset$ and $\{p, q\}$. The latter model may seem problematic since it is not explained by any 'external' observation - $\{p, q\}$ is deduced since $\{p, q\}$ is assumed. To address this issue, the concept of well-supported models is introduced:

Definition 2.2.5. A model M of program P is said to be *well-supported* if there exists a well-founded ordering[2] $<$ of B_P such that: If $h \in M$, there exists a rule $r \in P$ such that $body^+(r) \subseteq M$, $body^-(r) \cap M = \emptyset$ and for every $b_i \in body^+(r)$ we have $b_i < h$.

Well-supported models, by definition, are always supported models. In the example program above, $\emptyset$ is a well-supported model, but $\{p, q\}$ is not, since any well-founded ordering over $\{p, q\}$ would not satisfy the condition that an atom in the head must be greater than an atom in the body.

2.2.2 Stable Semantics

Definition 2.2.6. Let P be a normal program and I be an interpretation. The reduct of P with respect to I is a Horn program P^I constructed as follows: For each $r \in P$, if $body^-(r) \cap I = \emptyset$, then add a clause r' to P^I such that $head(r') = head(r)$ and $body(r') = body^+(r)$.

Intuitively, P^I is constructed by deleting the rules of P such that I does not satisfy their negative body. Then, for the remaining rules, delete their negative body (that I does satisfy).

Definition 2.2.7. Let P be a normal program. M is a stable model of P if $M = LHM(P^M)$.

[2] A well-founded ordering $<$ over a set A is a total ordering such that every non-empty subset of A has a least element. (Ciesielski, 1997)

Remark 2.2.1. The minimal model of a Horn program is stable.

Stable models are indeed models of the program. In addition, stable models are always minimal. Stable semantics forms the basis of Answer Set Programming. In the Answer Set Programming paradigm, a problem domain is formulated as a normal logic program such that its stable models correspond to desired solutions. Finding a stable model of a normal program is in general NP-complete (Dantsin et al., 2001).

Proposition 2.2.2. *M is a stable model of a program P if and only if M is a well-supported model of a program P.*

The proposition is proven by Fages (1991). Hence stable models are also supported models, and are therefore fixed points of T_P.

Example Consider the following three program:

$$\begin{aligned} p &\leftarrow q \\ q &\leftarrow p \end{aligned} \tag{2.14}$$

$$\begin{aligned} p &\leftarrow q \\ q &\leftarrow p \\ r &\leftarrow \text{not}\, q \end{aligned} \tag{2.15}$$

$$p \leftarrow \text{not}\, p \tag{2.16}$$

The first program has two supported models but only one is stable: $\emptyset$. The second program has two supported models as well: $\{p, q\}$ and $\{r\}$, but only the latter is stable. The third program has no supported models at all, and hence no stable models either.

2.2.3 Stratified Programs

Definition 2.2.8. A normal program P is stratified if there exists a partitioning of the program into strata:

$$P = P_0 \cup P_1 \cup ... \cup P_k \tag{2.17}$$

Such that for every $p \in B_P$, every clause with p as its head is contained in a single strata, and for every clause $r \in P_i$ we have:

- If $b \in body^+(r)$ then every clause with b as its head is contained in $\bigcup_{j \leq i} P_j$.
- If $c \in body^-(r)$ then every clause with c as its head is contained in $\bigcup_{j < i} P_j$.

The idea behind stratifying a program is that each stratum only depends on previous ones, so they can be considered in sequence, and that computing a model for each part is efficient. For instance:

$$\begin{aligned} p &\leftarrow \text{not } q \\ q &\leftarrow \text{not } r \\ r &\leftarrow \end{aligned} \tag{2.18}$$

We divide the program into 3 strata $\{r \leftarrow\} \cup \{q \leftarrow \text{not } r\} \cup \{p \leftarrow \text{not } q\}$. Then we can compute a model $M_0 = \{r\}$ for $\{r \leftarrow\}$, then assuming the model M_0 we can compute a model $M_1 = M_0$ for $\{r \leftarrow\} \cup \{q \leftarrow \text{not } r\}$. Finally, assuming M_1 we can compute $M_2 = \{r, p\}$ for the entire program.

Proposition 2.2.3. *A stratified program P has a unique stable model.*

The proposition is proven by Gelfond and Lifschitz (1988). Hence in the case of stratified programs, we can avoid ambiguity regarding the model of the program. The model can also be efficiently computed as described above.

2.3 First-Order Logic

Propositional programs are not scalable in many scenarios. Logic Programs are usually written in 'First-Order Predicate logic'. This kind of formalism allows us to introduce variables to capture general relations. For instance, the statement "All men are mortal" can be written as:

$$\text{mortal}(X) \leftarrow \text{man}(X) \tag{2.19}$$

Where X is a variable that is 'universally quantified', meaning the relation holds for all possible values for X. So for $X = \text{socrates}$ we have:

$$\text{mortal(socrates)} \leftarrow \text{man(socrates)} \tag{2.20}$$

mortal and man are called predicate symbols, while socrates is called a constant. X is called a variable. In addition, we may have function symbols such as $succ$ in the following example:

$$\text{odd}(X) \leftarrow \text{even}(\text{succ}(X)) \tag{2.21}$$

odd and $even$ are predicate symbols since the atoms $odd(X)$ and $even(X)$ can be assigned values of true and false. $succ(X)$, being a function, produces another term instead and cannot be assigned such values. Following conventions, variable names begin with a capitalized letter, while predicates, function and constant symbols are all lower case.

Formally, we consider an alphabet Σ containing a set of symbols called predicate, a set of variable symbols, a set of constants and a set of function symbols.

Definition 2.3.1. A term over L is a constant, a variable or a function $f(t_1, t_2, ..., t_k)$ where $t_1, t_2, ..., t_k$ are terms. A term is *ground* if it does not contain variables.

Definition 2.3.2. An atom is a symbol of the form $p(t_1, t_2, ..., t_k)$ where p is a predicate symbol from Σ and $t_1, t_2, ..., t_k$ are terms. The number of terms in an atom is called the *arity* of the predicate.

A predicate symbol always has the same arity, in every atom it appears. Propositional symbols are in essence predicates with arity 0 (sometimes called 0-ary predicates). A predicate with arity 1 is called unary, and with arity 2 - binary. In general, k arity predicates are called k-ary.

Clauses in the First-Order case are of the same form as in the propositional case, except that atoms can now have any arity. In particular, variables may appear in any atom as terms. Variables in the head are are assumed to be universally quantified, while variables in the body are assumed to be existentially quantified. A First-Order clause reverts to the propositional case if all predicates are 0-ary. Another way to revert to propositional logic is by grounding:

Definition 2.3.3. The grounding of a clause r is a set of all clauses produced by replacing each variable with some ground term over Σ.

Even with a finite alphabet, grounding may produce an infinite set of clauses. For instance, the set of natural numbers can be represented in a logic program using the following language: A unary predicate $nat()$, representing the proposition that a term is a natural number, the constant symbol 0, and the function symbol $succ()$ used to produce the successor of a number. Over this alphabet, we can write the clause:

$$nat(X) \leftarrow \tag{2.22}$$

which, when grounded, produces an infinite set of clauses:

$$\begin{aligned} &nat(0) \leftarrow \\ &nat(succ(0)) \leftarrow \\ &nat(succ(succ(0)) \leftarrow \\ &\vdots \end{aligned} \tag{2.23}$$

If, besides limiting ourselves to a finite alphabet, we also restrict our programs to the case of no function symbols (so-called Function-Free Predicate Logic), then the grounding of a clause is assured to be finite.

Definition 2.3.4. The grounding of a program P, denoted $ground(P)$, is a program constructed by grounding all of the clauses of P.

$ground(P)$ is therefore a propositional program.

Definition 2.3.5. The *Herbrand domain* of a program P is the set of all terms that can be constructed from function symbols and constants. Elements of the Herbrand domain are often called *entities*.

Under this definition, the constant symbols of our programs receive a semantic meaning by directly associating them with entities in the Herbrand domain. As before, we can ensure the Herbrand domain is finite by not allowing function symbols.

Definition 2.3.6. The *Herbrand base* of a program is the set of all grounded atoms that can be constructed by predicate symbols and entities of the Herbrand domain, in addition to the symbols $\top$ and $\bot$.

As before, an interpretation is a subset of the Herbrand base that contains $\top$. All definitions from the propositional case naturally extend to First Order logic.

2.4 Tensors

Our definitions in this section follow Kolda and Bader (2009). In the context of Computer Science, tensors are multidimensional arrays. A tensor $A \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_K}$ of is of order K. Its elements are therefore indexed by K numbers: $A_{i_1 i_2 \ldots i_K}$. The various dimensions of the array are called its *modes*.

A 0th order tensor is a scalar, and has no index. We denote scalars by a lower case letter $a \in \mathbb{R}$. A 1st order tensor is a vector which is denoted by a bold lower case letter $\boldsymbol{v} \in \mathbb{R}^N$. We use the convention that $\boldsymbol{v}$ is a column vector, and its transpose $\boldsymbol{v}^\intercal$ is therefore a row vector. A 2nd order tensor is a matrix which is denoted by a bold upper case letter $\boldsymbol{D} \in \mathbb{R}^{N_1 \times N_2}$, and its transpose is denoted $\boldsymbol{D}^\intercal$. A tensor of order 3 or above is called a high order tensor.

Definition 2.4.1. Let $T \in \mathbb{R}^{N^1_1 \times N^1_2 \times \ldots \times N^1_{K^1}}$ and $S \in \mathbb{R}^{N^2_1 \times N^2_2 \times \ldots \times N^2_{K^2}}$ be tensors. The outer product of T and S, denoted $T \circ S$ is a tensor $Q \in \mathbb{R}^{N^1_1 \times N^1_2 \times \ldots \times N^1_{K^1} \times N^2_1 \times N^2_2 \times \ldots \times N^2_{K^2}}$ with elements:

$$Q_{i^1_1 i^1_2 \ldots i^1_{K^1} i^2_1 i^2_2 \ldots i^2_{K^2}} = T_{i^1_1 i^1_2 \ldots i^1_{K^1}} S_{i^2_1 i^2_2 \ldots i^2_{K^2}} \tag{2.24}$$

So the outer product of tensor X of order K_1 and tensor Y of order K_2 is a $K_1 + K_2$ order tensor. In particular, the outer product of two vectors $\boldsymbol{v}$ and $\boldsymbol{u}$ is a matrix, and is denoted:

$$\boldsymbol{v} \circ \boldsymbol{u} = \boldsymbol{v} \cdot \boldsymbol{u}^\intercal \tag{2.25}$$

A tensor can also be thought of as a linear algebraic entity constructed by performing an outer product operation on vectors. For instance, if:

$$T = \boldsymbol{v^1} \circ \boldsymbol{v^2} \circ ... \circ \boldsymbol{v^K} \tag{2.26}$$

then T is a K-th order tensor with elements:

$$T_{i_1 i_2 ... i_K} = v^1_{i_1} v^2_{i_2} ... v^K_{i_K} \tag{2.27}$$

Example. For the following three vectors:

$$\boldsymbol{v}^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \boldsymbol{v}^2 = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \quad \boldsymbol{v}^3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{2.28}$$

$T = \boldsymbol{v^1} \circ \boldsymbol{v^2} \circ \boldsymbol{v^3}$ is a $\mathbb{R}^{2\times3\times2}$ tensor. We have:

$$\begin{aligned}
T_{111} &= v^1_1 v^2_1 v^3_1 = 1 \cdot 0 \cdot (-1) = 0 & T_{112} &= v^1_1 v^2_1 v^3_2 = 1 \cdot 0 \cdot 1 = 0 \\
T_{121} &= v^1_1 v^2_2 v^3_1 = 1 \cdot 1 \cdot (-1) = -1 & T_{122} &= v^1_1 v^2_2 v^3_2 = 1 \cdot 1 \cdot 1 = 1 \\
T_{131} &= v^1_1 v^2_3 v^3_1 = 1 \cdot 3 \cdot (-1) = -3 & T_{132} &= v^1_1 v^2_3 v^3_2 = 1 \cdot 3 \cdot 1 = 3 \\
T_{211} &= v^1_2 v^2_1 v^3_1 = 2 \cdot 0 \cdot (-1) = 0 & T_{212} &= v^1_2 v^2_1 v^3_2 = 2 \cdot 0 \cdot 1 = 0 \\
T_{221} &= v^1_2 v^2_2 v^3_1 = 2 \cdot 1 \cdot (-1) = -2 & T_{222} &= v^1_2 v^2_2 v^3_2 = 2 \cdot 1 \cdot 1 = 2 \\
T_{231} &= v^1_2 v^2_3 v^3_1 = 2 \cdot 3 \cdot (-1) = -6 & T_{232} &= v^1_2 v^2_3 v^3_2 = 2 \cdot 3 \cdot 1 = 6
\end{aligned} \tag{2.29}$$

T as written above is a rank 1 tensor, since it can be written as a single outer product of K vectors. In general, a tensor will require summing over many such products. The most general form of a K-th order tensor is:

$$T = \sum_{i_1, i_2, \ldots, i_K} T_{i_1 i_2 ... i_K} \boldsymbol{e^1_{i_1}} \circ \boldsymbol{e^2_{i_2}} \circ ... \circ \boldsymbol{e^K_{i_k}} \tag{2.30}$$

Where $\{\boldsymbol{e^k_{i_k}}\}_{i_k=1}^{N_k}$ are the standard basis vectors of the k-th vector space.

Definition 2.4.2. Let $X, Y \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_K}$. The dot product of X and Y is defined as:

$$\langle X, Y \rangle = \sum_{i_1, i_2, \ldots, i_K} X_{i_1, i_2, \ldots, i_K} Y_{i_1, i_2, \ldots, i_K} \tag{2.31}$$

The dot product always produces a scalar. In the case of two vectors $\boldsymbol{v}, \boldsymbol{u} \in \mathbb{R}^N$, the dot product is equivalent to the regular dot product $\boldsymbol{v}^\intercal \boldsymbol{u}$. Note that in the case where

X and Y are matrices, the dot product as defined here is different from the matrix product, since the dot product produces a scalar rather than a matrix. For instance:

$$\left\langle \begin{bmatrix}1\\2\end{bmatrix} \circ \begin{bmatrix}2\\3\end{bmatrix}, \begin{bmatrix}6\\1\end{bmatrix} \circ \begin{bmatrix}4\\1\end{bmatrix} \right\rangle = \left\langle \begin{bmatrix}2 & 3\\4 & 6\end{bmatrix}, \begin{bmatrix}24 & 6\\4 & 1\end{bmatrix} \right\rangle = \\ = 2 \cdot 24 + 3 \cdot 6 + 4 \cdot 4 + 6 \cdot 1 = 88 \tag{2.32}$$

One useful identity of the dot product is:

$$\left\langle \boldsymbol{v^1} \circ \boldsymbol{v^2} \circ ... \circ \boldsymbol{v^K}, \boldsymbol{u^1} \circ \boldsymbol{u^2} \circ ... \circ \boldsymbol{u^K} \right\rangle = (\boldsymbol{v^1})^\intercal \boldsymbol{u^1} \cdot (\boldsymbol{v^2})^\intercal \boldsymbol{u^2} \cdot ... \cdot (\boldsymbol{v^K})^\intercal \boldsymbol{u^K} \tag{2.33}$$

For instance:

$$\left\langle \begin{bmatrix}1\\2\end{bmatrix} \circ \begin{bmatrix}2\\3\end{bmatrix}, \begin{bmatrix}6\\1\end{bmatrix} \circ \begin{bmatrix}4\\1\end{bmatrix} \right\rangle = \begin{bmatrix}1 & 2\end{bmatrix} \cdot \begin{bmatrix}6\\1\end{bmatrix} \cdot \begin{bmatrix}2 & 3\end{bmatrix} \cdot \begin{bmatrix}4\\1\end{bmatrix} = \\ = (1 \cdot 6 + 2 \cdot 1)\,(2 \cdot 4 + 3 \cdot 1) = 88 \tag{2.34}$$

2.5 Matrix Representation of Horn Programs

Sakama, Inoue and Sato (Sakama et al., 2017) offer a characterization of Horn propositional programs as matrices. Their work serves as a basis for the rest of this report, and so we bring here a brief summary of their results, before giving a different but equivalent formulation in the next chapter. Programs that can be mapped to matrices are those that satisfy a multiple definitions condition.

Definition 2.5.1. We say a Horn program P satisfies a *multiple definitions (MD) condition* if for every two clauses r_1 and r_2 if $head(r_1) = head(r_2)$ then either $|body(r_1)| = 1$ or $|body(r_2)| = 1$. [3]

Intuitively, the Multiple Definitions condition means every symbol can only be defined by one clause with more than one atom in its body. Any other clause which defines it must have a single body atom. Every Horn program can be mapped to a program that satisfies the MD condition. In the next chapter, we will give the exact details. For now, consider the example:

$$\begin{aligned} p &\leftarrow q, r \\ p &\leftarrow s, t \\ p &\leftarrow u \end{aligned} \tag{2.35}$$

p violates the MD condition. To fix this, we can add an auxiliary variable:

[3] This is a slightly looser condition than the one imposed in (Sakama et al., 2017), but it retains the correctness of the mapping.

$$\begin{aligned}
&p_1 \leftarrow q, r \\
&p \leftarrow s, t \\
&p \leftarrow u \\
&p \leftarrow p_1
\end{aligned} \tag{2.36}$$

And the violation has been removed. We can do so for all violating clauses. Next, Sakama et al describe how to map interpretations to vectors. Suppose our Herbrand base has N atoms: $\{p_1 = \bot, p_2 = \top, ..., p_N\}$. Then each atom can be mapped into a vector using the one-hot encoding. So for example, in the program:

$$\begin{aligned}
&p \leftarrow q, r \\
&q \leftarrow t \\
&r \leftarrow \\
&t \leftarrow s \\
&\leftarrow s
\end{aligned} \tag{2.37}$$

The Herbrand base has 7 atoms: $\{\bot, \top, p, q, r, t, s\}$. These could be mapped into the standard base of $\mathbb{R}^7$ using the one-hot encoding:

$$\boldsymbol{v}^{\bot} = \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{\top} = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{p} = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{q} = \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{r} = \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\end{bmatrix} \tag{2.38}$$

$$\boldsymbol{v}^{t} = \begin{bmatrix}0\\0\\0\\0\\0\\1\\0\end{bmatrix} \quad \boldsymbol{v}^{s} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

An interpretation like $\{p, q, s\}$ can then be encoded into a vector by setting the appropriate entries to 1:

$$\boldsymbol{v}^{\{p,q,s\}} = \begin{bmatrix}0\\1\\1\\1\\0\\0\\1\end{bmatrix} \tag{2.39}$$

Note that since interpretations always contain $\top$, the second entry of the vector is always set to 1.

A program with a Herbrand base of size N is embedded into a matrix in $\mathbb{R}^{N\times N}$. We denote the matrix representing a Horn program P as $\boldsymbol{D}^P$. Each clause in P is embedded into $\boldsymbol{D}^P$ using its elements. The row of the element corresponds to the head of the clause while the body elements correspond to columns. More formally, if $head(r) = p_i$ and $body(r) = \{p_{j_1}, p_{j_2}, ..., p_{j_n}\}$ then:

$$D^P_{ij_1} = D^P_{ij_2} = ... = D^P_{ij_n} = \frac{1}{n} \tag{2.40}$$

In addition, rules of the form $p_i \leftarrow \bot$ and $\top \leftarrow p_i$ are assumed to implicitly be part of the program, as they represent 'False implies everything' and 'Everything implies True', respectively. These are also encoded into the matrix. All other elements are set to 0. For the program above, we have:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.41}$$

The first column corresponds to the clauses: $\bot \leftarrow \bot$, $\top \leftarrow \bot$, $p \leftarrow \bot$, $q \leftarrow \bot$, $r \leftarrow \bot$, $t \leftarrow \bot$ and $s \leftarrow \bot$.

Similarly, the second row corresponds to: $\top \leftarrow \bot$, $\top \leftarrow \top$, $\top \leftarrow p$, $\top \leftarrow q$, $\top \leftarrow r$, $\top \leftarrow t$, $\top \leftarrow s$.

$D^P_{17} = 1$ corresponds to the constraint $\bot \leftarrow s$. $D^P_{46} = 1$ corresponds to the clause $q \leftarrow t$ and $D^P_{67} = 1$ corresponds to $t \leftarrow s$. The fact $r \leftarrow$ is encoded by $D^P_{52} = 1$. Finally the clause $p \leftarrow q, r$ is represented by $D^P_{34} = D^P_{35} = \frac{1}{2}$ together.
Define:

$$H_1(x) = \begin{cases} 1 & x \geq 1 \\ 0 & x < 1 \end{cases} \tag{2.42}$$

Proposition 2.5.1. *Let P be a Horn program satisfying the MD condition and $\boldsymbol{D}^P$ its matrix representation. Let I, J be interpretations. Then $J = T_P(I)$ if and only if $\boldsymbol{v}^J = H_1(\boldsymbol{D}^P \boldsymbol{v}^I)$ where H_1 is applied element-wise.*

We prove this proposition in the next chapter once we present our own formalism. For now we see that $LHM(P)$ can be computed linear algebraically by starting with $\boldsymbol{v_0} = \boldsymbol{v}^{\emptyset}$ and at each step computing $\boldsymbol{v_{k+1}} = H_1(\boldsymbol{D}^P \boldsymbol{v_k})$ until a fixed point is reached.

Chapter 3

Multiple Definitions

Before we turn our attention to a linear algebraic formulation of Logic Programming, we need to address the question of Multiple Definitions. As mentioned in section 2.5, the Matrix Representation suggested by Sakama et al. (2017) assumes that a Horn program follows a multiple definitions condition. The condition is required to ensure deduction is performed correctly. In this work, we consider not only Horn programs but also normal programs, so the condition must first be extended.

In this chapter, we provide some new notation that will be useful for the rest of the work. We then extend the MD condition to case of normal clauses and provide a transformation from an arbitrary program to one satisfying the condition. Finally, we show that considering only the class of programs satisfying the MD condition does not lose generality.

3.1 Notation

We denote the number of elements in the Herbrand base as N, and the number of clauses in our program as R. Clauses will be indexed by a small r. If we consider a single clause in our program:

$$h^r \leftarrow b_1^r, b_2^r, ..., b_{n^r}^r, \text{not } c_1^r, \text{not } c_2^r, ..., \text{not } c_{m^r}^r \tag{3.1}$$

we see it is comprised of three parts: the head h^r, the positive body which we denote $B^r = \{b_1^r, b_2^r, ..., b_{n^r}^r\}$ and the negative body $C^r = \{c_1^r, c_2^r, ..., c_{m^r}^r\}$. To make our formulation more simple, we will need to impose the restriction that both B^r and C^r are not empty. This can be done by replacing an empty positive body with $\{\top\}$ (which is a set trivially contained in every interpretation), and an empty negative body with $\{\bot\}$ (which is independent of any consistent interpretation). Note that in our notation $|B^r| = n^r \geq 1$ and $|C^r| = m^r \geq 1$.

For concision, clauses can be represented in triplet form $\langle h^r, B^r, C^r \rangle$. Facts are therefore of the form $\langle h^r, \{\top\}, \{\bot\} \rangle$ and constraints are of the form $\langle \bot, B^r, C^r \rangle$. Horn clauses are those that have $C^r = \{\bot\}$. A program can now be seen as a set of triplets. For example, the reduct of program P with respect to interpretation I is the program:

$$P^I = \left\{ \langle h^r, B^r, \{\bot\}\rangle \,\middle|\, \langle h^r, B^r, C^r\rangle \in P,\ I \cap C^r = \emptyset \right\} \tag{3.2}$$

3.2 Defining the Condition

Definition 3.2.1. Given a program P, the definitions of an atom $p \in B_P$ is the set of rules with p in their head. We denote:

$$\text{def}(p) = \{r \in P | head(r) = p\} \tag{3.3}$$

Definition 3.2.2. A rule $r \in P$ is *long* if $|body^{+}(r)| + |body^{-}(r)| > 2$.

Definition 3.2.3. The long definitions of p is the set of long rules with p in their head. We denote:

$$\text{longdef}(p) = \{r \in P | head(r) = p, |body^{+}(r)| + |body^{-}(r)| > 2\} \tag{3.4}$$

Definition 3.2.4. A normal program P satisfies the *multiple definitions condition* (MD condition) if for every $p \in B_P$ we have $|\text{longdef}(p)| \leq 1$. In other words, p is the head of at most one long rule.

For every normal program P, there exists an equivalent program P' that satisfies the multiple definitions condition. P' can be constructed as follows: For every $p \in B_P$ such that $|\text{longdef}(p)| > 1$, suppose $\{r_i = \langle p, B^i, C^i\rangle\}$ be a set of all the rules in longdef(p) except one. Replace each r_i in P with two rules: $\langle p_i, B^i, C^i\rangle$ and $\langle p, \{p_i\}, \{\bot\}\rangle$. The resulting program P' satisfies the MD condition.

3.3 Proof of Equivalence

Suppose we are given an arbitrary normal program P. We transform it to a program P' satisfying the MD condition by the procedure outlined above. The following set of propositions ensure that searching for models of P' is essentially equivalent to searching for models of P, and that once a model M' of P' is found, a very simple operation can transform it to a model M of P. Furthermore, the condition of being a supported or stable model is retained in this transformation.

Proposition 3.3.1. *Let M' be a model of P'. Then $M = M' \cap B_P$ is a model of P.*

Proof. Let $r \in P$. We must show M satisfies r. If $r \in P'$ we know that M' satisfies r meaning that if $body^{+}(r) \subseteq M'$ and $body^{-}(r) \cap M' = \emptyset$ then $head(r) \in M'$. In addition, since $r \in P$ then $head(r) \in B_P$, $body^{+}(r) \subseteq B_P$ and $body^{-}(r) \subseteq B_P$. So M

satisfies r as well.

Otherwise, suppose $r \notin P'$. Then from the construction of P', there exists two rules $r_1, r_2 \in P$ and an auxiliary symbol p such that $r_1 = \langle p, body^+(r), body^-(r)\rangle$ and $r_2 = \langle head(r), \{p\}, \{\perp\}\rangle$. M' satisfies both r_1 and r_2. If $body^+(r) \subseteq M'$ and $body^-(r) \cap M' = \emptyset$ then $p \in M'$ and since M' is consistent then $head(r) \in M'$. Since $r \in P$, as before, this implies M satisfies r. Otherwise, if $body^+(r) \not\subseteq M'$ or $body^-(r) \cap M' \neq \emptyset$ then M satisfies r again (trivially).

□

Proposition 3.3.2. *Let M be a model of P. Then there exists a model M' of P' such that $M = M' \cap B_P$.*

Proof. We construct M' as follows: All elements of M are in M' and if there exists a clause $r \in P'$ such that M' does not satisfy it, then add $head(r)$ to M'. M' is now trivially a model of P'. Now, for such a clause r that M did not satisfy, since M is a model of P, then r is not in P. Now M trivially satisfies auxiliary rules where the body is an auxiliary variable, so r cannot be a rule of that form. It must be the case then that r is an auxiliary rule with $head(r) \notin B_P$. Since this is true for all such rules r, we have $M = M' \cap B_P$. □

Finding models of P' is therefore equivalent to finding models of P.

Proposition 3.3.3. *Let M' be a supported model of P'. Then $M = M' \cap B_P$ is a supported model of P.*

Proof. We know M is a model. Let $p \in M$. We must show there exists a clause $r \in P$ such that $head(r) = p$, $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$.
Since $p \in M'$ and M' is supported there exists a clause $r \in P'$ such that $head(r) = p$, $body^+(r) \subseteq M'$ and $body^-(r) \cap M' = \emptyset$. If $r \in P$, then $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$, as required. If $r \notin P$ then from the construction of P', it is of the form $\langle p, \{q\}, \{\perp\}\rangle$ such that $q \notin B_P$, hence $q \in M'$. In addition there exists a rule $r_1 = \langle q, B, C\rangle$ such that $B \subseteq B_P$ and $\langle p, B, C\rangle$ is in P. Note that this is the only clause with q as its head. Since M' is supported, we have $B \subseteq M'$ and $C \cap M' = \emptyset$. So $B \subseteq M$ and $C \cap M = \emptyset$, as required. □

Proposition 3.3.4. *Let M be a supported model of P. Then there exists a supported model M' of P' such that $M = M' \cap B_P$.*

Proof. Construct M' as in the proof of 3.3.2. We know M' is a model of P' and $M = M' \cap B_P$. Let $p \in M'$. If $p \in M$ then there exists $r \in P$ such that $head(r) = p$ and $body^+(r) \subseteq M$ and $body^-(r) \cap M = \emptyset$. If $r \in P'$ then p is supported. Otherwise, there exist rules $\langle p, \{q\}, \{\perp\}\rangle$ and $\langle q, B, C\rangle$ in P' such that $q \notin B_P$ and $B \subseteq M$ and $C \cap M = \emptyset$. Hence $q \in M$, and p is supported. Finally, if $p \notin M$, then by the construction of M' there exists a rule r such that $head(r) = p$ and $body^+(r) \subseteq M'$ and $body^-(r) \cap M = \emptyset$. □

Proposition 3.3.5. *Let M' be a stable model of P'. Then $M = M' \cap B_P$ is a stable model of P.*

Proof. We know M is a supported model. To be well-supported, all that remains is to show a well-founded ordering over B_P as the definition requires. Since M' is a well-supported model, there exists a well-founded ordering $>$ over $B_{P'}$ such that for every $p \in M'$ there exists a clause $r \in P'$ with $head(r) = h$, $body^+(r) \subseteq M'$, $body^-(r) \cap M' = \emptyset$ and for every $b_i \in body^+(r)$ we have $p > b_i$. The same relation $>$ is also suitable for M, since if $p \in M$, then if $r \in P$ then the condition for $>$ is satisfied, and if not then from the construction of P', there exist two rules $\langle p, \{q\}, \{\perp\}\rangle$, $\langle q, body^+(r), body^-(r)\rangle$. Then for every $b_i \in body^+(r)$ we have $p > q > b_i$ as required. □

Proposition 3.3.6. *Let M be a stable model of P. Then there exists a stable model M' of P' such that $M = M' \cap B_P$.*

Proof. Construct M' as in proposition 3.3.2. We know M' is a supported model, so to show it is well-supported it is enough to show a well-founded ordering over $B_{P'}$ as required by the definition. Such an ordering exists for M over B_P. We need only to extend this ordering for any atom $q \in M \backslash B_P$. This is simple to do, since for such an atom there exist two rules $\langle p, \{q\}, \{\perp\}\rangle$ and $\langle q, B, C\rangle$ that were constructed for the program P' out of a rule $\langle p, B, C\rangle$ in the original program P. Now, since $q \in M'$, M' is a supported model and $\langle q, B, C\rangle$ is the only rule with q as its head, we need to make sure that for every $b \in B$ we have $q > b$. Now, for the original rule $\langle p, B, C\rangle$, it may be the case that we also need to ensure that $p > b$ (if this is the rule that satisfies the well-ordering condition for p). In such a case we must also ensure that $p > q$, but such an ordering can always be found since $B_{P'}$ is a countable set. □

Example Consider the program:

$$\begin{aligned} &p \leftarrow q, b \\ &q \leftarrow p, b \\ &p \leftarrow b, c \\ &b \leftarrow \\ &c \leftarrow \end{aligned} \tag{3.5}$$

The program has a well-supported model $\{b, c, p, q\}$, with one possible ordering being $q > p > b > c$. Since it violates the MD condition, we transform it to:

$$\begin{aligned} &p_1 \leftarrow q, b \\ &p \leftarrow p_1 \\ &q \leftarrow p, b \\ &p \leftarrow b, c \\ &b \leftarrow \\ &c \leftarrow \end{aligned} \tag{3.6}$$

The model as constructed in the proof would be $\{b, c, p, q, p_1\}$. To arrive at an ordering, we only need to ensure $p_1 > q$ and $p_1 > b$. Since p is already supported by the clause $p \leftarrow b, c$, p_1's location in the ordering does not affect it. It is simple to find such an ordering since our set of symbols is finite: $p_1 > q > p > b > c$. Now, consider this program:

$$\begin{aligned} &p \leftarrow b, r \\ &p \leftarrow q, r \\ &q \leftarrow p, r \\ &r \leftarrow \\ &b \leftarrow \end{aligned} \tag{3.7}$$

The well-supported model is $\{p, b, r, q\}$ for ordering $q > p > b > r$. After the transformation we have:

$$\begin{aligned} &p_1 \leftarrow b, r \\ &p \leftarrow p_1 \\ &p \leftarrow q, r \\ &q \leftarrow p, r \\ &r \leftarrow \\ &b \leftarrow \end{aligned} \tag{3.8}$$

The well-supported model is $\{p_1, b, r, p, q\}$. This time, we need $p_1 > b > r$ but since p's location in the ordering was dependent on the rule that was replaced, we also require that $p > p_1$. This is again easy to achieve since our Herbrand base is finite: $q > p > p_1 > b > r$.

We see then that looking for supported and stable models of the transformed program P' is equivalent to searching for models of our original program. We can therefore assume from this point on, without loss of generality, that we are only dealing with programs that satisfy the MD condition. If this is not the case, we can simply apply the transformation as a preprocessing step.

Chapter 4

Monotonic Deduction

We begin by considering the Monotonic case for deduction. The mapping by Sakama et al. (2017) described in section 2.5 provides the framework for this simple case. Under that framework, one-step deduction is realized by first applying a linear mapping (matrix multiplication) followed by a non-continuous operation.

In this chapter, we first rewrite the representation of Sakama et al. (2017) in vector format. Such notation would be helpful when we extend the approach to the non-monotonic case. We also provide a full proof of the correctness of the representation, using this format.

Next, we describe how the non-continuous operation can be replaced by a differentiable one. The differentiable operation is dependent on a choice of two parameters, which we call Threshold and Temperature. Exact inference can be maintained under this new operation for a suitable choice of the parameters. We show the conditions for exact inference and prove the correctness of the approach. We end the chapter by showing an example of how differentiable deduction can be used to find a supported model of a given Horn program.

4.1 Matrix Representation

Suppose we are given a program P satisfying the MD condition. We denoted the number of clauses in P as R, and the size of the Herbrand base as N. We can also define an ordering over B_P by specifying indices for each atom: $B_P = \{p_1 = \bot, p_2 = \top, p_3, ..., p_N\}$. We assign $\bot$ the index 1 and $\top$ the index 2.

We map each element of B_P to a vector in $\mathbb{R}^N$ using the one-hot encoding. Hence, if the standard base of $\mathbb{R}^N$ is $\{e_1, e_2, ..., e_N\}$ then we assign for $p_i \in B_P$ the vector $\boldsymbol{v}^{p_i} = e_i$. A set of atoms A (not necessarily containing $\top$) is mapped to:

$$\boldsymbol{v}^A = \sum_{p \in A} \boldsymbol{v}^p \tag{4.1}$$

So $\boldsymbol{v}^A \in \{0,1\}^N$ and we have:

- If $p_i \in A$, $v_i^A = (\boldsymbol{v}^{p_i})^\intercal \boldsymbol{v}^A = 1$.
- If $p_i \notin A$, $v_i^A = (\boldsymbol{v}^{p_i})^\intercal \boldsymbol{v}^A = 0$.

Note that since an interpretation I always contains $\top$, we would have $v_2^I = 1$ and if $v_1^I = 1$ then I is inconsistent.

We can now define the matrix representation of a Horn program. A rule r of P is of the form $\langle h^r, B^r, \{\bot\}\rangle$. We denote $|B^r| = n^r$. P is mapped to a matrix $\boldsymbol{D}^P \in \mathbb{R}^{N\times N}$ and is given by:

$$\boldsymbol{D}^P = \sum_{r\in P} \frac{1}{n^r} \boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal \tag{4.2}$$

The above can be seen to be equivalent to:

$$\boldsymbol{D}^P = \sum_{p\in B_P} \boldsymbol{v}^p \sum_{head(r)=p} \frac{1}{n^r} (\boldsymbol{v}^{B^r})^\intercal \tag{4.3}$$

Where the first summation is over the Herbrand base, and for each atom in the Herbrand base we sum over the rules of P that define it. For example, if P is the following program:

$$\begin{aligned} &p \leftarrow q \\ &p \leftarrow r \qquad (4.4) \\ &r \leftarrow \qquad (4.5) \end{aligned}$$

$\boldsymbol{D}^P$ is written as in equation 4.2 as:

$$\boldsymbol{D}^P = \boldsymbol{v}^\top (\boldsymbol{v}^\top)^\intercal + \boldsymbol{v}^p (\boldsymbol{v}^q)^\intercal + \boldsymbol{v}^p (\boldsymbol{v}^r)^\intercal + \boldsymbol{v}^r (\boldsymbol{v}^\top)^\intercal \tag{4.6}$$

and as in equation 4.3 as:

$$\boldsymbol{D}^P = \boldsymbol{v}^\bot \cdot \boldsymbol{0}^\intercal + \boldsymbol{v}^\top (\boldsymbol{v}^\top)^\intercal + \boldsymbol{v}^p \left[(\boldsymbol{v}^q)^\intercal + (\boldsymbol{v}^r)^\intercal\right] + \boldsymbol{v}^q \cdot \boldsymbol{0}^\intercal + \boldsymbol{v}^r (\boldsymbol{v}^\top)^\intercal \tag{4.7}$$

Another way to write $\boldsymbol{D}^P$ is using matrix multiplication instead of summation. Consider all the head vectors $\{\boldsymbol{v}^{h^r}\}_{r=1}^R$. We can stack these vectors in order as columns of a matrix. Denote this matrix $\boldsymbol{H}$. So we have $H_{ir} = v_i^{h^r}$. Similarly, we can consider the sequence of body vectors after they have been divided by the size of the body: $\{\frac{1}{n^r}\boldsymbol{v}^{B^r}\}_{r=1}^R$. These too can be stacked into a matrix $\boldsymbol{B}$, making sure to retain the same order of rules as in $\boldsymbol{H}$. So we have $B_{ir} = \frac{1}{n^r} v_i^{B^r}$. Note that $\boldsymbol{H}, \boldsymbol{B} \in \mathbb{R}^{N\times R}$. It can be shown that:

$$\boldsymbol{D}^P = \boldsymbol{H}\boldsymbol{B}^\intercal \tag{4.8}$$

Proof.

$$
\begin{aligned}
(HB^\intercal)_{ij} &= \sum_{r=1}^{R} H_{ir}B_{jr} = \sum_{r=1}^{R} \frac{1}{n^r} v_i^{h^r} v_j^{B^r} = \sum_{r=1}^{R} \frac{1}{n^r} \left(\boldsymbol{v}^{h^r}(\boldsymbol{v}^{B^r})^\intercal\right)_{ij} = \\
&= \left(\sum_{r=1}^{R} \frac{1}{n^r} \boldsymbol{v}^{h^r}(\boldsymbol{v}^{B^r})^\intercal\right)_{ij} = D^P_{ij}
\end{aligned}
\tag{4.9}
$$

□

The next proposition shows the correctness of the matrix representation. Namely, multiplying the matrix $\boldsymbol{D}^P$ with an interpretation vector and applying the operation H_1 realises application of T_P.

Proposition 4.1.1. *Let I and J bet two interpretations. $J = T_P(I)$ if and only if $\boldsymbol{v}^J = H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I)$, where H_1 is applied element-wise.*

Proof. Denote $\boldsymbol{u} = H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I)$. Note that:

$$
\begin{aligned}
H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I) &= H_1\Bigg(\sum_{p \in B_P} \boldsymbol{v}^p \sum_{head(r)=p} \frac{1}{n^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I\Bigg) = \\
&= \sum_{p \in B_P} \boldsymbol{v}^p \cdot H_1\Bigg(\sum_{head(r)=p} \frac{(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r}\Bigg)
\end{aligned}
\tag{4.10}
$$

Hence:

$$
(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = H_1\Bigg(\sum_{head(r)=p} \frac{(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r}\Bigg) \tag{4.11}
$$

Now, suppose $J = T_P(I)$. If $p \in J$, then $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 1$ and there exists a rule $\langle p, B^r, \{\bot\}\rangle \in P$ such that $B^r \subseteq I$. Therefore $(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I = n^r$ which implies $H_1\Bigg(\sum_{head(r)=p} \frac{(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r}\Bigg) = 1$ and therefore $(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = 1$.
If $p \notin J$ then for every rule $\langle p, B^r, \{\bot\}\rangle \in P$, $B^r \nsubseteq I$. Since P satisfies the MD condition, for every rule except at most one we have $|B^r| = 1$ and therefore $(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I = 0$. If there exists a rule $\langle p, B^r, \{\bot\}\rangle$ such that $|B^r| > 1$ then, since $B^r \nsubseteq I$, we have $(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I < n^r$. Therefore $H_1\Bigg(\sum_{head(r)=p} \frac{(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r}\Bigg) = 0$ and $(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = 0$. All of these together imply $\boldsymbol{u} = \boldsymbol{v}^{\boldsymbol{J}}$.

Conversely, suppose that $\boldsymbol{v}^J = H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I)$. Since we know that $\boldsymbol{v}^{T_P(I)} = H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I)$, then we necessarily have $\boldsymbol{v}^J = \boldsymbol{v}^{T_P(I)}$. From the definition of the vector representation of a set of atoms, this implies $J = T_P(I)$.

□

With the matrix representation we have a linear algebraic method to compute the Least Herbrand Model of any given Horn program. Moreover, a matrix representation could theoretically allow the combination of prior knowledge in the form of a logic program into a loss function. Notice, however, that the H_1 operation is not differentiable (or even continuous), so the resulting loss function would not be differentiable either. To address this, we next consider replacing H_1 with a differentiable function.

4.2 Differentiable Deduction

H_1 has several differentiable approximations (Bracewell, 2000). Here we consider replacing H_1 with a sigmoid function. Other options may also be possible, but as our intention here is to demonstrate that a differentiable form of matrix-based deduction is possible, we leave consideration of alternative approximations to future work.

Given constants $0 < \gamma < 1$ and $\tau > 0$, we define:

$$\sigma_{\gamma,\tau}(x) = \frac{1}{1 + e^{\frac{\gamma - x}{\tau}}} \tag{4.12}$$

γ is a 'confidence threshold' which represents a boundary between values representing 'True' ($x > \gamma$) and 'False' ($x < \gamma$). τ is called 'temperature'. The values of γ and τ must be carefully picked, or the approximation will not produce correct results. For instance, suppose P has as its longest rule:

$$h^r \leftarrow b_1^r \wedge b_2^r \wedge ... \wedge b_{n^r}^r \tag{4.13}$$

We must set $\gamma > \frac{n^r - 1}{n^r}$ or partially satisfied bodies may cause the head to incorrectly be classified as 'True'.

Clearly we must pick our values carefully. Suppose an atom $p \in B_P$ is the head of R_p rules in P. We assume the first of these rules, $\langle p, B_p^1, \{\perp\} \rangle$, is of length $|B_p^1| = n_p^1$. Since P satisfies the MD condition, the rest are of length one, and are of the form $\langle p, \{b_p^r\}, \{\perp\} \rangle$ for $r = 2 ... R_p$. The rules that may affect the truth value of p are therefore represented by the matrix:

$$\boldsymbol{D}^p = \frac{1}{n_p^1} \boldsymbol{v}^p (\boldsymbol{v}^{B_p^1})^\intercal + \sum_{r=2}^{R_p} \boldsymbol{v}^p (\boldsymbol{v}^{b_p^r})^\intercal \tag{4.14}$$

For differentiable deduction to work, we require that:

- If $B_p^1 \subseteq I$ or $b_p^r \in I$ then $(\boldsymbol{v}^p)^\intercal \cdot \sigma_{\gamma,\tau}(\boldsymbol{D}^p \cdot \boldsymbol{v}^I) > \gamma$.
- If $B_p^1 \not\subseteq I$ and $b_p^r \notin I$ for all b_p^r then $(\boldsymbol{v}^p)^\intercal \cdot \sigma_{\gamma,\tau}(\boldsymbol{D}^p \cdot \boldsymbol{v}^I) < \gamma$.

Now:

$$\sigma_{\gamma,\tau}(\boldsymbol{D}^p \cdot \boldsymbol{v}^I) = \sigma_{\gamma,\tau}\Bigg(\big(\frac{(\boldsymbol{v}^{B^1_p})^\intercal \boldsymbol{v}^I}{n^1_p} + \sum_{r=2}^{R_p} (\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I\big)\boldsymbol{v}^p \Bigg) =$$

$$= \sigma_{\gamma,\tau}\Bigg(\frac{(\boldsymbol{v}^{B^1_p})^\intercal \boldsymbol{v}^I}{n^1_p} + \sum_{r=2}^{R_p} (\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I \Bigg)\boldsymbol{v}^p \tag{4.15}$$

So our conditions become $B^r \subseteq I$ or $b^r_p \in I$ if and only if:

$$\sigma_{\gamma,\tau}\Bigg(\frac{(\boldsymbol{v}^{B^1_p})^\intercal \boldsymbol{v}^I}{n^1_p} + \sum_{i=2}^{R_p} (\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I \Bigg) > \gamma \tag{4.16}$$

This condition is not trivial to satisfy. For instance, suppose all of $\boldsymbol{v^I}$'s entries corresponding to B^r_p are set to values greater than γ, and all the entries corresponding to b^r_p are set to 0. This only guarantees $\sigma_{\gamma,t}\big(\frac{(\boldsymbol{v}^{B^i_p})^\intercal \boldsymbol{v}^I}{n^i_p}\big) > \frac{1}{2}$. To remedy this, we introduce two new constants $0 < \gamma^\perp < \gamma < \gamma^\top < 1$. $\gamma^\perp$ represents an upper-bound for 'False' and $\gamma^\top$ a lower-bound for 'True'. Our requirements now become:

1. If for every $b \in B^r$, $(\boldsymbol{v}^b)^\intercal \boldsymbol{v}^I > \gamma^\top$ or $(\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I > \gamma^\top$ for some b^r_p, then $\sigma_{\gamma,\tau}\Bigg(\frac{(\boldsymbol{v}^{B^1_p})^\intercal \boldsymbol{v}^I}{n^1_p} + \sum_{i=2}^{R_p} (\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I \Bigg) > \gamma^\top$.
2. If there exists $p \in B^r$ such that $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^I < \gamma^\perp$ and for every b^r_p we have $(\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I < \gamma^\perp$, then $\sigma_{\gamma,\tau}\Bigg(\frac{(\boldsymbol{v}^{B^1_p})^\intercal \boldsymbol{v}^I}{n^1_p} + \sum_{r=2}^{R_p} (\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I \Bigg) < \gamma^\perp$.

First, we denote:

$$x = \frac{(\boldsymbol{v}^{B^1_p})^\intercal \boldsymbol{v}^I}{n^1_p} + \sum_{r=2}^{R_p} (\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I \tag{4.17}$$

Looking at condition 1, if for every $b \in B^r$ we have $(\boldsymbol{v}^b)^\intercal \boldsymbol{v}^I > \gamma^\top$ then $(\boldsymbol{v}^{B^1_p})^\intercal \boldsymbol{v}^I \geq n^1_p \cdot \gamma^\top$. Since $(\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I > 0$ we are guaranteed that $x > \gamma^\top$. Similarly, if $(\boldsymbol{v}^{b^r_p})^\intercal \boldsymbol{v}^I > \gamma^\top$ we are also guaranteed $x > \gamma^\top$. Hence condition one is equivalent to demanding:

$$x > \gamma^\top \quad \Rightarrow \quad \sigma_{\gamma,\tau}(x) > \gamma^\top \tag{4.18}$$

This gives an upper bound for τ as follows:

$$\sigma_{\gamma,\tau}(x) = \frac{1}{1 + e^{\frac{\gamma - x}{\tau}}} \underset{x > \gamma^\top}{>} \frac{1}{1 + e^{\frac{\gamma - \gamma^\top}{\tau}}} \tag{4.19}$$

$$\frac{1}{1+e^{\frac{\gamma-\gamma^{\top}}{\tau}}} > \gamma^{\top} \quad \Leftrightarrow \quad \tau < \frac{\gamma-\gamma^{\top}}{\ln(\frac{1}{\gamma^{\top}}-1)} \tag{4.20}$$

Note that for $\tau > 0$ to be true, we require in equation 4.20 that $\gamma^{\top} > \frac{1}{2}$.

The second condition is more complicated. In the worst case scenario, in our long rule all elements in the body except one are true. The upper bound for 'True' entries in $\boldsymbol{v}^I$ is '1' and the upper bound for 'False' entries is $\gamma^{\perp}$, so we have as an upper bound:

$$(\boldsymbol{v}^{B_p^1})^{\intercal}\boldsymbol{v}^I < \frac{n_p^1-1}{n_p^1} + \frac{\gamma^{\perp}}{n_p^1} \tag{4.21}$$

In addition, all short rules must have their body set to false, which means $(\boldsymbol{v}^{b_p^r})^{\intercal}\boldsymbol{v}^I < \gamma^{\perp}$ for all $R_p - 1$ short rules. Putting it all together we get an upper bound for x:

$$x < \frac{n_p^1-1}{n_p^1} + (\frac{1}{n_p^1} + R_p - 1)\gamma^{\perp} \triangleq x^{\perp} \tag{4.22}$$

So condition two becomes:

$$x < x^{\perp} \quad \Rightarrow \quad \sigma_{\gamma,\tau}(x) < \gamma^{\perp} \tag{4.23}$$

Similar to condition one, we get:

$$\sigma_{\gamma,\tau}(x) = \frac{1}{1+e^{\frac{\gamma-x}{\tau}}} \underset{x<x^{\perp}}{<} \frac{1}{1+e^{\frac{\gamma-x^{\perp}}{\tau}}} \tag{4.24}$$

$$\begin{aligned} \frac{1}{1+e^{\frac{\gamma-x^{\perp}}{\tau}}} <\gamma^{\perp} \quad &\Leftrightarrow \quad \tau < \frac{\gamma-x^{\perp}}{\ln(\frac{1}{\gamma^{\perp}}-1)} \\ &\Leftrightarrow \quad \tau < \frac{\gamma-\frac{n_p^1-1}{n_p^1}-(\frac{1}{n_p^1}+R_p-1)\gamma^{\perp}}{\ln(\frac{1}{\gamma^{\perp}}-1)} \end{aligned} \tag{4.25}$$

For equation 4.25, we now require that $\gamma^{\perp} < \frac{1}{2}$ and:

$$\begin{aligned} \gamma-\frac{n_p^1-1}{n_p^1} - (\frac{1}{n_p^1}+R_p-1)\gamma^{\perp} > 0 \\ \Leftrightarrow \quad \gamma^{\perp} < \frac{\gamma-\frac{n_p^1-1}{n_p^1}}{\frac{1}{n_p^1}+R_p-1} \end{aligned} \tag{4.26}$$

Which, as noted in the beginning, requires $\gamma > \frac{n_p^1-1}{n_p^1}$.

Putting all of these conditions together, we must set $\gamma, \gamma^{\perp}, \gamma^{\top}$ and τ such that:

1. $\gamma > \max_p \left\{\frac{n_p^1 - 1}{n_p^1}\right\}$.
2. $\gamma^\top > \max\left\{\frac{1}{2}, \gamma\right\}$.
3. $\gamma^\perp < \min_p \left\{\frac{1}{2}, \frac{\gamma - \frac{n_p^1-1}{n_p^1}}{\frac{1}{n_p^1} + R_p - 1}\right\}$.
4. $\tau < \min_p \left\{\frac{\gamma - \gamma^\tau}{\ln(\frac{1}{\gamma^\tau} - 1)}, \frac{\gamma - \frac{n_p^1-1}{n_p^1} - (\frac{1}{n_p^1} + R_p - 1)\gamma^\top}{\ln(\frac{1}{\gamma^\perp} - 1)}\right\}$

To show one possible solution that satisfies the above conditions, denote:

$$n = \max_p\{n_p^1\} \qquad r = \max_p\{R_p\} \tag{4.27}$$

We need to set $\gamma > \frac{n-1}{n}$. Two possible solutions, for example, are $\gamma = \frac{n - \frac{1}{2}}{n}$ and $\gamma = \frac{n}{n+1}$. In either case we have $\gamma \geq \frac{1}{2}$.

We can then set:

$$\gamma^\top = \frac{\gamma + 1}{2} \tag{4.28}$$

And:

$$\gamma^\perp = \frac{1}{2} \cdot \frac{\gamma - \frac{n-1}{n}}{\frac{1}{n} + r - 1} \tag{4.29}$$

The temperature can then be set as:

$$\tau = \min\left\{\frac{\gamma - \gamma^\top}{\ln(\frac{1}{\gamma^\top} - 1)}, \frac{\gamma - \frac{n-1}{n} - (\frac{1}{n} + r - 1)\gamma^\perp}{\ln(\frac{1}{\gamma^\perp} - 1)}\right\} \tag{4.30}$$

Definition 4.2.1. Suppose $\boldsymbol{v}, \boldsymbol{u} \in [0,1]^N$ and let $0 < \gamma^\perp < \gamma^\top < 1$. We say $\boldsymbol{v}$ and $\boldsymbol{u}$ are *semantically equivalent* if for all $1 \leq i \leq N$:

1. $v_i > \gamma^\top$ if and only if $u_i > \gamma^\top$.
2. $v_i < \gamma^\perp$ if and only if $u_i < \gamma^\perp$.

In such a case we write $\boldsymbol{v} \sim_{\gamma^\perp}^{\gamma^\top} \boldsymbol{u}$.

Intuitively, semantically equivalent vectors represent the same interpretation. The next proposition ensures our differentiable method correctly performs deduction.

Proposition 4.2.1. *Let P be a Horn program and $\boldsymbol{D}^P$ its matrix representation. Let I be an interpretation. Suppose $\boldsymbol{u} \sim^{\gamma^T}_{\gamma^F} \boldsymbol{v}^I$. Then $H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I) \sim^{\gamma^T}_{\gamma^F} \sigma_{\gamma,t}(\boldsymbol{D}^P \cdot \boldsymbol{u})$.*

Proof. Denote $\boldsymbol{v}^J = H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I)$ and let $p \in B_P$. Suppose $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 1$. Then there exists a rule $r \in P$ such that $head(r) = p$ and for all $b \in body^+(r)$ we have $(\boldsymbol{v}^b)^\intercal \boldsymbol{v}^I = 1$. Hence for all $b \in body^+(r)$ we have $(\boldsymbol{v}^b)^\intercal \boldsymbol{u} > \gamma^\top$ and as we've seen, for a suitable choice of γ and t we are assured $\sigma_{\gamma,t}(\boldsymbol{D}^P \cdot \boldsymbol{u}) > \gamma^T$ as required.
Suppose now that $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 0$. Then for every rule such that $head(r) = p$, there exists at least one atom $b \in body^+(r)$ such that $(\boldsymbol{v}^b)^\intercal \boldsymbol{v}^I = 0$ and hence $(\boldsymbol{v}^b)^\intercal \boldsymbol{u} < \gamma^F$. We are therefore assured $\sigma_{\gamma,t}(\boldsymbol{D}^P \cdot \boldsymbol{u}) < \gamma^F$. □

Define the sequence:

1. $\boldsymbol{v}^{M_0} = \boldsymbol{v}^{\{\top\}}$.
2. $\boldsymbol{v}^{M_k} = \sigma_{\gamma,t}(\boldsymbol{D}^P \cdot \boldsymbol{v}^{M_{k-1}})$.

If P is satisfiable, we know that $\boldsymbol{v}^{M_k} \sim^{\gamma^T}_{\gamma^F} \boldsymbol{v}^{LHM(P)}$ for a large enough k.

4.3 Deduction as Root Search

Proposition 4.3.1. *Let P be a Horn program. Then M is a supported model if and only if $\boldsymbol{v}^M = H_1(D^P \boldsymbol{v}^M)$.*

Proof. Follows immediately from propositions 2.2.1 and 4.1.1 □

Given a program P, define the mapping:

$$F(\boldsymbol{v}) = \sigma_{\gamma,\tau}(\boldsymbol{D}^P \cdot \boldsymbol{v}) - \boldsymbol{v} \tag{4.31}$$

Supported models of P correspond to roots of F. We can therefore search for such models by finding the roots of F. Here we consider using the Newton-Raphson algorithm for this task (Parker and Chua, 1989). Suppose $\boldsymbol{J_v}$ is the Jacobian of F at point $\boldsymbol{v}$. The Newton-Raphson algorithm starts with some initial point $\boldsymbol{v}^0$, and at the k-th step of the algorithm performs:

$$\boldsymbol{v}^{k+1} = \boldsymbol{v}^k - \boldsymbol{J}^{-1}_{\boldsymbol{v}^k} \cdot F(\boldsymbol{v^k}) \tag{4.32}$$

To avoid inverting a matrix, we can instead solve the following equation:

$$\boldsymbol{J}_{\boldsymbol{v}^k}(\boldsymbol{v}^{k+1} - \boldsymbol{v}^k) = -F(\boldsymbol{v^k}) = \boldsymbol{v}^k - \sigma_{\gamma,\tau}(\boldsymbol{D}^P \cdot \boldsymbol{v}^k) \tag{4.33}$$

To find the difference $\boldsymbol{v}^{k+1} - \boldsymbol{v}^k$.

Here we show the derivation of the Jacobian. For simplicity of notation, we omit the superscript from $\boldsymbol{v}^{\boldsymbol{k}}$. We have:

$$J_{ij} = \frac{\partial F(\boldsymbol{v})_i}{\partial v_j} = \frac{\partial \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i}{\partial v_j} - \frac{\partial v_i}{\partial v_j} = \frac{\partial \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i}{\partial v_j} - \delta_{ij} \tag{4.34}$$

Where:

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{4.35}$$

Now:

$$\sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i = \sigma_{\gamma,\tau}\big(\sum_{k=1}^{N} D^P_{ik} v_k\big) \tag{4.36}$$

And:

$$\frac{\mathrm{d}\sigma_{\gamma,\tau}(x)}{\mathrm{d}x} = \frac{1}{\tau}\sigma_{\gamma,\tau}(x)(1 - \sigma_{\gamma,\tau}(x)) \tag{4.37}$$

Hence:

$$\begin{aligned} \frac{\partial \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i}{\partial v_j} &= \frac{1}{\tau}\sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i(1 - \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i) \cdot \frac{\partial \sum\limits_{k=1}^{N} D^P_{ik} v_k}{\partial v_j} = \\ &= \frac{1}{\tau}\sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i(1 - \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i) \cdot \sum_{k=1}^{N} D^P_{ik}\frac{\partial v_k}{\partial v_j} = \\ &= \frac{1}{\tau}\sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i(1 - \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i) \cdot \sum_{k=1}^{N} D^P_{ik}\delta_{kj} = \\ &= \frac{1}{\tau}\sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i(1 - \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i) D^P_{ij} \end{aligned} \tag{4.38}$$

Putting it all together:

$$J_{ij} = \frac{1}{t}\sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i(1 - \sigma_{\gamma,\tau}(\boldsymbol{D}^P\boldsymbol{v})_i) D^P_{ij} - \delta_{ij} \tag{4.39}$$

To write the above in matrix form, we introduce the notation $\boldsymbol{diag}(\boldsymbol{u})$ for the diagonal matrix with main diagonal entries taken from vector $\boldsymbol{u}$. Then:

$$\boldsymbol{J}_{\boldsymbol{v}} = \frac{1}{\tau}\boldsymbol{diag}\big(\sigma_{\gamma,\tau}(\boldsymbol{D}^{\boldsymbol{P}}\boldsymbol{v}) \times (\boldsymbol{1} - \sigma_{\gamma,\tau}(\boldsymbol{D}^{\boldsymbol{P}}\boldsymbol{v}))\big) \cdot \boldsymbol{D}^P - \mathbb{I} \tag{4.40}$$

Where $\times$ is element-wise multiplication between vectors, and $\boldsymbol{1}$ is a vector with all entries set to 1.

Chapter 5

Non-Monotonic Deduction

In this chapter we move from the monotonic case of logic programming to the non-monotonic one. Stable semantics forms the basis of Answer Set Programming, and so we wish to leverage the methods developed in the previous chapter to search for stable models. In the case of normal programs, however, a direct matrix representation is more complicated. Sakama et al. (2017) mapped normal programs into third order tensors, but their size was exponential in the number of clauses.

In this chapter, we take a different approach to normal programs by using the concept of reducts to construct matrix representations. The differentiable approach from section 4.2 is then extended to the reduct case, and conditions on the parameters are shown. We next consider a vector mapping similar to section 4.3, and derive the Jacobian for the reduct case. We finish the chapter by exploring a few example programs that show how the algorithm fares in practice.

5.1 Reduct Matrix

Suppose we are given a normal program $P = \{\langle h^r, B^r, C^r \rangle\}_{r=1}^{R}$ and an interpretation M. We would like to construct a matrix representation of the reduct P^M. It is possible to compute the reduct program P^M symbolically and, as it is a Horn program, construct its matrix representation as described above. However, we would like a completely linear algebraic characterization of reducts instead.

Consider a single clause $\langle h^r, B^r, C^r \rangle$. Recall that $|C^r| = m^r$ and note:

$$(\mathbf{1} - \boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r} = \left\{ \begin{array}{cc} m^r & M \cap C^r = \emptyset \\ < m^r & M \cap C^r \neq \emptyset \end{array} \right. \tag{5.1}$$

Meaning that:

$$H_1\left(\frac{(\mathbf{1} - \boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}}{m^r}\right) = \left\{ \begin{array}{cc} 1 & M \cap C^r = \emptyset \\ 0 & M \cap C^r \neq \emptyset \end{array} \right. \tag{5.2}$$

So we can use the term $\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}}{m^r}$ to 'turn off' clauses where $M \cap C^r \neq \emptyset$. In particular, given an arbitrary interpretation I, we have:

$$H_1\left(\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}}{m^r} \cdot \frac{1}{n^r}\boldsymbol{v}^{h^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I\right) = \begin{cases} \boldsymbol{v}^{h^r} & B^r \subseteq I,\ M \cap C^r = \emptyset \\ \mathbf{0} & B^r \not\subseteq I \text{ or } M \cap C^r \neq \emptyset \end{cases} \tag{5.3}$$

We therefore propose to encode a normal clause as:

$$\boldsymbol{D}^{r^M} = \frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}}{n^r m^r}\boldsymbol{v}^{h^r}(\boldsymbol{v}^{B^r})^\intercal \tag{5.4}$$

Assuming P satisfies the MD condition, we can sum over all the clauses to construct a matrix representation of the reduct P^M:

$$\boldsymbol{D}^{P^I} = \sum_{r=1}^{R} \boldsymbol{D}^{r^I} = \sum_{r \in P} \frac{(\mathbf{1}-\boldsymbol{v}^I)^\intercal \boldsymbol{v}^{C^r}}{n^r m^r}\boldsymbol{v}^{h^r}(\boldsymbol{v}^{B^r})^\intercal \tag{5.5}$$

Which, analogously to equation 4.3, is equivalent to:

$$\boldsymbol{D}^{P^I} = \sum_{p \in B_P} \boldsymbol{v}^p \sum_{head(r)=p} \frac{(\mathbf{1}-\boldsymbol{v}^I)^\intercal \boldsymbol{v}^{C^r}}{n^r m^r}(\boldsymbol{v}^{B^r})^\intercal \tag{5.6}$$

Note that if P is a Horn program, meaning $C^r = \{\bot\}$ for all r, then for a consistent interpretation I we always have $\frac{(\mathbf{1}-\boldsymbol{v}^I)^\intercal \boldsymbol{v}^{C^r}}{m^r} = 1$, and the definition of $\boldsymbol{D}^{P^I}$ reduces to the regular representation of the Horn program $\boldsymbol{D}^P$.

Example: Consider the program:

$$\begin{aligned} &p \leftarrow \text{not } q \\ &q \leftarrow \text{not } r \\ &r \leftarrow \end{aligned} \tag{5.7}$$

We have:

$$\boldsymbol{v}^{h^1} = \boldsymbol{v}^p = \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{B^1} = \boldsymbol{v}^{\{\top\}} = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{C^1} = \boldsymbol{v}^{\{q\}} = \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \tag{5.8}$$

$$\boldsymbol{v}^{h^2} = \boldsymbol{v}^q = \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \quad \boldsymbol{v}^{B^2} = \boldsymbol{v}^{\{\top\}} = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{C^2} = \boldsymbol{v}^{\{r\}} = \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix} \tag{5.9}$$

$$\boldsymbol{v}^{h^3} = \boldsymbol{v}^{r} = \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix} \quad \boldsymbol{v}^{B^3} = \boldsymbol{v}^{\{\top\}} = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{C^3} = \boldsymbol{v}^{\{\bot\}} = \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} \tag{5.10}$$

Suppose we are given the interpretation $\{p, r\}$. Then:

$$\boldsymbol{v}^{\{p,r\}} = \begin{bmatrix}0\\1\\1\\0\\1\end{bmatrix} \qquad \boldsymbol{1} - \boldsymbol{v}^{\{p,r\}} = \begin{bmatrix}1\\0\\0\\1\\0\end{bmatrix} \tag{5.11}$$

Each clause is now encoded into a matrix. For the clause '$p \leftarrow$ not q' we have:

$$\begin{aligned} \frac{(\boldsymbol{1} - \boldsymbol{v}^{\{p,r\}})^\intercal \boldsymbol{v}^{C^1}}{|B^1||C^1|} \boldsymbol{v}^{h^1} (\boldsymbol{v}^{B^1})^\intercal &= \frac{\overbrace{\begin{bmatrix}1 & 0 & 0 & 1 & 0\end{bmatrix} \cdot \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}}^{=1}}{1 \cdot 1} \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix} \cdot \begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix} = \\ &= \begin{bmatrix}0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\\0 & 1 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\end{bmatrix} \end{aligned} \tag{5.12}$$

For '$q \leftarrow$ not r' we have:

$$\frac{(\boldsymbol{1} - \boldsymbol{v}^{\{p,r\}})^\intercal \boldsymbol{v}^{C^2}}{|B^2||C^2|} \boldsymbol{v}^{h^2} (\boldsymbol{v}^{B^2})^\intercal = \frac{\overbrace{\begin{bmatrix}1 & 0 & 0 & 1 & 0\end{bmatrix} \cdot \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}}^{=0}}{1 \cdot 1} \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \cdot \begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix} =$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.13}$$

Finally, for '$r \leftarrow$' we have:

$$\frac{(\mathbf{1} - \boldsymbol{v}^{\{p,r\}})^\intercal \boldsymbol{v}^{C^3}}{|B^3||C^3|} \boldsymbol{v}^{h^3} (\boldsymbol{v}^{B^3})^\intercal = \frac{\overbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}^{=1}}{1 \cdot 1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} =$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.14}$$

Adding these together, in addition to encoding '$\top \leftarrow \top, \text{not } \bot$', we get:

$$\boldsymbol{D}^{P^{\{p,r\}}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.15}$$

which represents the Horn program:

$$\begin{aligned} p &\leftarrow \\ r &\leftarrow \end{aligned} \tag{5.16}$$

as desired. We next show the correctness of the reduct mapping.

Proposition 5.1.1. *Suppose P is a normal program satisfying the MD condition and let I, J, M be interpretations. Then $J = T_{P^M}(I)$ if and only if $\boldsymbol{v}^J = H_1(\boldsymbol{D}^{P^M} \boldsymbol{v}^I)$ where H_1 is applied element-wise.*

Proof. Denote $\boldsymbol{u} = H_1(\boldsymbol{D}^{P^M} \boldsymbol{v}^I)$. Similarly to equation 4.11 of Horn program case, for any $p \in B_P$ we have:

$$(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = H_1\left(\sum_{head(r)=p} \frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r m^r}\right) \tag{5.17}$$

Suppose $J = T_{P^M}(I)$. If $p \in J$, then $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 1$ and there exists a rule $\langle p, B^i, \{\bot\}\rangle \in P^M$ such that $B^r \subseteq I$ and $M \cap C^r = \emptyset$. Therefore $(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I = n^r$ and $(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r} = m^r$ which implies $H_1\left(\sum_{head(r)=p} \frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r m^r}\right) = 1$ and therefore $(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = 1$. If $p \notin J$ then $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 0$ and for every rule $r = \langle p, B^r, C^r\rangle \in P$, one of the following applies:

- r is the only long rule with p as its head, and either $B^r \not\subseteq I$ and hence $(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I < n^r$ or $C^r \cap M \neq \emptyset$ and hence $(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r} < m^r$. Either way we have $\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r m^r} < 1$.
- r is a short rule and either $B^r \not\subseteq I$ and hence $(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I = 0$ or $C^r \cap M \neq \emptyset$ and hence $(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r} = 0$. Either way we have $\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r m^r} = 0$

We hence get $H_1\left(\sum_{head(r)=p} \frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^I}{n^r m^r}\right) = 0$ which means $(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = 0$. All of these together imply $\boldsymbol{u} = \boldsymbol{v}^J$.

Conversely, suppose $\boldsymbol{v}^J = H_1(\boldsymbol{D}^{P^M}\boldsymbol{v}^I)$. We know that $\boldsymbol{v}^{T_{P^M}(I)} = H_1(\boldsymbol{D}^{P^M}\boldsymbol{v}^I)$, which means $\boldsymbol{v}^J = \boldsymbol{v}^{T_{P^M}(I)}$. From the definition of the vector representation of a set of atoms, this implies $J = T_{P^M}(I)$.

□

Example: Continuing from the example above, where $M = \{p, r\}$, suppose $I = \emptyset$. Then $J = \{p, r\}$. Now, consider an atom in J such as p. $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 1$ (it is the entry for p in $\boldsymbol{v}^J$, so we must show that $(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = 1$. Since $p \in J$, there is a rule with p as its head that is satisfied. This is the rule $p \leftarrow$ not q. Because of this rule, we have:

$$\sum_{head(r)=p} \frac{(\mathbf{1}-\boldsymbol{v}^{\{p,r\}})^\intercal \boldsymbol{v}^{C^r}(\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}^{\emptyset}}{n^r m^r} > \frac{(\mathbf{1}-\boldsymbol{v}^{\{p,r\}})^\intercal \boldsymbol{v}^{C^1}(\boldsymbol{v}^{B^1})^\intercal \boldsymbol{v}^{\emptyset}}{n^1 m^1} = \tag{5.18}$$

$$= \frac{\overbrace{\begin{bmatrix}1 & 0 & 0 & 1 & 0\end{bmatrix} \cdot \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}}^{=1} \overbrace{\begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix} \cdot \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}}^{=1}}{1 \cdot 1} = 1$$

So the rule that satisfies p causes the argument of H_1 to be at least 1, and hence $(\boldsymbol{v}^p)^\intercal \boldsymbol{u} = 1$. This can similarly be shown for r (and $\top$, through the implicit rule

$\top \leftarrow \top, \text{not } \bot$). For q, we have $q \notin J$, meaning any rule with q as its head was not satisfied. In our example, there is only one short rule with q as its head: $q \leftarrow \text{not } r$. Short rules that are not satisfied always contribute 0 to the H_1's argument as we can see:

$$\frac{(\mathbf{1}-\boldsymbol{v}^{\{p,r\}})^\intercal \boldsymbol{v}^{C^2}(\boldsymbol{v}^{B^2})^\intercal \boldsymbol{v}^{\emptyset}}{n^2m^2} = \frac{\overbrace{\begin{bmatrix}1&0&0&1&0\end{bmatrix}\cdot\begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}}^{=0}\overbrace{\begin{bmatrix}0&1&0&0&0\end{bmatrix}\cdot\begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}}^{=1}}{1\cdot 1} = 0 \tag{5.19}$$

So in our case we see that H_1 receives 0 as an argument and hence $(\boldsymbol{v}^q)^\intercal \boldsymbol{u} = 0$.

Long rules that are not satisfied can contribute 0 or a positive number less than 1, since both $\frac{(\mathbf{1}-\boldsymbol{v}^{\{p,r\}})^\intercal \boldsymbol{v}^{C^r}}{m^r}$ and $\frac{(\boldsymbol{v}^{B^1})^\intercal \boldsymbol{v}^{\emptyset}}{n^r}$ are upper bounded by 1. This means that as long as an atom is defined by one long rule at the most, and none of the rules are satisfied, then the argument for H_1 is less than 1. However, if q were defined by multiple long rules, their contributions can add together to a sum greater than 1, and q would be incorrectly assigned 1. Hence the MD condition is necessary.

5.2 Differentiable Reducts

Just as was the case with Horn programs, we can replace H_1 in our deduction operation above with a sigmoid $\sigma_{\gamma,\tau}$. As before, we need to introduce two additional constants $0 < \gamma^\bot < \gamma < \gamma^\top < 1$, and derive bounds for these values.

Suppose there are R_P clauses with $p \in B_p$ as their head. At most one of these rules is long, denote it $\langle p, B_p^1, C_p^1\rangle$. The rest are short rules of the form $\langle p, \{b_p^r\}, \{c_p^r\}\rangle$ for $r = 2 \ldots R_P$. Given an interpretation M, the reduct representation of these rules is:

$$\boldsymbol{D}^{p^M} = \frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C_p^1}}{n_p^1 m_p^1}\boldsymbol{v}^p(\boldsymbol{v}^{B_p^1})^\intercal + \sum_{r=2}^{R_p}(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{c_p^r}\boldsymbol{v}^p(\boldsymbol{v}^{b_p^r})^\intercal \tag{5.20}$$

Given an interpretation I, which may be different from M, we have:

$$\sigma_{\gamma,\tau}(\boldsymbol{D}^{p^M}\boldsymbol{v}^I) = \sigma_{\gamma,\tau}\left(\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C_p^1}(\boldsymbol{v}^{B_p^1})^\intercal \boldsymbol{v}^I}{n_p^1 m_p^1} + \sum_{r=2}^{R_p}(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{c_p^r}(\boldsymbol{v}^{b_p^r})^\intercal \boldsymbol{v}^I\right)\boldsymbol{v}^p \tag{5.21}$$

Denote:

$$x = \frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C_p^1}(\boldsymbol{v}^{B_p^1})^\intercal \boldsymbol{v}^I}{n_p^1 m_p^1} + \sum_{r=2}^{R_p} (\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{c_p^r}(\boldsymbol{v}^{b_p^r})^\intercal \boldsymbol{v}^I \tag{5.22}$$

We require:

- If $B_p^1 \subseteq I$ and $C_p^1 \cap M = \emptyset$ then $\sigma_{\gamma,\tau}(x) > \gamma^\top$. $B_p^1 \subseteq I$ implies $\frac{(\boldsymbol{v}^{B_p^1})^\intercal \boldsymbol{v}^I}{n_p^1} > \gamma^\top$ and $C_p^1 \cap M = \emptyset$ implies $\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C_p^1}}{m_p^1} > 1-\gamma^\perp$.
- If $b_p^r \in I$ and $c_p^r \notin M$ then $\sigma_{\gamma,\tau}(x) > \gamma^\top$. $b_p^r \in I$ implies $(\boldsymbol{v}^{b_p^r})^\intercal \boldsymbol{v}^I > \gamma^\top$ and $c_p^r \notin M$ implies $(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C_p^1} > 1-\gamma^\perp$.

The two conditions above can be summarized by:

$$x > (1-\gamma^\perp)\gamma^\top \qquad \Rightarrow \qquad \sigma_{\gamma,\tau}(x) > \gamma^\top \tag{5.23}$$

Then we get an upper bound for τ by:

$$\sigma_{\gamma,\tau}(x) = \frac{1}{1+e^{\frac{\gamma-x}{\tau}}} \underset{x>(1-\gamma^\perp)\gamma^\top}{>} \frac{1}{1+e^{\frac{\gamma-(1-\gamma^\perp)\gamma^\top}{\tau}}} \tag{5.24}$$

$$\frac{1}{1+e^{\frac{\gamma-(1-\gamma^\perp)\gamma^\top}{\tau}}} > \gamma^\top \quad \Leftrightarrow \quad \tau < \frac{\gamma-(1-\gamma^\perp)\gamma^\top}{\ln(\frac{1}{\gamma^\top}-1)} \tag{5.25}$$

As was the case of Horn programs, we require in 5.25 that $\gamma^\top > \frac{1}{2}$ but also $\gamma < (1-\gamma^\perp)\gamma^\top$.

Next, we consider the case when p is not satisfied. For the long rule $\langle p, B_p^1, C_p^1 \rangle$ one of the following conditions apply:

- $B_p^1 \not\subseteq I$. In this case we have $\frac{(\boldsymbol{v}^{B_p^1})^\intercal \boldsymbol{v}^I}{n_p^1} < \frac{n_p^1 - 1 + \gamma^\perp}{n_p^1}$ and $\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C_p^1}}{m_p^1} \leq 1$.
- $C_p^1 \cap M \neq \emptyset$. In this case we have $\frac{(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{C_p^1}}{m_p^1} < \frac{m_p^1 - \gamma^\top}{m_p^1}$ and $\frac{(\boldsymbol{v}^{B_p^1})^\intercal \boldsymbol{v}^I}{n_p^1} \leq 1$.

In addition, for all $R_p - 1$ short rules one of the following applies:

- $b_p^r \notin I$. In this case we have $(\boldsymbol{v}^{B_p^1})^\intercal \boldsymbol{v}^I < \gamma^\perp$ and $(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{c_p^r} \leq 1$.
- $c_p^r \in M$. In this case we have $(\mathbf{1}-\boldsymbol{v}^M)^\intercal \boldsymbol{v}^{c_p^r} < 1-\gamma^\top$ and $(\boldsymbol{v}^{B_p^1})^\intercal \boldsymbol{v}^I \leq 1$.

Putting it all together, we have:

$$x < \max\big\{\frac{n_p^1 - 1 + \gamma^\perp}{n_p^1}, \frac{m_p^1 - \gamma^\top}{m_p^1}\big\} + (R_p - 1)\max\big\{\gamma^\perp, 1 - \gamma^\top\big\} \triangleq x^\perp \tag{5.26}$$

And we require:

$$x < x^\perp \qquad \Rightarrow \qquad \sigma_{\gamma,\tau}(x) < \gamma^\perp \tag{5.27}$$

So we get an upper bound for τ by:

$$\sigma_{\gamma,\tau}(x) = \frac{1}{1 + e^{\frac{\gamma - x}{\tau}}} \underset{x<x^\perp}{<} \frac{1}{1 + e^{\frac{\gamma - x^\perp}{\tau}}} \tag{5.28}$$

$$\begin{aligned} \frac{1}{1 + e^{\frac{\gamma - x^\perp}{\tau}}} < \gamma^\perp \quad &\Leftrightarrow \quad \tau < \frac{\gamma - x^\perp}{\ln(\frac{1}{\gamma^\perp} - 1)} \\ &\Leftrightarrow \quad \tau < \frac{\gamma - \max\big\{\frac{n_p^1 - 1 + \gamma^\perp}{n_p^1}, \frac{m_p^1 - \gamma^\top}{m_p^1}\big\} - (R_p - 1)\max\big\{\gamma^\perp, 1 - \gamma^\top\big\}}{\ln(\frac{1}{\gamma^\perp} - 1)} \end{aligned} \tag{5.29}$$

And for equation 5.29 we require $\gamma^\perp < \frac{1}{2}$ but also:

$$\gamma > \max\left\{\frac{n_p^1 - 1 + \gamma^\perp}{n_p^1}, \frac{m_p^1 - \gamma^\top}{m_p^1}\right\} + (R_p - 1)\max\{\gamma^\perp, 1 - \gamma^\top\} \tag{5.30}$$

For this inequality to have a solution, γ would have to be strictly greater than both $\frac{n_p^1 - 1}{n_p^1}$ and $\frac{m_p^1 - 1}{m_p^1}$.

Putting it all together, we select $\gamma, \gamma^\perp, \gamma^\top, \tau$ such that:

1. $\gamma > \max_p\big\{\frac{n_p^1 - 1}{n_p^1}, \frac{m_p^1 - 1}{m_p^1}\big\}$.
2. $\gamma < (1 - \gamma^\perp)\gamma^\top$.
3. $\gamma > \max_p\left\{\max\big\{\frac{n_p^1 - 1 + \gamma^\perp}{n_p^1}, \frac{m_p^1 - \gamma^T}{m_p^1}\big\} + (R_p - 1)\max\big\{\gamma^\perp, 1 - \gamma^\top\big\}\right\}$
4. $\gamma^\top > \frac{1}{2}$.
5. $\gamma^\perp < \frac{1}{2}$.
6. $\tau < \min_p\left\{\frac{\gamma - (1 - \gamma^\perp)\gamma^\top}{\ln(\frac{1}{\gamma^\top} - 1)}, \frac{\gamma - \max\left\{\frac{n_p^1 - 1 + \gamma^\perp}{n_p^1}, \frac{m_p^1 - \gamma^\top}{m_p^1}\right\} - (R_p - 1)\max\left\{\gamma^\perp, 1 - \gamma^\top\right\}}{\ln(\frac{1}{\gamma^\perp} - 1)}\right\}$

To show one possible solution, denote:

$$n = \max_p \{n_p^1, m_p^1\} \qquad r = \max_p \{R_p\} \tag{5.31}$$

We can set $\gamma = \frac{n-\frac{1}{2}}{n}$ or $\gamma = \frac{n}{n+1}$, for example. We can also decide that $\gamma^\perp < 1 - \gamma^\top$. So condition 3 becomes:

$$\gamma > \frac{n - \gamma^\top}{n} + (r-1)(1-\gamma^\top) \tag{5.32}$$

Which gives us:

$$\gamma^\top > \frac{r - \gamma}{\frac{1}{n} + r - 1} \tag{5.33}$$

So we can set:

$$\gamma^\top = \frac{1}{2}\left(1 + \max\left\{\frac{r-\gamma}{\frac{1}{n} + k - 1}, \frac{1}{2}\right\}\right) \tag{5.34}$$

And:

$$\gamma^\perp = \frac{1}{2}(1 - \gamma^T) \tag{5.35}$$

And finally:

$$\tau = \min\left\{\frac{\gamma - (1-\gamma^\perp)\gamma^\top}{\ln(\frac{1}{\gamma^\top} - 1)}, \frac{\gamma - \frac{n-1}{n} - (r-1)\gamma^\perp}{\ln(\frac{1}{\gamma^\perp} - 1)}\right\} \tag{5.36}$$

The following proposition ensures our differentiable operator correctly realizes application of T_{PM} on an interpretation.

Proposition 5.2.1. *Let P be normal program and I, M interpretations. Suppose $\boldsymbol{u} \sim_{\gamma^F}^{\gamma^T} \boldsymbol{v}^I$. Then $H_1(\boldsymbol{D}^{P^M}\boldsymbol{v}^I) \sim_{\gamma^F}^{\gamma^T} \sigma_{\gamma,t}(\boldsymbol{D}^{P^M}\boldsymbol{u})$.*

Proof. Denote $\boldsymbol{v}^J = H_1(\boldsymbol{D}^P \cdot \boldsymbol{v}^I)$ and let $p \in B_P$. Suppose $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 1$. Then there exists a rule $r \in P$ such that $head(r) = p$ and for all $b \in body^+(r)$ we have $(\boldsymbol{v}^b)^\intercal \boldsymbol{v}^I = 1$ and for all $c \in body^-(r)$ we have $(\boldsymbol{v}^c)^\intercal \boldsymbol{v}^I = 0$. Hence for all $b \in body^+(r)$ we have $(\boldsymbol{v}^b)^\intercal \boldsymbol{u} > \gamma^\top$ and for all $c \in body^-(r)$ we have $(\boldsymbol{v}^c)^\intercal \boldsymbol{u} < \gamma^\perp$ and as we've seen, for a suitable choice of γ and τ we are assured $\sigma_{\gamma,\tau}(\boldsymbol{D}^{P^M} \cdot \boldsymbol{u}) > \gamma^\top$ as required.
Suppose now that $(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^J = 0$. Then for every rule such that $head(r) = p$ one of the following occurs:

- There exists at least one atom $b \in \mathit{body}^+(r)$ such that $(\boldsymbol{v}^b)^\intercal \boldsymbol{v}^I = 0$ and hence $(\boldsymbol{v}^b)^\intercal \boldsymbol{u} < \gamma^\perp$. For a suitable choice of parameters we are assured $\sigma_{\gamma,\tau}(\boldsymbol{D}^{P^M} \cdot \boldsymbol{u}) < \gamma^\perp$.
- There exists at least one atom $c \in \mathit{body}^-(r)$ such that $(\boldsymbol{v}^c)^\intercal \boldsymbol{v}^I = 1$ and hence $(\boldsymbol{v}^c)^\intercal \boldsymbol{u} > \gamma^\top$. For a suitable choice of parameters we are assured $\sigma_{\gamma,\tau}(\boldsymbol{D}^{P^M} \cdot \boldsymbol{u}) < \gamma^\perp$.

□

5.3 Searching for Stable Models

Proposition 5.3.1. *Let P be a normal program. Then M is a supported model of P if and only if $\boldsymbol{v}^M = H_1(D^{P^M} \boldsymbol{v}^M)$.*

Proof. It is easy to see that $T_{P^M}(M) = T_P(M)$. The proposition then follows immediately from propositions 2.2.1 and 5.1.1. □

As was the case with Horn programs, we can define a mapping based on the above proposition to turn deduction into root search. This time, however, the matrix $\boldsymbol{D}^{\boldsymbol{v}^I}$ depends on the input vector $\boldsymbol{v}$. We denote:

$$\boldsymbol{D}^{P^{\boldsymbol{v}}} = \sum_{r \in P} \frac{(\boldsymbol{1} - \boldsymbol{v})^\intercal \boldsymbol{v}^{C^r}}{n^r m^r} \boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal \tag{5.37}$$

Then our mapping is now:

$$F(\boldsymbol{v}) = \sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}}} \boldsymbol{v}) - \boldsymbol{v} \tag{5.38}$$

Deriving the Jacobian starts off very similar to section 4.3. The derivation starts to deviate at:

$$\frac{\partial \sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}}} \boldsymbol{v})_i}{\partial v_j} = \frac{1}{\tau} \sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}}} \boldsymbol{v})_i (1 - \sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}}} \boldsymbol{v})_i) \cdot \frac{\partial \sum\limits_{k=1}^{N} D_{ik}^{P^{\boldsymbol{v}}} v_k}{\partial v_j} \tag{5.39}$$

Since now $D_{ik}^{P^{\boldsymbol{v}}}$ depends on $\boldsymbol{v}$. We therefore have to use the product rule for differentiation:

$$\frac{\partial \sum\limits_{k=1}^{N} D_{ik}^{P^{\boldsymbol{v}}} v_k}{\partial v_j} = \sum_{k=1}^{N} \left(\frac{\partial D_{ik}^{P^{\boldsymbol{v}}}}{\partial v_j} v_k + D_{ik}^{P^{\boldsymbol{v}}} \frac{\partial v_k}{\partial v_j} \right) = \sum_{k=1}^{N} \left(\frac{\partial D_{ik}^{P^{\boldsymbol{v}}}}{\partial v_j} v_k + D_{ik}^{P^{\boldsymbol{v}}} \delta_{kj} \right) \tag{5.40}$$

Now:

$$D_{ik}^{P^{\boldsymbol{v}}} = \sum_{r \in P} (\mathbf{1} - \boldsymbol{v})^\intercal \boldsymbol{v}^{C^r} \cdot \left[\frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal}{n^r m^r} \right]_{ik} \tag{5.41}$$

So:

$$\frac{\partial D_{ik}^{P^{\boldsymbol{v}}}}{\partial v_j} = \sum_{r \in P} \frac{\partial (\mathbf{1} - \boldsymbol{v})^\intercal \boldsymbol{v}^{C^r}}{\partial v_j} \cdot \left[\frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal}{n^r m^r} \right]_{ik} = -\sum_{r \in P} v_j^{C^r} \cdot \left[\frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal}{n^r m^r} \right]_{ik} \tag{5.42}$$

Hence:

$$\begin{aligned} \sum_{k=1}^{N} \left(\frac{\partial D_{ik}^{P^{\boldsymbol{v}}}}{\partial v_j} v_k + D_{ik}^{P^{\boldsymbol{v}}} \delta_{kj} \right) &= D_{ij}^{P^{\boldsymbol{v}}} - \sum_{k=1}^{N} \sum_{r \in P} \left(v_j^{C^r} \left[\frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal}{n^r m^r} \right]_{ik} v_k \right) = \\ &= D_{ij}^{P^{\boldsymbol{v}}} - \sum_{r \in P} \left(v_j^{C^r} \left[\frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v}}{n^r m^r} \right]_{i} \right) = \\ &= D_{ij}^{P^{\boldsymbol{v}}} - \sum_{r \in P} \left[\frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v} (\boldsymbol{v}^{C^r})^\intercal}{n^r m^r} \right]_{ij} \end{aligned} \tag{5.43}$$

So the Jacobian is given by:

$$J_{ij} = \frac{1}{\tau} \sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}}} \boldsymbol{v})_i (1 - \sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}}} \boldsymbol{v})_i) \cdot \left(D_{ij}^{P^{\boldsymbol{v}}} - \left[\sum_{r \in P} \frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v} (\boldsymbol{v}^{C^r})^\intercal}{n^l m^l} \right]_{ij} \right) - \delta_{ij} \tag{5.44}$$

Which in matrix form can be written as:

$$\boldsymbol{J_v} = \frac{1}{\tau} \boldsymbol{diag}\big(\sigma_{\gamma,\tau}(\boldsymbol{D^{P^v}} \boldsymbol{v}) \times (\mathbf{1} - \sigma_{\gamma,\tau}(\boldsymbol{D^{P^v}} \boldsymbol{v})) \big) \cdot \left(\boldsymbol{D}^{P^{\boldsymbol{v}}} - \sum_{r \in P} \frac{\boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal \boldsymbol{v} (\boldsymbol{v}^{C^r})^\intercal}{n^r m^r} \right) - \mathbb{I} \tag{5.45}$$

Chapter 6

Case Studies

In this chapter we study how the algorithm developed in section 5.3 works in practice. A deduction algorithm based on real-valued and differentiable interpretations requires a different intuition to algorithms based on symbolic reasoning alone. To develop this intuition, we explore three different cases.

The first case study explores a basic non-monotonic deductive task. We explore the effects of temperatures on convergence to stable models and see we get high convergence rates when τ is set to relatively high values. We then study a simple stratified program to ensure our algorithm correctly converges to its unique stable model. Finally, we examine a larger non-monotonic program to develop intuition on how convergence changes when dependency between atoms becomes more complicated.

6.1 Case Study 1

Consider the following program:

$$\begin{aligned} p &\leftarrow \text{not}\, q \\ q &\leftarrow \text{not}\, p \end{aligned} \tag{6.1}$$

The program has four interpretations: $\emptyset, \{p\}, \{q\}$ and $\{p, q\}$. Two of these are supported (and stable) models: $\{p\}$ and $\{q\}$. Note that, even though this program is very simple, it embodies two main challenges in non-monotonic deduction: The fact that addition of new information invalidates previously acquired information (adding q to an interpretation invalidates p being true), and the fact that there is no special model (neither $\{p\}$ nor $\{q\}$ stands out as a preferred). It is therefore worthwhile to study how differentiable deduction fares in this case.

Suppose we wish to find stable models of the program. If we work symbolically, we need to perform a search over the space of interpretations. One very simple strategy is to guess an initial interpretation and check if it is stable by computing the Least Herbrand Model of the reduct. For the two stable models, we find the fixed point immediately after applying T_P. For the other two interpretations, a fixed point is never found since:

$$\begin{aligned} T_P(\{\}) &= \{p, q\} \\ T_P(\{p, q\}) &= \{\} \end{aligned} \tag{6.2}$$

We therefore have a 50% chance of finding a stable model in this manner. To explore the Newton-Raphson method for deduction, we consider the following algorithm:

Algorithm 1 Newton-Raphson Deduction

```
Require: $\epsilon > 0$, $MaxK \geq 1$, $0 < \gamma < 1$, $\tau > 0$
  Pick an initial vector:

  $$\boldsymbol{v}^0 = \begin{bmatrix} 0 \\ 1 \\ \alpha \\ \beta \end{bmatrix} \qquad \alpha, \beta \sim U[0,1] \tag{6.3}$$

  repeat
    Solve $\boldsymbol{J}_{\boldsymbol{v}^k}(\boldsymbol{v}^{k+1} - \boldsymbol{v}^k) = \boldsymbol{v}^k - \sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}^k}} \cdot \boldsymbol{v}^k)$
  until $||\boldsymbol{v}^{k+1} - \boldsymbol{v}^k||_2 < \epsilon$ or $k \geq MaxK$
  if $k < MaxK$ then

    return $\boldsymbol{v}^k$
  else

    return 'FAIL'
  end if
```

Algorithm 1 begins by selecting a consistent interpretation at random. The values for p and q are chosen from a uniform distribution over $[0, 1]$. At each iteration it performs a Newton-Raphson step. ϵ is a threshold parameter to test for convergence to a root. $MaxK$ is the maximum number of iterations before stopping, to ensure halting in case the algorithm does not converge. The algorithm can return an undesirable output under two scenarios: First, it may not converge to a root. Second, even if a root is found, it may not correspond to a supported model. We therefore seek to establish what the probability of finding a stable model is.

To answer this question, we implemented the algorithm, and tested how often we were able to converge to a stable model. We fixed the values of γ and τ and repeated the experiment 1000 times, each time starting from a different initial vector (selected randomly as described above). To test further the effect of temperature on success rate, we fixed $\gamma = \frac{1}{2}$ and varied the temperature, repeating the experiment 1000 times for each one. The proposed solution from equation 5.36 suggests the value of τ should not exceed 0.1422, so we varied its value from 0.01422 to 0.1422.

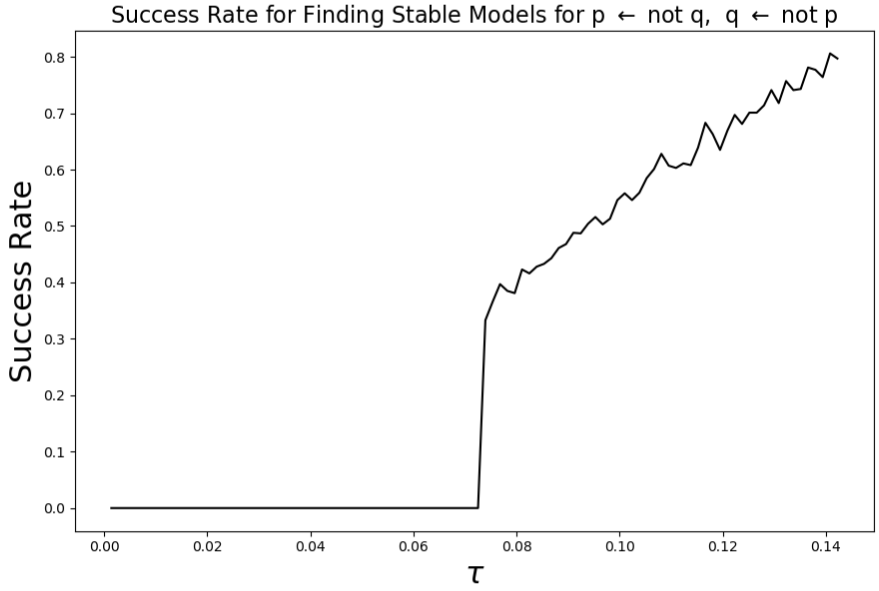


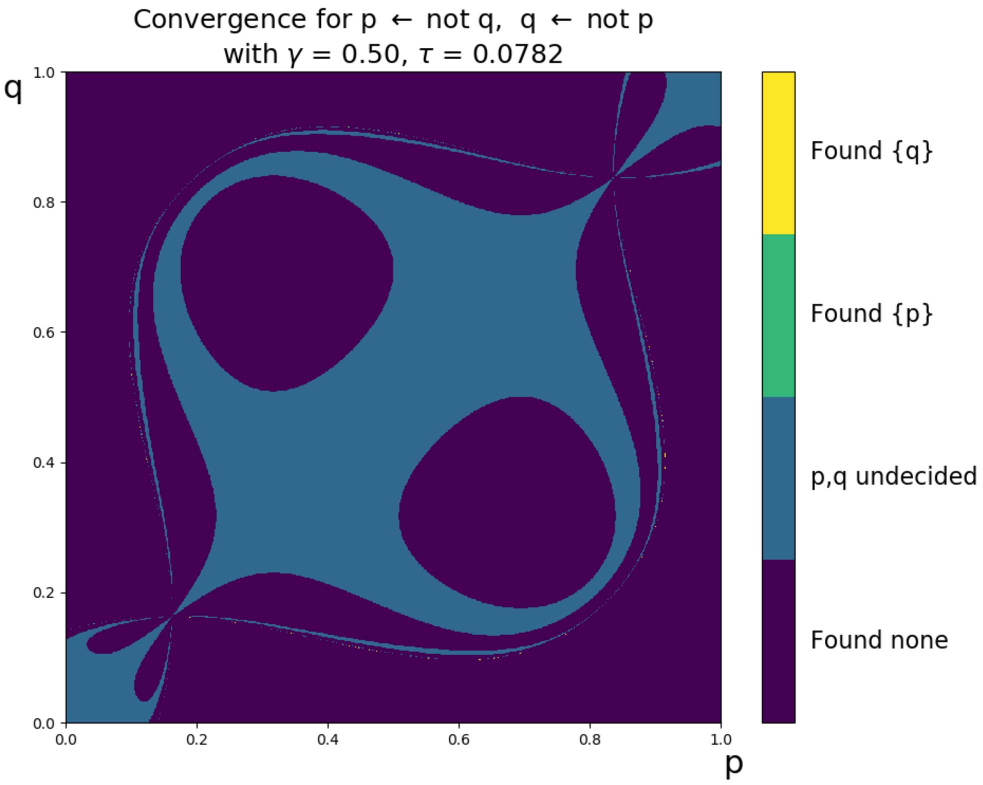
Figure 6.1: Success rate for finding a stable model, given a temperature. γ was fixed to a value of 0.5. Temperatures were varied from 0.01422 to 0.1422.

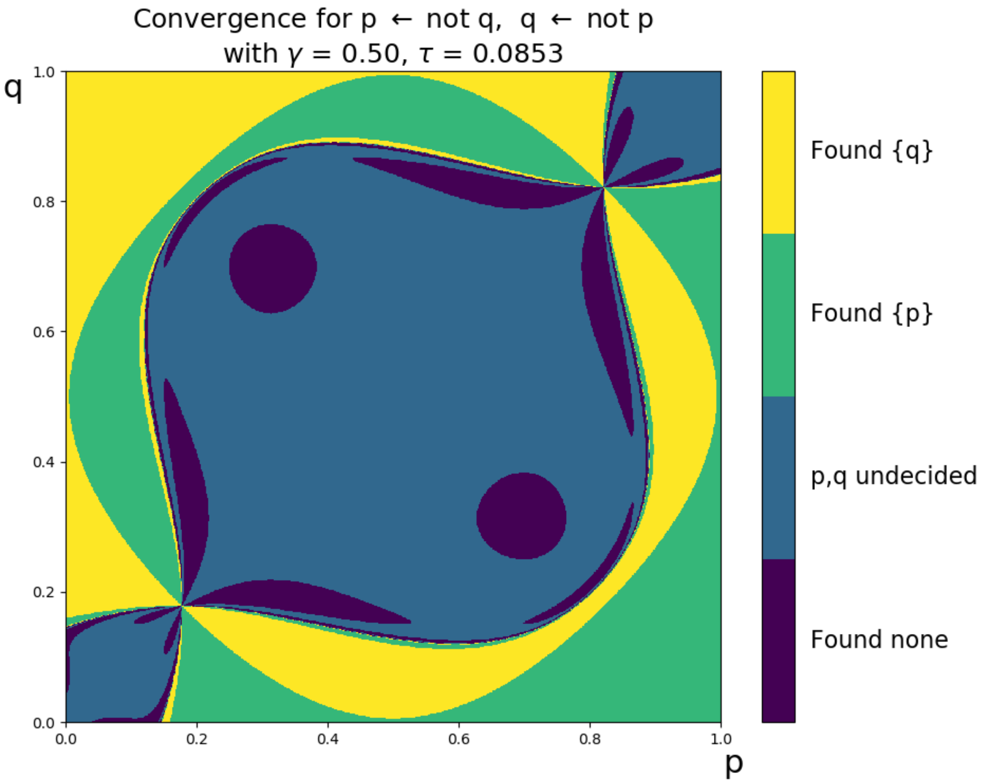
Graph 6.1 shows a clear picture of the algorithm's performance. For temperatures up to about 0.0782, we fail to find any models. For higher temperature, however, the success rate increases steadily (the line is jagged only due to randomness in the initial vector pick), up to about 80% for the highest temperatures tested.

Most importantly, we can see from figure 6.1 that our chance of success can be much higher than 50%. To better understand for which initial values we converge, another experiment was performed. We repeated the algorithm above yet again, but instead of selecting initial vectors at random, we divided the unit square $[0,1] \times [0,1]$ in $p-q$ space into a 1000×1000 grid, iterating over all points on the grid. In this manner, we can test convergence for each point and get a full picture of the convergence map in the unit square. For each point, we observed four possible outcomes:

1. The algorithm converged to the stable model $\{p\}$.
2. The algorithm converged to the stable model $\{q\}$.
3. The algorithm converged to a third root that does not correspond to an interpretation. In this case both p and q had a final value of about $\frac{1}{2}$ (equivalent to $1-\gamma$). We refer to this as p and q being 'undecided', in analogue to the concept of 'unknown' atoms in well-founded semantics (Van Gelder et al., 1991).
4. The algorithm did not converge to a root. We refer to this as 'finding none'.

The following charts show the convergence map for a few different temperature values. Each point has been coloured according to outcome of the algorithm, as described above.





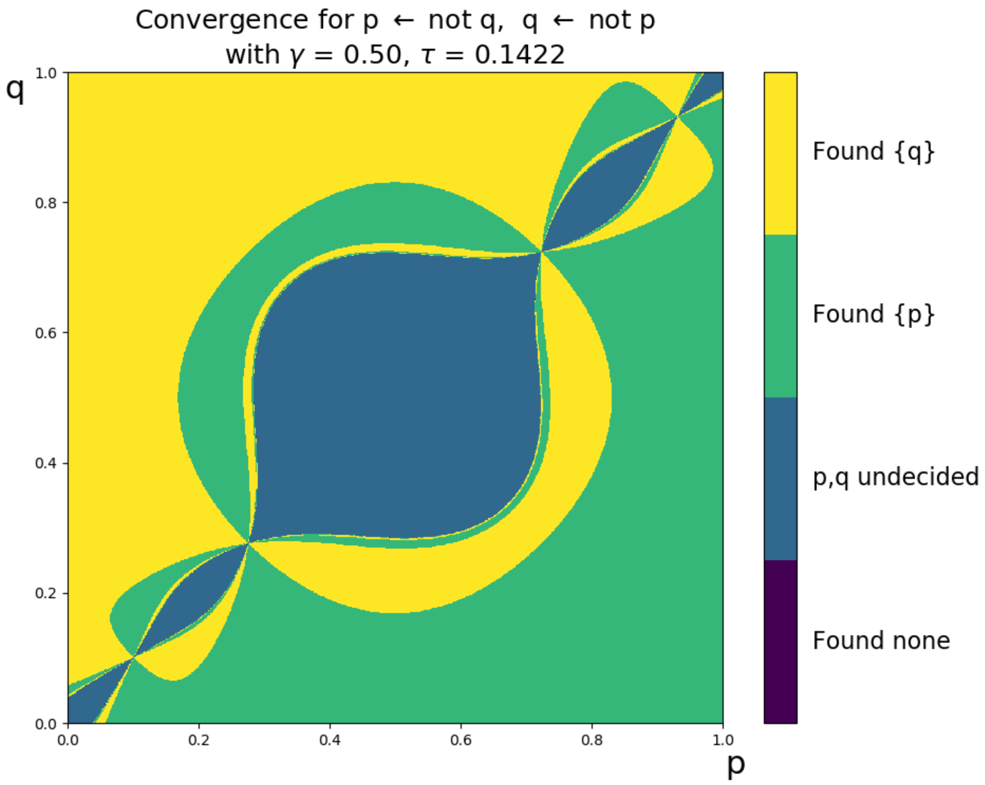


Figure 6.2: Convergence map for three different temperature values. The p and q axis indicate their initial value in $\boldsymbol{v}^0$. The colour indicates to which point the algorithm converged, if at all.

The top figure exemplifies the change that occurs in the temperature range of 0.7 to 0.8. For lower temperatures, the algorithm never converges. The chart here shows that at about 0.0782 the algorithm is already converging for many points, but mostly to a third, undesirable root.

The next figure shows that by the time a temperature reaches $0.0.853$, convergence to stable models increased substantially, and divergence is almost disappearing. At this point, the algorithm still finds the third root more than others. This changes in the last figure, where stable models are clearly dominating, explaining the high percentage of success we see in figure 6.1.

Some questions naturally arise when one looks at these figures. First, the meaning of the third root is worthy of some consideration. It can be seen to be a consequence of the program's symmetry. Indeed, were the root not there, an algorithm beginning at $(p, q) = (0, 0)$ and searching for a root could only converge to $\{p\}$ or $\{q\}$. However, neither of these models is preferable to the other, so such a scenario cannot arise. As an example, suppose we search for a root by performing gradient descent over $||F(\boldsymbol{v}^k)||^2$. We must converge to some local minimum in this scenario, but that local minimum cannot be $\{p\}$ or $\{q\}$ due to symmetry, and hence a third minimum exists.

We do not here give a full theoretical treatment for the origins of the third root. We speculate, however, that there is some connection to the well-founded model of the program (Van Gelder et al., 1991). In well-founded semantics, interpretations are three-valued, meaning that in addition to 'True' and 'False', atoms may also be assigned a value of 'Unknown' (or 'Undecided'). Under such semantics, the program we are considering has a unique well-founded model where both p and q are 'undecided', since neither can be shown to definitively be true. We leave the study of the connection between the third root and the well-founded model to future work.

A fact that may be surprising to the reader, when looking at the top figure, is that the algorithm does not appear to converge to $\{p\}$ or $\{q\}$ even when it appears to start from $\{p\}$ or $\{q\}$ ($(1, 0)$ and $(0, 1)$ respectively). The explanation for this is simple: $(1, 0)$ and $(0, 1)$ are not in fact the roots. Instead, the roots are points very close to $(1, 0)$ and $(0, 1)$. This is because the sigmoid function never outputs 0 or 1 but can output values close to them. What we are seeing is a prime example of numerical instability - when the temperatures are very low, the determinant of the Jacobian is very high (this can be seen in equation 5.45, where the temperature is in the denominator). This causes the algorithm to act erratically and 'overshoot' the root, even when it begins close to it.

Finally, the reader may also wonder why initial points that are closer to one stable model may end up converging to another one. For instance, in the bottom map the initial point $(p, q) = (0.3, 0.7)$ is clearly closer to $\{q\}$ but in fact the algorithm from that point converges to $\{p\}$. This phenomenon is best understood when one looks at the topology of $||F(\boldsymbol{v}^k)||_2 = ||\sigma_{\gamma,\tau}(\boldsymbol{D}^{P^{\boldsymbol{v}^k}} \cdot \boldsymbol{v}^k) - \boldsymbol{v}^k||_2$ over the unit square:

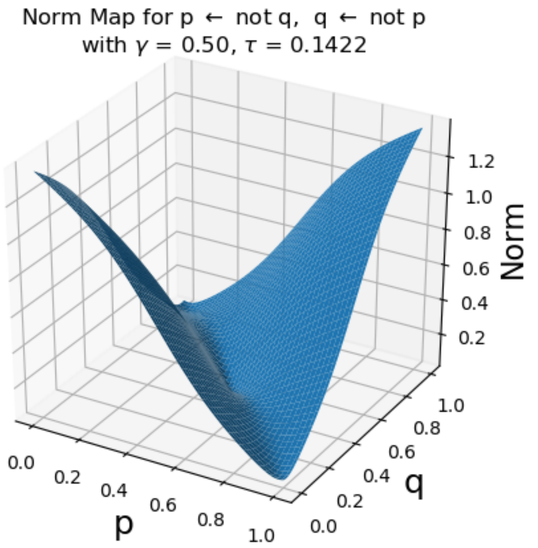
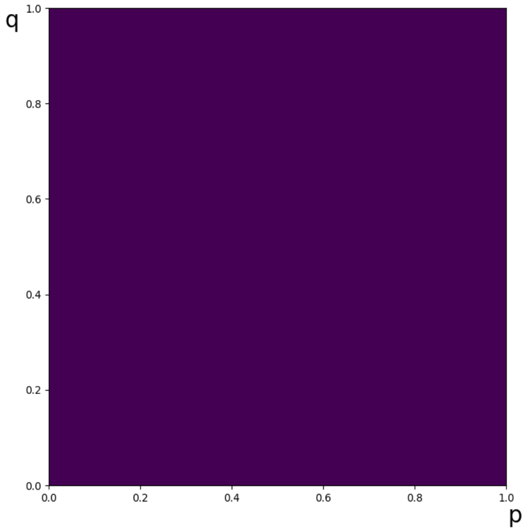
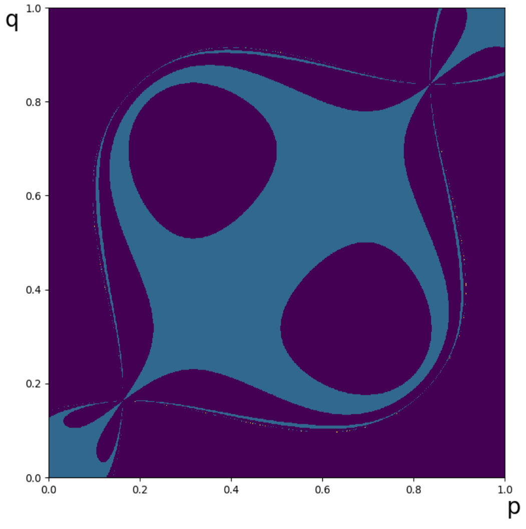


Figure 6.3: A plot of $||F(\boldsymbol{v}^k)||_2$ on the unit square. The two roots corresponding to stable models can be clearly seen, as well as the third root at $(\frac{1}{2}, \frac{1}{2})$. As we move from a stable model towards the third root, the direction of the slope changes.

Looking closely at the plot of the norm of F, we can see that even when one moves closer to one of the stable models such as $\{q\}$, the slope at that point may be directed towards the third root. A Newton-Raphson step would therefore be towards $(p, q) = \left(\frac{1}{2}, \frac{1}{2}\right)$, not towards $\{q\}$. However, a step may 'overshoot' that root, and end up closer to $\{p\}$. It will then converge to $\{p\}$ rather than $\left(\frac{1}{2}, \frac{1}{2}\right)$.

Convergence Map	Temperatures and Description
	$0.01422 < \tau < 0.0711$ For low temperatures, the algorithm never converges. Hence the entire map is filled with a single colour for 'Found None'.
	$\tau = 0.0782$ Around this temperature, the algorithm begins to converge for some values to p, q-undecided, which is the dominating root.

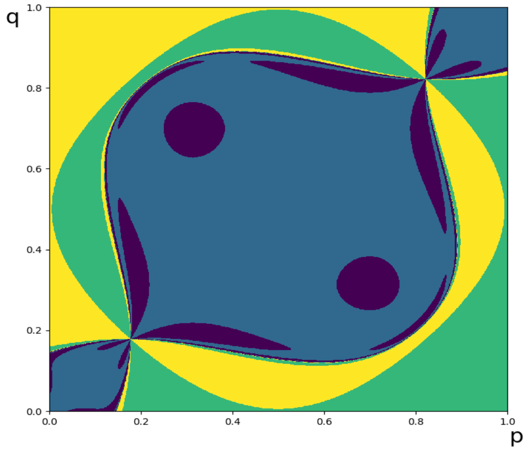
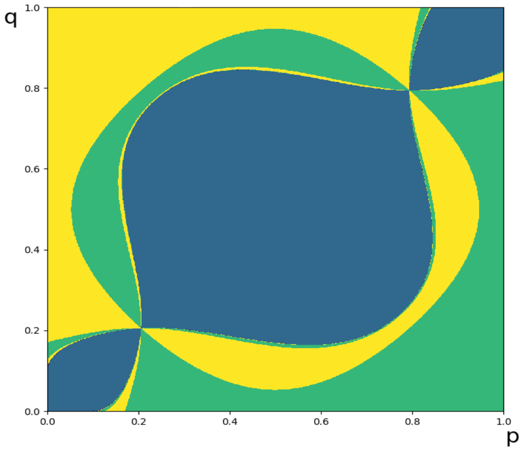
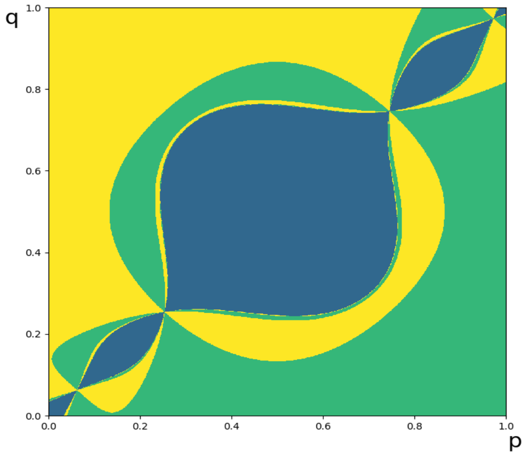
Convergence Map	Temperatures and Description
	$\tau = 0.0853$ Areas that result in no convergence are shrinking, and the algorithm begins to find stable models more often. p, q-undecided is still dominating.
	$\tau = 0.0996$ The algorithm now converges from every initial point, and stable models are found most often, and at about the same rate as p, q-undecided.
	$0.1138 < \tau < 0.1422$ The algorithm converges to stable models more often as the temperature increases. p, q-undecided areas are shrinking.

Table 6.1: Convergence maps for different temperatures, along with a description of convergence behaviour.

6.2 Case Study 2

Consider now the following program:

$$\begin{aligned} p &\leftarrow \text{not } q \\ q &\leftarrow \text{not } r \\ r &\leftarrow \end{aligned} \tag{6.4}$$

This program is stratified, and hence it has a unique stable model, namely $\{p, r\}$. In this case, it is also the only supported model. We would expect, then, that any reasonable algorithm for deduction would always find this model, and hence it can serve as a sanity-check of our algorithm. Indeed, when applying Algorithm 1 we find 100% success rate no matter what initial values are given to p, q and r and at all temperatures from around 0.7 to 0.1422:

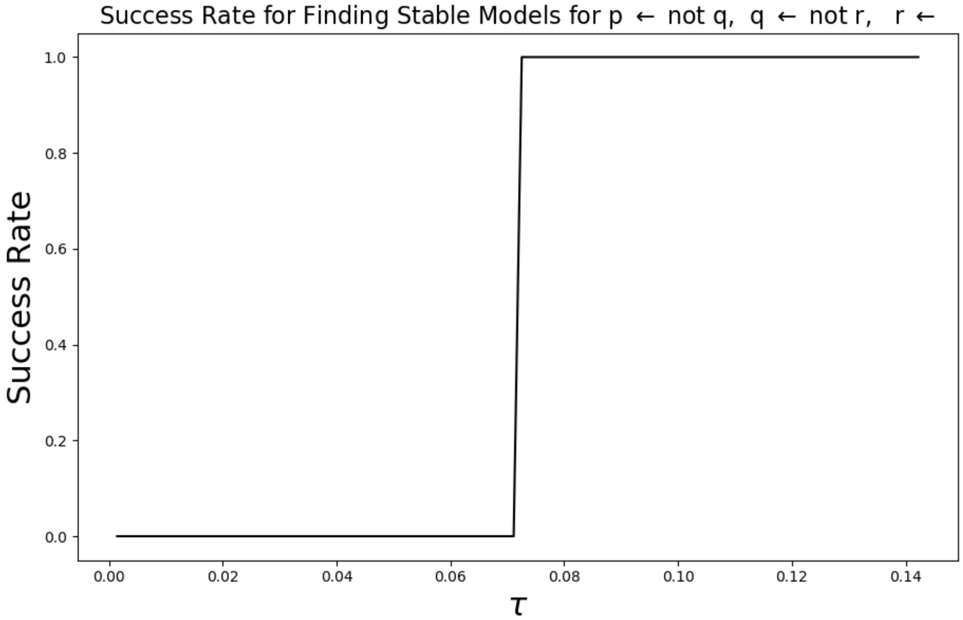


Figure 6.4: Success rate for finding a stable model, given a temperature. γ was fixed to a value of $\frac{1}{2}$. Temperatures were varied from 0.01422 to 0.1422.

For temperatures below 0.7 we again see a similar behaviour to case 1. Namely, the low temperatures lead to a Jacobian with very large determinant value, that fails to converge to a root. This shows the importance of picking temperature values carefully. In principle, the higher the temperature, the better the algorithm would perform. For completion, we show a convergence map for $\tau = 0.1422$. For all points the initial value for r was set to 0.

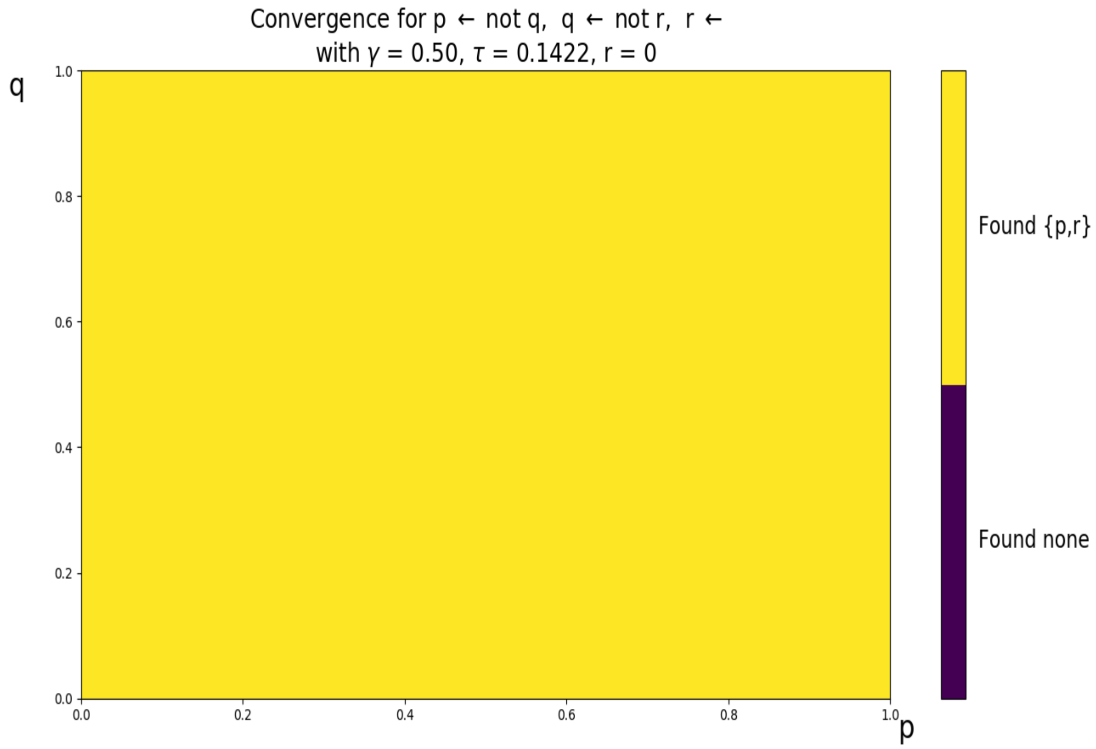


Figure 6.5: Convergence map for $\gamma = \frac{1}{2}$ and $\tau = 0.1422$. The p and q axis indicate their initial value in $\boldsymbol{v}^0$. The uniform colour indicates the algorithm always converged to the unique stable model.

6.3 Case Study 3

Lastly, consider the following program:

$$\begin{aligned} p &\leftarrow \text{not}\, q \\ q &\leftarrow \text{not}\, r \\ r &\leftarrow \text{not}\, s \\ s &\leftarrow \text{not}\, p \end{aligned} \tag{6.5}$$

Like the program in case study 1, this program is both non-monotonic and has multiple stable models. Namely, both $\{p, r\}$ and $\{q, s\}$ are stable models of the program. Because of this, a theorem-prover like Prolog would flounder if the truth-value of any atom is queried. Differently from case study 1, there is a dependency between two atoms in a model. Namely, it is not enough to deduce that p is in a stable model if r is not deduced at the same time. Similarly with q and s. To see how algorithm 1 performs in this case, we again test for the success rate of finding stable models:

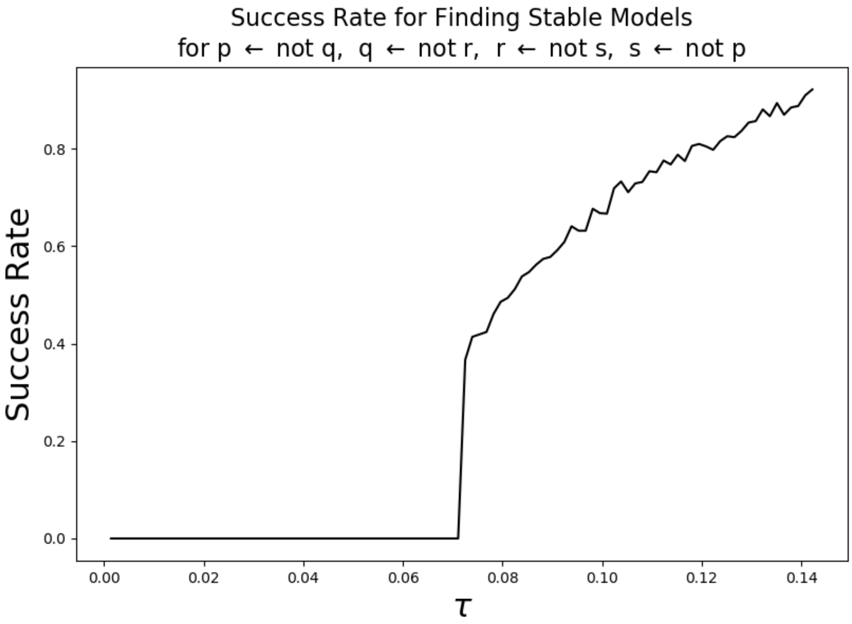


Figure 6.6: Success rate for finding a stable model, given a temperature. γ was fixed to a value of $\frac{1}{2}$. Temperatures were varied from 0.01422 to 0.1422.

We can see in this case very similar behaviour to 6.1. For low temperatures, up to about 0.07, we see no convergence to a stable model. At about $\tau = 0.07$ we see a sharp change, with the rate of convergence to a stable model rising steadily from about 50% up to more than 90% by the time we reach $\tau = 0.1422$. This is a somewhat better convergence rate than in case study 1.

To understand these results, we again turn to convergence maps. This time, however, a difficulty arises. Since there are four atoms of interest, the unit square we considered in case study 1 becomes a four-dimensional hypercube here. To achieve visualisation still, we resort to looking at section planes. We create a section plane by fixing the initial values of r and s and only allowing the initial values of p and q to vary. Once an initial point is chosen, the algorithm continues as usual, meaning the values of r and s can change. We then note the result. As before, we observed four possible outcomes:

1. The algorithm converged to $\{p, r\}$.
2. The algorithm converged to $\{q, s\}$.
3. The algorithm converged to a third root in which p, q, r, s are undecided.
4. The algorithm did not converge.

The convergence maps are listed in the following table. For this experiment we fixed the values of the parameters as $\gamma = \frac{1}{2}$ and $\tau = 0.1422$.

The maps clearly show how most points do end up converging to one stable model or the other. Further, we clearly see how the initial value of r and s affect the outcome of the algorithm. When r is initially set to 0, we converge to the model $\{q, s\}$ more often. This can be understood of being a consequence of the clause $q \leftarrow$ not r - r being assumed false 'encourages' q to be true. Similarly, when r is set to 1, q is 'encouraged' to be false and we converge to $\{p, r\}$ more often. The initial value of s also has effect on convergence, in that s being set to 1 allows for $\{q, s\}$ to be found more often, but the effect is smaller compared to r.

One may wonder about this lack of asymmetry between r and s, given the symmetry of the program. This is especially surprising when one considers that our program has no special model - the program is perfectly symmetrical with respect to its two stable models $\{p, r\}$ and $\{q, s\}$. Yet, if one one looks at the convergence maps above, some asymmetry can be clearly seen. The maps for $(r, s) = (0, 0)$ and $(r, s) = (1, 1)$ have somewhat similar structure but are clearly not a mirror image of each other. The same can be seen for $(r, s) = (1, 0)$ and $(r, s) = (0, 1)$.

In fact, the program does have one subtle asymmetry that arises when one fixes the initial values of r and s. This asymmetry can best be appreciated when looking at the program's dependency graph (Apt et al., 1988):

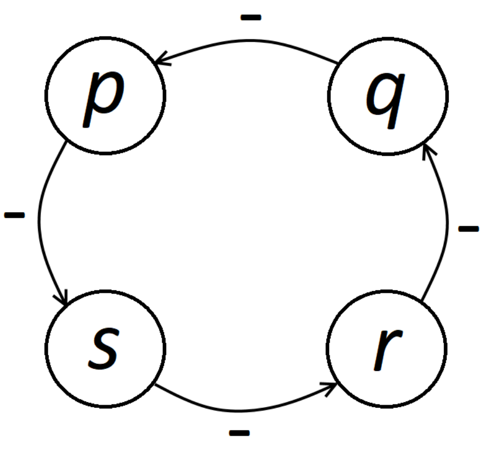
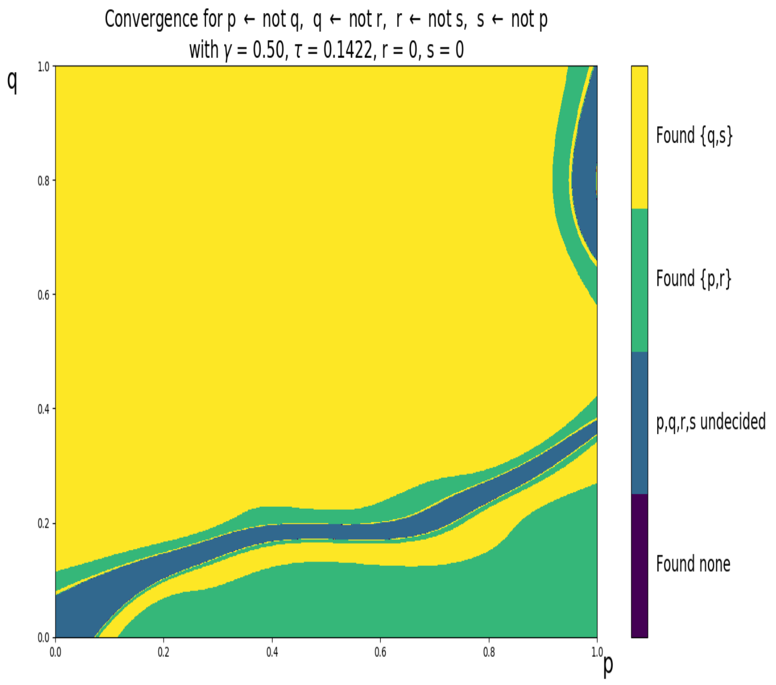
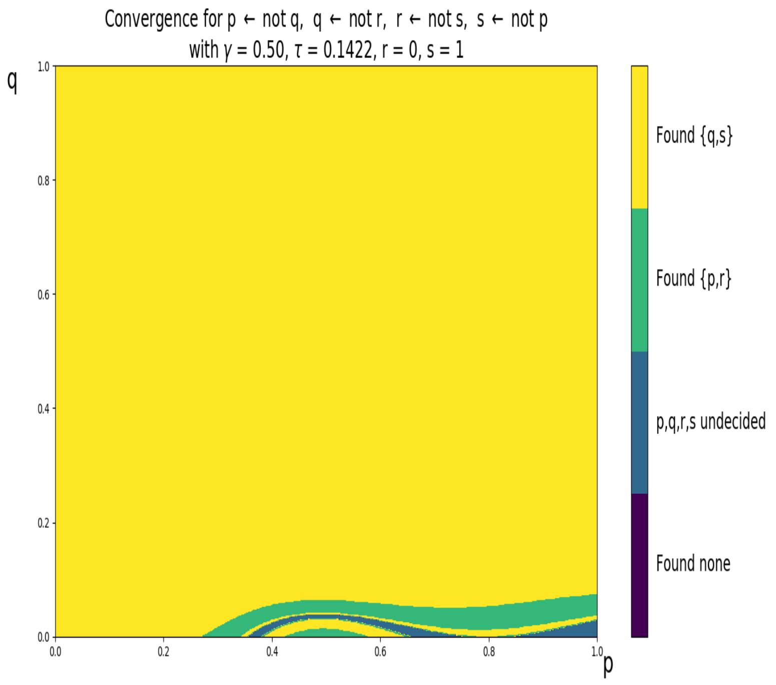


Figure 6.7: Dependency graph of the program. An edge between atoms marked with '-' indicates a negative dependency. Asymmetry can be seen in the direction of the edges.

The dependency graph can be read as follows: Each atom is a node in the graph. An edge from q to p marked with '-' means there is a clause in the program with p as its head and with q in the negative body (Apt et al., 1988). The asymmetry that we see in the convergence map stems from the direction of the edges. Namely, there is an edge from q to p but there is no edge from p to q. This means that if q is assumed to be false, p is an immediate consequence, but the other way around is not true. Instead, multiple steps are required to deduce p. This also explains why there is a stronger dependency on r's initial value than s - r affects q's truth value immediately while s requires two steps.

Convergence Map	r, s values
	$r = 0$ $s = 0$
	$r = 0$ $s = 1$

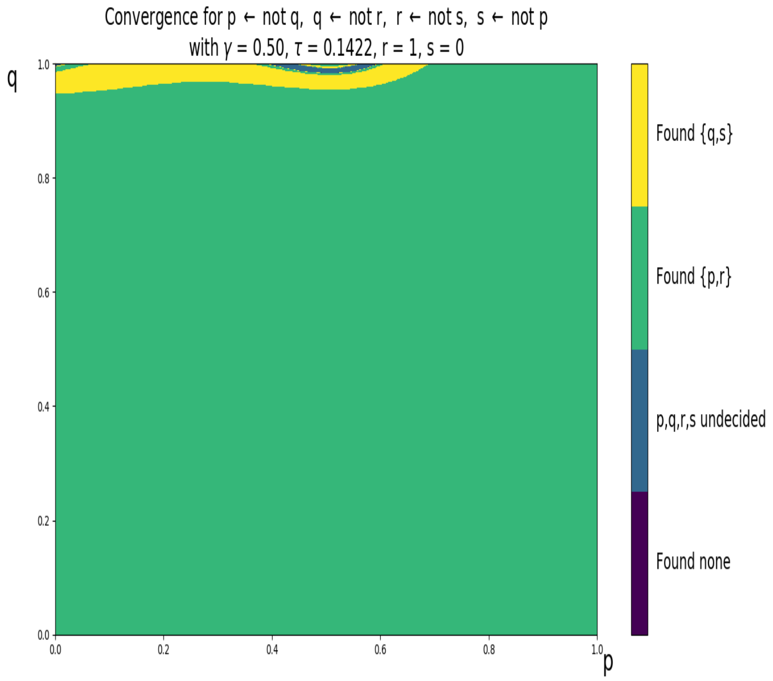
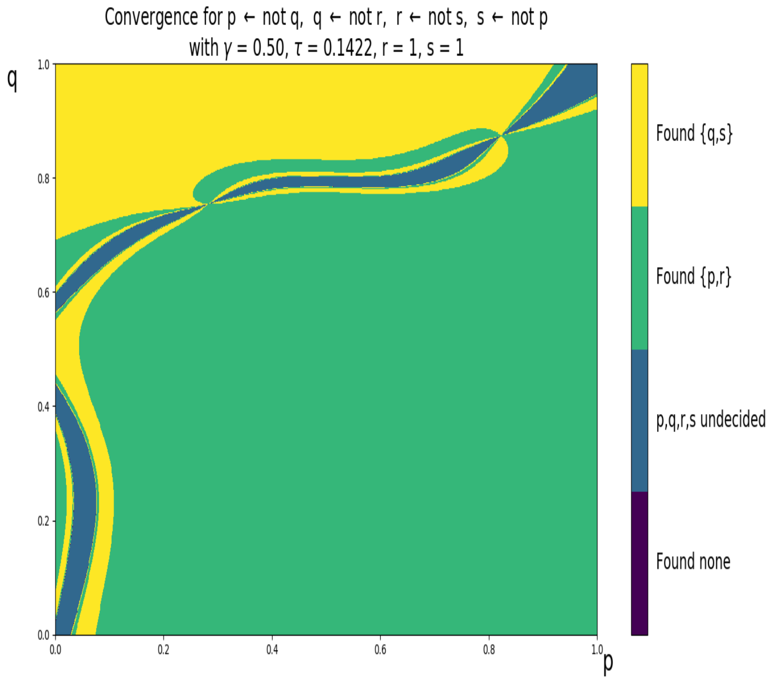
Convergence Map	r, s values
	$r = 1$ $s = 0$
	$r = 1$ $s = 1$

Table 6.2: Convergence maps for four different section planes.

Chapter 7

First-Order Deduction

In previous chapters we have shown how Horn and normal programs can be characterized using linear algebra in the propositional case. In realistic scenarios, logic programs are often written using First Order Predicate logic instead of propositional logic, which allows us to capture general rules using universally and existentially qualified variables.

In this chapter, we show one possible way to extend the linear algebraic approach from the propositional case to First Order. The ideas presented here have not yet been implemented and tested, and so are provided as a theoretical framework only, to be explored further in the future.

7.1 Grounding

Our linear algebraic description of Logic Programming can be extended from the propositional case to function-free First-Order Predicate Logic. The most simple method of extension is to simply ground the program. For instance, for the following program:

$$\begin{aligned} & p(X) \leftarrow q(X) \\ & p(a) \leftarrow \\ & q(b) \leftarrow \end{aligned} \tag{7.1}$$

We can assume our domain contains only constant symbols that appear in the program and produce the following grounding:

$$\begin{aligned} & p(a) \leftarrow q(a) \\ & p(b) \leftarrow q(b) \\ & p(a) \leftarrow \\ & q(b) \leftarrow \end{aligned} \tag{7.2}$$

The program is then transformed into the matrix:

$$\begin{array}{c|cccccc} & \bot & \top & p(a) & p(b) & q(a) & q(b) \\ \hline \bot & 0 & 0 & 0 & 0 & 0 & 0 \\ \top & 0 & 1 & 0 & 0 & 0 & 0 \\ p(a) & 0 & 1 & 0 & 0 & 1 & 0 \\ p(b) & 0 & 0 & 0 & 0 & 0 & 1 \\ q(a) & 0 & 0 & 0 & 0 & 0 & 0 \\ q(b) & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \tag{7.3}$$

To be even more optimal, we can produce only the so-called “relevant grounding” - only produce grounded symbols if they are the head of some rule. The advantage of this method is its simplicity.

The disadvantage is that it loses the structure of the original program. Predicate symbols and constants are no longer distinguished. Instead, they are merged into a new symbol. For instance, the symbols $p(a)$ and $q(a)$ are no longer related, even though they share the same constant a. Similarly, there is no relation between the symbols $p(a)$ and $p(b)$, despite sharing the same predicate symbol. If, for instance, we wish to differentiate a function with respect to the vector representation of a constant such as a (see, for instance, Rocktäschel and Riedel (2017)), we would not be able to do so since a is no longer a symbol in the program. We therefore need an approach that separates between the vector representation of predicates and constants.

Grefenstette (2013) proposed to embed the semantics of function-free and quantifier-free predicate calculus into High-Order tensors. Sato (2017a) extended the method to include quantifiers. Using this representation, the truth-value of a first order function-free formula can be evaluated under a given model. Unlike the linear algebraic approach of Sakama et al. (2017) and the one provided in this work, no method of computing a model of an entire program is given. We here build upon their previous work to construct high-order tensor representations of first order logic clauses, while providing a method to compute suitable models under such programs.

Suppose our program contains N predicate symbols $\{p_1 = \bot, p_2 = \top, p_3, ..., p_N$ and $L - 1$ entities $\{e_1, e_2, ..., e_{L-1}\}$. We introduce a new entity e_0, for a total of L, the purpose of which will be shown later.

7.2 The Unary Case

As an example, we begin by assuming our program only contains unary predicates. We embed predicate symbols into one-hot vectors in $\mathbb{R}^N$, and entity symbols into one-hot vectors in $\mathbb{R}^L$. So, for the above program, we have predicates $\{\bot, \top, p, q\}$, represented by:

$$\boldsymbol{v}^{\perp} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{\top} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{p} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \boldsymbol{v}^{q} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{7.4}$$

And entities $\{e_0, a, b\}$ represented by:

$$\boldsymbol{v}^{e_0} = \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \boldsymbol{v}^{a} = \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \boldsymbol{v}^{b} = \begin{bmatrix}0\\0\\1\end{bmatrix} \tag{7.5}$$

If we wish to embed an interpretation I such as $\{p(a), q(b)\}$, we can do so by taking the outer product of the predicate representation with the entity representation as in:

$$T^{I} = \boldsymbol{v}^{p} \circ \boldsymbol{v}^{a} + \boldsymbol{v}^{q} \circ \boldsymbol{v}^{b} = \boldsymbol{v}^{p} \cdot (\boldsymbol{v}^{a})^{\intercal} + \boldsymbol{v}^{q} \cdot (\boldsymbol{v}^{b})^{\intercal} \tag{7.6}$$

So now an interpretation is embedded as a matrix, rather than a vector as in the propositional case. This leads to a problem, however. Interpretations also contain the atom $\top$, but $\top$ has no matrix representation. It is only represented as the vector $\boldsymbol{v}^{\top}$, so it cannot be added into the interpretation as addition between vectors and matrices is not defined. To remedy this, we use the new entity e_0. e_0 intuitively represents "no entity" or "the null entity" and is simply a place-holder when there is no entity to insert. Using e_0, we can turn propositional atoms such as $\top$ into unary predicates represented by $\top(e_0)$. Our interpretation is now represented as the matrix:

$$T^{I} = \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} + \boldsymbol{v}^{p} \circ \boldsymbol{v}^{a} + \boldsymbol{v}^{q} \circ \boldsymbol{v}^{b} \tag{7.7}$$

Next we turn our attention to clauses. Let's consider the fact:

$$r: \qquad p(a) \leftarrow \tag{7.8}$$

In the propositional case, the head of a rule was represented by a vector $\boldsymbol{v}^{h}$. Here instead it is given by a matrix $T^{h} = \boldsymbol{v}^{p} \circ \boldsymbol{v}^{a}$. The body is also a matrix representing top: $T^{B} = \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0}$. In the propositional case, a clause was transformed into a matrix by taking the outer product of the head vector $\boldsymbol{v}^{h}$ with the body vector $\boldsymbol{v}^{B}$. Here, a clause is transformed into a fourth order tensor in the same manner:

$$D^{r} = T^{h} \circ T^{B} = (\boldsymbol{v}^{p} \circ \boldsymbol{v}^{a}) \circ (\boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0}) \tag{7.9}$$

Now, suppose we are given the empty interpretation $T^{\emptyset} = \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0}$. We apply the T_P operator as before, by taking the dot product between the tensor representing the body T^B and the interpretation T^I, and applying H_1 element-wise. The result is given by:

$$\begin{aligned} T^{T_P(\emptyset)} &= H_1\left[T^h \cdot \left\langle T^B, T^I \right\rangle\right] = H_1\left[(\boldsymbol{v}^p \circ \boldsymbol{v}^a) \cdot \left\langle \boldsymbol{v}^\top \circ \boldsymbol{v}^{e_0}, \boldsymbol{v}^\top \circ \boldsymbol{v}^{e_0} \right\rangle\right] = \\ &= H_1\left[(\boldsymbol{v}^p \circ \boldsymbol{v}^a) \cdot ((\boldsymbol{v}^\top)^\intercal \boldsymbol{v}^\top \cdot (\boldsymbol{v}^{e_0})^\intercal \boldsymbol{v}^{e_0})\right] = H_1\left[(\boldsymbol{v}^p \circ \boldsymbol{v}^a) \cdot (1 \cdot 1)\right] = \boldsymbol{v}^p \circ \boldsymbol{v}^a \end{aligned} \tag{7.10}$$

$T_P(\emptyset)$ therefore contains the atom $p(a)$. To maintain $\top$ in the interpretation, we must again add to our program the tautology '$\top \leftarrow \top, \text{not}\,\bot$' which is represented by $(\boldsymbol{v}^\top \circ \boldsymbol{v}^{e_0}) \circ (\boldsymbol{v}^\top \circ \boldsymbol{v}^{e_0})$.

Now that we can deal with facts, let's consider the clause:

$$r: \qquad p(X) \leftarrow q(X) \tag{7.11}$$

Because the variable X is universally quantified, the clause must be embedded into our 4-order tensor for all entities:

$$\begin{aligned} D^r &= \sum_{e_j} (\boldsymbol{v}^p \circ \boldsymbol{v}^{e_j}) \circ (\boldsymbol{v}^q \circ \boldsymbol{v}^{e_j}) = \\ &= \boldsymbol{v}^p \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^q \circ \boldsymbol{v}^{e_0} + \boldsymbol{v}^p \circ \boldsymbol{v}^a \circ \boldsymbol{v}^q \circ \boldsymbol{v}^a + \boldsymbol{v}^p \circ \boldsymbol{v}^b \circ \boldsymbol{v}^q \circ \boldsymbol{v}^b \end{aligned} \tag{7.12}$$

So if our interpretation is $\{p(a), q(b)\}$ and is therefore embedded as $T^I = \boldsymbol{v}^p \circ \boldsymbol{v}^a + \boldsymbol{v}^q \circ \boldsymbol{v}^b$, we can take the dot product to get:

$$\begin{aligned} &\boldsymbol{v}^p \circ \boldsymbol{v}^{e_0} \cdot \left\langle \boldsymbol{v}^q \circ \boldsymbol{v}^{e_0}, \boldsymbol{v}^p \circ \boldsymbol{v}^a + \boldsymbol{v}^q \circ \boldsymbol{v}^b \right\rangle + \\ &\qquad \boldsymbol{v}^p \circ \boldsymbol{v}^a \cdot \left\langle \boldsymbol{v}^q \circ \boldsymbol{v}^a, \boldsymbol{v}^p \circ \boldsymbol{v}^a + \boldsymbol{v}^q \circ \boldsymbol{v}^b \right\rangle + \\ &\qquad \boldsymbol{v}^p \circ \boldsymbol{v}^b \cdot \left\langle \boldsymbol{v}^q \circ \boldsymbol{v}^b, \boldsymbol{v}^p \circ \boldsymbol{v}^a + \boldsymbol{v}^q \circ \boldsymbol{v}^b \right\rangle = \boldsymbol{v}^p \circ \boldsymbol{v}^b \end{aligned} \tag{7.13}$$

And applying H_1 doesn't change the result, hence $T_P(I)$ contains $p(b)$. Now suppose we had a clause of the form:

$$r: \qquad p(X) \leftarrow q(Y) \tag{7.14}$$

The variable Y is understood as existentially quantified over the body. It is sufficient therefore that an interpretation contain any atom $q(e_{j_1})$ to deduce all atoms of the form $p(e_j)$. This is encoded into the tensor representation of the body by summing over all entities, allowing any of them to be true to deduce the head, as follows:

$$D^r = \sum_{e_j} \boldsymbol{v}^p \circ \boldsymbol{v}^{e_j} \circ \sum_{e_{j_1}} \boldsymbol{v}^q \circ \boldsymbol{v}^{e_{j_1}} = \sum_{e_j, e_{j_1}} \boldsymbol{v}^p \circ \boldsymbol{v}^{e_j} \circ \boldsymbol{v}^q \circ \boldsymbol{v}^{e_{j_1}} \tag{7.15}$$

Finally, a more general definite clause:

$$r: \qquad h(X) \leftarrow b_1(X), b_2(X), ..., b_n(X), b_{n+1}(X_1), ..., b_{n+m}(X_m) \tag{7.16}$$

Is embedded into 4-order tensor by:

$$D^r = \frac{1}{n+m} \sum_{e_j} \boldsymbol{v}^h \circ \boldsymbol{v}^{e_j} \circ \big(\sum_{i=1}^{n} \boldsymbol{v}^{b_i} \circ \boldsymbol{v}^{e_j} + \sum_{i=n+1}^{m} \sum_{e_{j_i}} \boldsymbol{v}^{b_i} \circ \boldsymbol{v}^{e_{j_i}} \big) \tag{7.17}$$

In this manner we can embed each clause in the program, one by one, into a 4-order tensor, and by starting with the empty interpretation $\boldsymbol{v}^\top \circ \boldsymbol{v}^{e_0}$, we can iteratively apply T_P, hence computing the Least Herbrand Model. Just as in the propositional case, our program must satisfy the Multiple Definitions condition: Every grounded atom must be defined by at most one long rule. For instance, in the following program:

$$\begin{aligned} p(a) &\leftarrow q(a), r(b) \\ p(X) &\leftarrow q(X), s(X) \end{aligned} \tag{7.18}$$

$p(a)$ is a symbol that is defined by multiple long clauses. $p(b)$, however, is not, since it is only the head of the clause $p(b) \leftarrow q(b), s(b)$. Note that having a predicate defined by only one ungrounded clauses does not guarantee that the MD condition is satisfied. For instance, if we replaced the ungrounded clause in the above program with:

$$p(X) \leftarrow q(X), s(Y)$$

$p(b)$ would be multiply defined since it is the head of two grounded clauses:

$$\begin{aligned} p(b) &\leftarrow q(b), s(a) \\ p(b) &\leftarrow q(b), s(b) \end{aligned} \tag{7.19}$$

To ensure the MD condition is satisfied, we can introduce auxiliary variables as before. For instance, we can replace the above grounded clauses with:

$$\begin{aligned} p(b)_1 &\leftarrow q(b), s(a) \\ p(b)_2 &\leftarrow q(b), s(b) \\ p(b) &\leftarrow p(b)_1 \\ p(b) &\leftarrow p(b)_2 \end{aligned} \tag{7.20}$$

7.3 The Binary Case

The method outlined above can naturally be extended to k-ary predicates. An interpretation I will be embedded into a $(k+1)$-order tensor. So a symbol $p(a_1, a_2, ..., a_k)$ becomes associated with the tensor $\boldsymbol{v}^p \circ \boldsymbol{v}^{a_1} \circ ... \circ \boldsymbol{v}^{a_k}$, and similarly to above, programs can be embedded as $(2k+2)$-order tensors. However, the memory requirement to store a tensor is exponentially dependent on its order, making this method intractable for large values of k. We therefore limit ourselves to the case of binary predicates.

As an example of the binary case, consider the following set of facts (example from Prolog Programming for Artificial Intelligence, chapter 1, Bratko (2001)):

$$\begin{aligned}
&\text{parent}(\text{pam}, \text{bob}) \leftarrow\\
&\text{parent}(\text{tom}, \text{bob}) \leftarrow\\
&\text{parent}(\text{tom}, \text{liz}) \leftarrow\\
&\text{parent}(\text{bob}, \text{ann}) \leftarrow\\
&\text{parent}(\text{bob}, \text{pat}) \leftarrow\\
&\text{parent}(\text{pat}, \text{jim}) \leftarrow
\end{aligned} \tag{7.21}$$

The facts are embedded into the tensor:

$$\begin{aligned}
T^P =&\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{\text{pam}} \circ \boldsymbol{v}^{\text{bob}} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0} +\\
&\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{\text{tom}} \circ \boldsymbol{v}^{\text{bob}} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0} +\\
&\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{\text{tom}} \circ \boldsymbol{v}^{\text{liz}} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0} +\\
&\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{\text{bob}} \circ \boldsymbol{v}^{\text{ann}} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0} +\\
&\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{\text{bob}} \circ \boldsymbol{v}^{\text{pat}} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0} +\\
&\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{\text{pat}} \circ \boldsymbol{v}^{\text{jim}} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0}
\end{aligned} \tag{7.22}$$

Unary facts need to be extended to binary, so:

$$\begin{aligned}
&\text{female}(\text{pam}) \leftarrow\\
&\text{male}(\text{tom}) \leftarrow\\
&\text{male}(\text{bob}) \leftarrow
\end{aligned} \tag{7.23}$$

$$\tag{7.24}$$

is embedded as:

$$\begin{aligned}
T^P =&\boldsymbol{v}^{\text{female}} \circ \boldsymbol{v}^{\text{pam}} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0} +\\
&\boldsymbol{v}^{\text{male}} \circ \boldsymbol{v}^{\text{tom}} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0} +\\
&\boldsymbol{v}^{\text{male}} \circ \boldsymbol{v}^{\text{bob}} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{\top} \circ \boldsymbol{v}^{e_0} \circ \boldsymbol{v}^{e_0}
\end{aligned} \tag{7.25}$$

A clause with universal quantifiers can be embedded by summing over entities:

$$\text{child}(X, Y) \leftarrow \text{parent}(Y, X) \tag{7.26}$$

$$T^{\text{child}} = \sum_{e_{j_1}, e_{j_2}} \boldsymbol{v}^{\text{child}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} \circ \boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_2}} \circ \boldsymbol{v}^{e_{j_1}} \tag{7.27}$$

And unary predicates are again extended:

$$\text{mother}(X, Y) \leftarrow \text{parent}(X, Y), \text{female}(X) \tag{7.28}$$

$$T^{\text{mother}} = \sum_{e_{j_1}, e_{j_2}} \boldsymbol{v}^{\text{mother}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} \circ \big(\frac{1}{2}\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} + \frac{1}{2}\boldsymbol{v}^{\text{female}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_0}\big) \tag{7.29}$$

Existential quantifiers are again introduced by summing over entities in the body:

$$\text{grandparent}(X, Y) \leftarrow \text{parent}(X, Z), \text{parent}(Z, Y) \tag{7.30}$$

$$T^{\text{grandparent}} = \sum_{e_{j_1}, e_{j_2}, e_{j_3}} \boldsymbol{v}^{\text{grandparent}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} \circ \big(\frac{1}{2}\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_3}} + \frac{1}{2}\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_3}} \circ \boldsymbol{v}^{e_{j_2}}\big) \tag{7.31}$$

One can also introduce conditions on the variables, such as inequality, by removing entities from the summation:

$$\text{sibling}(X, Y) \leftarrow \text{parent}(Z, X), \text{parent}(Z, Y), X \neq Y \tag{7.32}$$

$$T^{\text{sibling}} = \sum_{e_{j_1}, e_{j_2}, e_{j_3}, e_{j_1} \neq e_{j_2}} \boldsymbol{v}^{\text{sibling}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} \circ \big(\frac{1}{2}\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_3}} \circ \boldsymbol{v}^{e_{j_1}} + \frac{1}{2}\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_3}} \circ \boldsymbol{v}^{e_{j_2}}\big) \tag{7.33}$$

Finally, recursion can also be represented by a tensor:

$$\begin{aligned} &\text{ancestor}(X, Y) \leftarrow \text{parent}(X, Y) \\ &\text{ancestor}(X, Y) \leftarrow \text{parent}(X, Z), \text{ancestor}(Z, Y) \end{aligned} \tag{7.34}$$

$$\begin{aligned} T^{\text{ancestor}} = &\sum_{e_{j_1}, e_{j_2}} \boldsymbol{v}^{\text{ancestor}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} \circ \boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} + \\ &\sum_{e_{j_1}, e_{j_2}, e_{j_3}} \boldsymbol{v}^{\text{ancestor}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_2}} \circ \big(\frac{1}{2}\boldsymbol{v}^{\text{parent}} \circ \boldsymbol{v}^{e_{j_1}} \circ \boldsymbol{v}^{e_{j_3}} + \frac{1}{2}\boldsymbol{v}^{\text{ancestor}} \circ \boldsymbol{v}^{e_{j_3}} \circ \boldsymbol{v}^{e_{j_2}}\big) \end{aligned} \tag{7.35}$$

7.4 The Non-Monotonic Case

If a First Order program contains normal rules, it is also possible to construct the reduct of a program much in the same way as in the propositional case. Suppose we are given a rule such as:

$$r: \qquad h(X,Y) \leftarrow p(X,Y), \text{not } q(X,Y) \tag{7.36}$$

The corresponding definite rule has a 6-order tensor representation:

$$\sum_{e_{j_1}, e_{j2}} \boldsymbol{v}^h \circ \boldsymbol{v}^{e_{j1}} \circ \boldsymbol{v}^{e_{j2}} \circ \boldsymbol{v}^p \circ \boldsymbol{v}^{e_{j1}} \circ \boldsymbol{v}^{e_{j2}} \tag{7.37}$$

The negative body $q(X,Y)$ has a third order tensor representation of its truth value:

$$T^q = \sum_{e_{j_1}, e_{j2}} \boldsymbol{v}^q \circ \boldsymbol{v}^{e_{j1}} \circ \boldsymbol{v}^{e_{j2}} \tag{7.38}$$

If a candidate stable model M is given, with tensor representation T^M, the reduct rule is embedded as:

$$T^r = \sum_{e_{j_1}, e_{j2}} \left\langle 1 - T^M, \boldsymbol{v}^q \circ \boldsymbol{v}^{e_{j1}} \circ \boldsymbol{v}^{e_{j2}} \right\rangle \cdot \boldsymbol{v}^h \circ \boldsymbol{v}^{e_{j1}} \circ \boldsymbol{v}^{e_{j2}} \circ \boldsymbol{v}^p \circ \boldsymbol{v}^{e_{j1}} \circ \boldsymbol{v}^{e_{j2}} \tag{7.39}$$

7.5 Discussion

In this chapter we provided a tensor-based method for representing First-Order logic programs. Our method is based on a grounding of a program that retains the structure of the original atoms by separating predicate and entity vector representations.

Grounding the program has the advantage of providing a natural way of expressing a wide variety of clauses, allowing universally and existentially quantified variables, as well as special conditions such as inequality. This freedom of expression comes with a disadvantage, however, as grounded programs potentially generates very large programs.

If a program, satisfying the MD condition, contains N predicate symbols and L entity symbols, the space requirement in the binary case is $\mathcal{O}(N^2L^4)$. If it contains R grounded clauses, constructing the tensor has a worst-case time complexity of $\mathcal{O}(R \cdot N \cdot L^2)$. Computing one-step deduction would take $\mathcal{O}(N^2L^4)$ (assuming a non-sparse representation of the tensor) and the Least Herbrand Model of a Horn program would then take $\mathcal{O}(N^3L^6)$.

To improve scalability, future implementations could use sparse tensor representations. Methods to provide grounding of clauses on a need-only basis may also be developed.

Chapter 8

Related Work

There are not many works that attempt to realize logic programming using linear algebraic methods. Lin (2013) gives a linear algebraic treatment to SAT solving. There are many works concerning relational learning by embedding entities of KBs in a vector space, such as Socher et al. (2013), Yang et al. (2014), Rocktäschel and Riedel (2017). Serafini and Garcez (2016) introduce Real Logic as a formulation of logical inference in a continuous space which can be realized by means of Logic Tensor Networks. Cohen (2016) introduces TensorLog as a differentiable framework for probabilistic logic. None of these are concerned with a linear algebraic treatment of logic programming for exact inference.

In this chapter we provide a summary of those works most relevant to our own. Namely, those works concerned with providing a linear algebraic theory for inference from logic programs.

8.1 "Linear Algebraic Characterization of Logic Programs"

The original work by Sakama et al. (2017) regarding a matrix representation of a Horn program provides the basis of our work. In their paper, Sakama et al. also treat the case of disjunctive and normal programs. While in our work we chose not to treat the case of disjunctive programs, in Sakama et al. (2017) these are reduced into the Horn case by means of *split programs*. Each disjunctive clause of the form:

$$h_1 \vee h_2 \vee ...h_m \leftarrow b_1, b_2, ..., b_n \tag{8.1}$$

is split into m Horn clauses:

$$\begin{aligned} h_1 \leftarrow& b_1, b_2, ..., b_n \\ h_2 \leftarrow& b_1, b_2, ..., b_n \\ &\vdots \\ h_m \leftarrow& b_1, b_2, ..., b_n \end{aligned} \tag{8.2}$$

Split programs are then created by incorporating one split Horn clause for each of the original disjunctive clause. Since each split program is Horn, it can then be transformed into a matrix, and these matrices are then stacked as the frontal slices of a third-order tensor. Computing minimal models is then simply the matter of computing the Least Herbrand Model for each frontal slice, as described in section 2.5.

Normal programs are also treated, but unlike in our work, reducts are not considered. Instead, these are transformed into disjunctive programs by replacing a negative body atom such as 'not c' with an auxiliary head atom ϵc, adding a clause of the form $\epsilon c \leftarrow c$ and requiring that if ϵc is in a model then c is as well. The minimal models of the resulting disjunctive program can be shown to be the stable models of the original.

While the third-order construction provides a systematic method to find all the stable models of a program, it suffers from a scalability problem, since in general a disjunctive program has an exponential number of split programs, and storing and manipulating the resulting tensor may not be achievable.

8.2 "Partial Evaluation of Logic Programs in Vector Spaces"

Sakama et al. (2018) build upon their previous and suggest several optimization for matrix-based deduction. First, $\bot$ and $\top$ are removed from the Herbrand base to reduce the matrix size. The removal of $\bot$ means the programs considered can no longer have constraints (definite programs). $\top$ can be removed by adding the facts of the program to the initial interpretation (instead of starting from the empty interpretation), and for each fact $p_i \leftarrow$ add instead the tautology $p_i \leftarrow p_i$ so that facts are not forgotten.

The next suggested improvement is by a method the authors term *column reduction*. The program is split into two: A *Singly Defined* program where every atom has a single definition, and a *Multiply Defined* program where atoms are the heads of multiple clauses. Clauses that multiply define an atom have their heads replaced by auxiliary variables, and a new rule is added to define the original atom, whose body is a disjunction of those new variables. Hence we have removed multiple definitions of the program at the cost of adding rules with disjunction in the body. Evaluation of T_P by matrix multiplication can be done as usual for the Horn clauses of the program, but columns corresponding to auxiliary variables can be removed, since these only appear as the heads of rules. Evaluation of the auxiliary rules is then done at the activation (non-linear) stage instead of the matrix multiplication stage. This results in less steps when computing T_P.

In addition to the optimisation above, the authors also consider how to perform partial evaluation of the program (Lloyd and Shepherdson, 1991) using linear algebra. They show that powers of the program matrix $\boldsymbol{D}^P$ capture the concept of partial evaluation when P is singly defined. This suggests partial evaluation can be used as a preprocessing step before computing the Least Herbrand Model in full.

It is possible that some of the techinques outlined in Sakama et al. (2018) would be beneficial for our work as well. We leave the study and evaluation of these techniques to future work.

8.3 "Towards a Formal Distributional Semantics: Simulating Logical Calculi with Tensors"

Grefenstette (2013) described a tensor-based framework for quantifier-free predicate logic, as a first step towards a linear algebraic characterization of formal semantics in natural language. The author associated the symbols $\top$ and $\bot$ with the vectors $[1, 0]^\intercal$ and $[0, 1]^\intercal$. If the Herbrand domain has L entities, those were embedded using one-hot encoding in $\mathbb{R}^L$. If p is a k-ary predicate, its truth value under model M is embedded in the tensor:

$$T^p = \sum_{\boldsymbol{e}^1_{i_1}, \boldsymbol{e}^2_{i_2}, ..., \boldsymbol{e}^L_{i_1}} c^M_{1 i_1 i_2 ... i_L} \top \circ \boldsymbol{e}^1_{i_1} \circ \boldsymbol{e}^2_{i_2} \circ ... \circ \boldsymbol{e}^L_{i_L} + \sum_{\boldsymbol{e}^1_{i_1}, \boldsymbol{e}^2_{i_2}, ..., \boldsymbol{e}^L_{i_L}} c^M_{2 i_1 i_2 ... i_L} \bot \circ \boldsymbol{e}^1_{i_1} \circ \boldsymbol{e}^2_{i_2} \circ ... \circ \boldsymbol{e}^L_{i_L} \tag{8.3}$$

where:

$$c^M_{1 i_1 i_2 ... i_L} = \begin{cases} 1 & p(e^1_{i_1}, e^2_{i_2}, ..., e^L_{i_L}) \in M \\ 0 & p(e^1_{i_1}, e^2_{i_2}, ..., e^L_{i_L}) \notin M \end{cases} \qquad c^M_{2 i_1 i_2 ... i_L} = 1 - c^M_{1 i_1 i_2 ... i_L} \tag{8.4}$$

The author also shows how the semantics of propositional logical connectors can be embedded in second or third order tensors. Finally, the author provides a proof that the semantics of universal and existential quantifiers cannot be captured using linear mappings alone.

In chapter 7 we drew on the ideas from Grefenstette (2013) to provide a tensor-based framework for First-Order logic programs. However, our method differs in several key points. First, we do not embed $\bot$ and $\top$ in their own vector space but rather treat them as 0-ary predicate symbols. In equation 8.3, rather than summing only for $\bot$ and $\top$, we sum for all predicate symbols in the program. This allows us to capture the truth value of all predicate symbols in the program, rather than a single one (see for example equation 7.7). In addition, we provide a general method for computing the truth value of predicates without being given a model in advance, by embedding clauses in a high-order tensor and allowing models to be computed iteratively.

8.4 "Embedding Tarskian Semantics in Vector Spaces"

Sato (2017a) extends the work of Grefenstette (2013) to include first-order logical formula with quantifiers. The author removes the vector representation of $\bot$ and $\top$

but does not provide a vector representation of predicates, so each predicate must be embedded in a separate tensor. Existential quantifiers are captured by summing the truth value of a predicate over all entities of the domain, and then applying a non-linear function $\text{min}_1(x) = \min(x, 1)$. A tensor representation of this operation is also given. This is quite similar to our approach for universally quantified variables, where the tensor representation of grounded clauses are summed over, and a non-linear H_1 operation is applied. However, by capturing clauses in this way, we do not need to be given a model of the program in advance to compute truth values of predicates.

While our method allows for compuation of models for any given program, the methods outlined by Sato (2017a) may be more efficient for certain classes of programs. For instance, in the special case of binary predicates, the author shows how the summation process can often be truncated by algebraic manipulation. Specifically, the author shows how the recursive relation:

$$\begin{aligned} r_2(X, Z) &\leftarrow r_1(X, Z) \\ r_2(X, Z) &\leftarrow r_1(X, Y), r_2(Y, Z) \end{aligned} \tag{8.5}$$

can be captured by the matrix equation $R_2 = \text{min}_1(R_1 + R_1 R_2)$. By replacing min_1 with multiplication by a small positive constant ϵ, the equation can be solved with matrix inversion and thresholding, achieving more efficient compuation in the case that R_1 is not a sparse matrix. More details are given by the author in Sato (2017b), where the matrix equation method is expanded to several classes of programs, but is still infeasible in the general case. For instance, for the program:

$$\begin{aligned} r_2(X, Z) &\leftarrow r_1(X, Z) \\ r_2(X, Z) &\leftarrow r_2(X, Y), r_2(Y, Z) \end{aligned} \tag{8.6}$$

the corresponding equation is quadratic in R_2. Negation in the case of non-stratified programs is also not treated.

8.5 “Abducing Relations in Continuous Spaces”

Sato et al. (2018) build upon their previous work (Sato, 2017a) to realize abductive inference in vector spaces. Specifically, binary datalog programs are considered for the purpose of abduction in knowledge graphs. Since a clause such as:

$$r_3(X, Y) \leftarrow r_1(X, Y), r_2(X, Y) \tag{8.7}$$

can be shown (Sato, 2017a) to be captured by the matrix equation $R_3(X, Y) = \text{min}_1(R_1 R_2)$, if we are given the matrices R_1, R_3, the abductive task becomes solving for R_2. Since an exact solution is intractable for large graphs, the equation is instead solved approximately ($R_3 \approx \text{min}_1(R_1 R_2)$) by solving $R_3 = R_1 X$ for X and then thresholding $R_2 = X_{>\theta}$ for an appropriately selected θ such that if $X_{ij} > \theta$ then $R_{2ij} = 1$, otherwise $R_{2ij} = 0$. A treatment of the recursive case from equation 8.5 is also given. Experimental results show that despite the approximation of the algorithm, exact or near-exact abductive inference can be achieved for fairly large programs.

8.6 "Tensor-Based Abduction in Horn Propositional Programs"

Aspis et al. (2018) extend the work of Sakama et al. (2017) to abductive inference for Horn programs. Inverted rules of the program are encoded into the frontal slices of a third-order *abductive tensor* A^P. So if the clause $\langle h^r, B^r, \emptyset \rangle$ has a matrix representation of $\frac{1}{n^r}\boldsymbol{v}^{h^r}(\boldsymbol{v}^{B^r})^\intercal$, the tensor would have a frontal slice of $A^P_{::r} = \boldsymbol{v}^{B^r}(\boldsymbol{v}^{h^r})^\intercal + \mathbb{I} - \boldsymbol{v}^{h^r}(\boldsymbol{v}^{h^r})^\intercal$. By multiplying A^P with an interpretation that contains h^r, the resulting interpretation contains B^r due to $\boldsymbol{v}^{B^r}(\boldsymbol{v}^{h^r})^\intercal$, does not contain h^r due to $-\boldsymbol{v}^{h^r}(\boldsymbol{v}^{h^r})^\intercal$, and leaves all other atoms intact due to $\mathbb{I}$. Hence the atoms of B^r can be thought to have been "abduced" from the clause. Since all clauses of the program are encoded in A^P, all possible abductive solutions can be found in this way. Integrity constraint are checked by computing the fixed point of each potential solution, similarly to the process described in section 2.5. A method for filtering solutions according to a set of abducible atoms is also offered.

It is worth noting that this method for abduction differs from the one outlined by Sato et al. (2018) in that it searches for all abductive solutions, given an observation, instead of abducing the truth value of specific predicates. It is, however, only propositional at this time. It may be possible to combine this method with the one in Sato et al. (2018) or chapter 7 of this work to realize full abduction in the first order case, but this is left for future work. Unlike the approach taken by Sato et al. (2018), such an extension would allow exact abductive inference. It would also search for all abductive solutions of the program, rather than considering one clause at a time.

Chapter 9

Extensions

In this report we provided a detailed description of how logic programs could be characterized using linear algebraic methods to realize exact inference. Our long-term goal involves finding efficient inference algorithms for web-scale knowledge bases. Such programs pose two main challenges: First, they often contain millions of distinct symbols, rendering inference tasks such as computing their Least Herbrand Model intractable. Second, they are highly dynamic, with new atoms and clauses added daily. To deal with these challenges, we advocate replacing exact inference with approximations using neural-symbolic integration approaches (see Besold et al. (2017) for a survey). More specifically, we propose to use Deep Learning techniques to learn sub-symbolic representation of atoms to achieve approximate inference.

In this chapter we outline initial ideas towards that goal. First, we examine the case where our Herbrand Base is very large but with a known, fixed size. We describe a method of learning an embedding function that can help speed up queries to such a database. Second, we consider the case where the knowledge base has an unknown size such that a one-hot encoding of atoms is no longer possible. We develop mathematical formulas for taming such large knowledge bases by usage of Kernel methods.

9.1 Program Embedding

Suppose a Horn program P represents a fairly large, but fixed size, knowledge base. As always, we denote the size of the Herbrand base of P as N. We assume N is not too large so that the task of computing $T_P(I)$ for an arbitrary interpretation I is still tractable. However, since computing the Least Herbrand Model is of the order of $\mathcal{O}(N^3)$ operations, full inference may be unachievable directly.

We propose to embed $\boldsymbol{D}^P$ in a low-dimensional vector space. This embedding should still allow us to maintain our deductive capabilities, approximately. Let us denote the embedding function as $\phi_\theta : \{0,1\}^N \to \mathbb{R}^k$ where we assume N is much larger than k ($N >> k$), and θ are parameters of ϕ. Given an atom $p \in B_P$, its one-hot encoding is denoted $\boldsymbol{v}^p$ and its low-dimensional embedding is given by:

$$\boldsymbol{u}^p = \phi_\theta(\boldsymbol{v}^p) \tag{9.1}$$

And given a set of atoms $A \subseteq B_P$ we have:

$$\boldsymbol{u}^A = \phi_\theta(\boldsymbol{v}^A) \tag{9.2}$$

This suggests that, once the parameters θ of ϕ_θ are fixed, we can compute the low-dimensional embedding of all the atoms in B_P and store the result.

The embedded program matrix $\boldsymbol{E}^P_\theta$ is related to the original matrix $\boldsymbol{D}^P$ by:

$$\boldsymbol{E}^P_\theta = \sum_{r_i} \frac{1}{n^i} \boldsymbol{u}^{h^i} (\boldsymbol{u}^{B^i})^\intercal = \sum_{r_i} \frac{1}{n^i} \phi_\theta(\boldsymbol{v}^{h^i}) \phi_\theta(\boldsymbol{v}^{B^i})^\intercal \tag{9.3}$$

In the high-dimensional space, the immediate consequence of an interpretation I was given by:

$$\sigma_{\gamma,t}(\boldsymbol{D}^P \boldsymbol{v}^I) = \sigma_{\gamma,t}\big(\sum_{p \in B_P} \boldsymbol{v}^p (\boldsymbol{v}^p)^\intercal \boldsymbol{D}^P \boldsymbol{v}^I \big) = \sum_{p \in B_P} \sigma_{\gamma,t}((\boldsymbol{v}^p)^\intercal \boldsymbol{D}^P \boldsymbol{v}^I) \boldsymbol{v}^p \tag{9.4}$$

We would like then the following approximation to be true:

$$\sigma_{\gamma,t}\left((\boldsymbol{v}^p)^\intercal \boldsymbol{D}^P \boldsymbol{v}^I \right) \approx \sigma_{\gamma,t}\left((\boldsymbol{u}^p)^\intercal \boldsymbol{E}^P_\theta \boldsymbol{u}^I \right) = \sigma_{\gamma,t}\left(\phi_\theta(\boldsymbol{v}^p)^\intercal \boldsymbol{E}^P_\theta \phi_\theta(\boldsymbol{v}^I) \right) \tag{9.5}$$

One-step deduction can therefore be done as follows: Compute the embedding of I ($\mathcal{O}(N \cdot k)$ operations). Multiply $\boldsymbol{u}^I$ by the low-dimensional matrix $\boldsymbol{E}^P$ ($\mathcal{O}(k^2)$ operations). For every atom $p \in B_P$, to decide if $p \in T_P(I)$, perform the projection $(\boldsymbol{u}^p)^\intercal \boldsymbol{E}^P \boldsymbol{u}^I$ ($\mathcal{O}(k)$). We then apply $\sigma_{\gamma,t}$ and test whether the result is bigger than γ, and if it is, then $p \in T_P(I)$. The entire deductive step therefore takes $\mathcal{O}(N \cdot k)$ operations, instead of the $O(N^2)$ it would have ordinarily taken. Computing the Least Herbrand Model requires chaining one-step deductions until a fixed point is reached, and would take $\mathcal{O}(N^2 k)$ operations in the worst case, instead of $\mathcal{O}(N^3)$. To find the embedding function ϕ_θ, we define the loss function:

$$\mathcal{L}(\theta, P) = \frac{1}{2} \mathop{\mathbb{E}}_{\substack{p \sim U(B_P) \\ I \sim U(2^{B_P})}} \left(\sigma_{\gamma,t}\left((\boldsymbol{v}^p)^\intercal \boldsymbol{D}^P \boldsymbol{v}^I \right) - \sigma_{\gamma,t}\left(\phi_\theta(\boldsymbol{v}^p)^\intercal \boldsymbol{E}^P_\theta \phi_\theta(\boldsymbol{v}^I) \right) \right)^2 + \frac{1}{2} \lambda ||\theta||^2 \tag{9.6}$$

where $U(B_P)$ is a uniform distribution over B_P and $U(2^{B_P})$ is a uniform distribution over subsets of B_P. Computing the immediate consequence in the low-dimensional space achieves the best approximate inference when $\mathcal{L}(\theta, P)$ is minimized with respect to θ. This can be done, for instance, with stochastic gradient descent (Bottou, 1998).

9.2 Kernel Methods

Suppose now that N, the size of the Herbrand Base, is of such large order that even computing $T_P(I)$ is intractable, or that N may be constantly changing. Mathematically, we can capture such a situation by letting $N \to \infty$. For an arbitrary atom B_P, we have $\boldsymbol{v}^p \in \mathbb{R}^\infty$ and similarly $\boldsymbol{D}^P$ is of infinite size, and hence a direct linear algebraic representation of atoms and programs is not computable. To get around this problem, we notice that:

$$p \in T_P(I) \quad \Leftrightarrow \quad (\boldsymbol{v}^p)^\intercal \sigma_{\gamma,\tau}\left(\boldsymbol{D}^P \boldsymbol{v}^I\right) > \gamma \quad \Leftrightarrow \quad \sigma_{\gamma,\tau}\left((\boldsymbol{v}^p)^\intercal \boldsymbol{D}^P \boldsymbol{v}^I\right) > \gamma \tag{9.7}$$

Now, since:

$$\boldsymbol{D}^P = \sum_{r \in P} \frac{1}{n^r} \boldsymbol{v}^{h^r} (\boldsymbol{v}^{B^r})^\intercal \tag{9.8}$$

and:

$$\boldsymbol{v}^I = \sum_{q \in I} \boldsymbol{v}^q \qquad \boldsymbol{v}^{B^R} = \sum_{b \in B^r} \boldsymbol{v}^b \tag{9.9}$$

we have:

$$p \in T_P(I) \quad \Leftrightarrow \quad \sigma_{\gamma,\tau}\left(\sum_{r \in P} \sum_{b \in B^r} \sum_{q \in I} \frac{1}{n^r} (\boldsymbol{v}^p)^\intercal \boldsymbol{v}^{h^r} (\boldsymbol{v}^b)^\intercal \boldsymbol{v}^q \right) > \gamma \tag{9.10}$$

This suggests that if we knew how to compute the inner product of two vectors, even if those vectors are of infinite size, we can still perform deduction in a finite number of steps. In statistical machine learning this is known as the *Kernel method*. For Support Vector Machine (SVM) models, the Kernel method is used when examples are not separable by a linear hyperplane (Schölkopf et al., 2002). The examples are then mapped to high-dimensional or infinite vector space where they are separable, by using an implicit non-linear function. The computation remains finite by specifying a Kernel function that effectively computes the dot product between vectors in the high dimensional vector space.

Drawing on this idea, if we had in our possession a Kernel function $K_P(\cdot,\cdot)$ that, given two symbols p and q that are part of program P, can compute:

$$K_P(p,q) = (\boldsymbol{v}^p)^\intercal \boldsymbol{v}^q \tag{9.11}$$

then equation 9.10 becomes:

$$p \in T_P(I) \quad \Leftrightarrow \quad \sigma_{\gamma,\tau}\left(\sum_{r\in P}\sum_{b\in B^r}\sum_{q\in I}\frac{1}{n^r}K_P(p,h^r)K_P(b,q)\right) > \gamma \tag{9.12}$$

This poses the challenge of how to find such a function. One commonly used Kernel function is the Radial Basis Function (RBF) (Shalev-Shwartz and Ben-David, 2014). Denote the word embedding of an atom p as θ_p. Then the Gaussian RBF kernel is defined as:

$$K_\sigma(p,q) = e^{-\frac{|\theta_p-\theta_q|^2}{2\sigma^2}} \tag{9.13}$$

which suggests a new way of learning vector representations of atoms of a program.

As a final note, suppose we are in possession of a Kernel function $K_P(\cdot,\cdot)$ that perfectly computes the dot product for program P. Define:

$$Con_P(p,I) = (\boldsymbol{v}^p)^\intercal \boldsymbol{v}^I = \sum_{q\in I} K_P(p,q) \tag{9.14}$$

$Con_P(p,I)$ can be thought of as an oracle function that answers whether p is in an interpretation I when the Kernel function $K_P(p,q)$ has been learned for program P. If M is a supported model of P, then $Con_P(p,M)$ becomes an oracle function to test if p is a consequence of the program P. Note that since $M = T_P(M)$ we have:

$$(\boldsymbol{v}^p)^\intercal \boldsymbol{v}^M = (\boldsymbol{v}^p)^\intercal \boldsymbol{v}^{T_P(M)} = \sigma_{\gamma,\tau}\left(\sum_{r\in P}\sum_{b\in B^r}\sum_{q\in M}\frac{1}{n^r}K_P(p,h^r)K_P(b,q)\right) \tag{9.15}$$

which implies the following recursive relation holds:

$$Con_P(p,M) = \sigma_{\gamma,\tau}\left(\sum_{r\in P}\sum_{b\in B^r}\frac{K_P(p,h^r)}{n^r}Con_P(b,M)\right) \tag{9.16}$$

Hence the problem of searching for a supported model of a web-scale program can be replaced by a learning task for $Con_P(p,M)$. Equation 9.16 may be helpful in designing a loss function for said task.

Chapter 10

Conclusion

This work advocated for a linear algebraic approach to logic programming. Several theoretical results have been developed towards that goal. We began by providing necessary background information to the reader about logic programming, high-order tensors and previous work on the subject by other authors. We then posed a restriction on our program in the form of a Multiple Definitions condition, to ensure the correctness of some of the algorithms in this work. We proved this restriction is without loss of generality.

We then showed how the linear algebraic algorithm described by Sakama et al. (2017) can be altered to allow for a fully differentiable deductive process. Differentiable deduction depends on the selection of a threshold (γ) and temperature (τ) parameters, and we developed conditions on these parameters to ensure the correctness of exact inference. We then showed how using differentiable deduction we can search for supported models of a program by employing the Newton-Raphson method for root search.

Next, we introduced a completely linear algebraic characterization of the concept of program reduct for stable semantics. After proving the correctness of the characterization, we described how differentiable deduction can still be maintained in this non-monotonic case by developing further conditions on γ and τ. We then described how an algorithm based on the Newton-Raphson method can be applied in this case as well.

We then studied how the algorithm works in practice for several example programs. We noticed that for high temperature values the algorithm converges at impressive rates (including 100% convergence for the stratified program tested) which suggests it can be useful for directed search of the space of possible models of a program. We provided several illustrations of convergence maps to help develop intuition for the algorithm's behaviour.

We then introduced a new linear algebraic characterization of function-free First-Order logic programs. We showed how embedding a program into a high-order tensor can allow for grounding while maintaining syntactic structure of atoms. Several examples were given for the unary and binary case, though the case of general k-ary predicates remains intractable. Non-monotonic programs were also discussed.

Finally, we surveyed some of the most relevant work that has been published on the subject of combining logic programming with linear algebra, and outlined initial ideas on how this approach can be scaled to large knowledge-bases, either by learning low-dimensional embeddings for interpretations, or by learning a kernel function that measures the similarity between two symbols.

10.1 Future Work

Our work here provided a theoretical framework for linear algebraic algorithms for deduction from logic programs. Much work remains to done.

Algorithm 1 has been implemented for testing on small programs. An optimized implementation running on GPU hardware could allow for efficient scalability testing, and to provide further intuition when the algorithm is applied to more complex programs. Using sparse tensor representations could be especially beneficial to improve performance.

The First-Order case has only received theoretical treatment here. Future work will be directed at providing a full proof for the approach and implementation as a proof of concept. It may be possible to apply algorithm 1 in this case as well. It is unknown to us if a version of the Newton-Raphson algorithm where the variable is a high-order tensor exists in the literature. However, flattening the tensor into a vector may be a way to solve this difficulty.

Abduction as has been described by Aspis et al. (2018) may also be extended from the propositional case to First-Order, by combining some of the ideas from chapter 7. A full first-order abductive process has been used in several systems of Inductive Logic Programming such as XHAIL (Ray, 2009), which raises the possibility of a linear algebraic implementation of ILP.

Finally, the ideas outlined in chapter 9 require full development, implementation and testing of their efficacy. Inference in the case of web-scale programs such as NELL (Mitchell et al., 2015) and KBpedia (Bergman and Giasson) remains an open challenge and these may be first steps towards overcoming it.

Appendix A

Legal and Ethical Considerations

The work that has been carried out throughout this project, and that which has been presented in this report, is of theoretical nature. Experiments that have been carried out are also of a theoretical nature on self-generated data, and did not involve any third party. Specifically, the project has not involved human embryos, humans, human cells or tissues, animals or developing countries. The author is not aware of any possible environmental impact by the project. No personal data has been collected or handled. No copyrighted code or data has been used or produced, to the best knowledge of the author. Great care has been made to credit original authors, when their work is referenced or built upon. The author is not aware of any direct possible misuse of this work. For more details, see the Ethics Checklist next page.

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		X
Does your project involve the use of human embryos?		X
Does your project involve the use of human foetal tissues / cells?		X
Section 2: HUMANS		
Does your project involve human participants?		X
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		X
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		X
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		X
Does it involve processing of genetic information?		X
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		X
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		X
Section 5: ANIMALS		
Does your project involve animals?		X
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		X
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		X
Could the situation in the country put the individuals taking part in the project at risk?		X
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		X
Does your project deal with endangered fauna and/or flora /protected areas?		X
Does your project involve the use of elements that may cause harm to humans, including project staff?		X
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		X
Section 8: DUAL USE		
Does your project have the potential for military applications?		X
Does your project have an exclusive civilian application focus?		X
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		X
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		X
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		X
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		X
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		X
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		X
SECTION 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		X
Will your project use or produce goods or information for which there are data protection, or other legal implications?		X
SECTION 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		X

Table A.1: Ethics Checklist

Bibliography

Apt, K. R., Blair, H. A., and Walker, A. (1988). Towards a theory of declarative knowledge. In *Foundations of deductive databases and logic programming*, pages 89–148. Elsevier. pages 54

Aspis, Y., Broda, K., and Russo, A. (2018). Tensor-based abduction in horn propositional programs. Accepted for publication in proceedings of the 28th International Conference on Inductive Logic Programming. pages 69, 75

Bergman, M. and Giasson, F. Kbpedia. `http://kbpedia.com/`. Accessed: 2018-08-21. pages 75

Besold, T. R., d'Avila Garcez, A. S., Bader, S., Bowman, H., Domingos, P. M., Hitzler, P., Kühnberger, K.-U., Lamb, L. C., Lowd, D., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H., and Zaverucha, G. (2017). Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902. pages 2, 70

Bottou, L. (1998). Online algorithms and stochastic approximations. In Saad, D., editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK. revised, oct 2012. pages 71

Bracewell, R. (2000). *The Fourier Transform and Its Applications*. Electrical engineering series. McGraw Hill. pages 26

Bratko, I. (2001). *Prolog (3rd Ed.): Programming for Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. pages 1, 62

Ciesielski, K. (1997). *Set Theory for the Working Mathematician*. London Mathematical Society Student Texts. Cambridge University Press. pages 9

Cohen, W. W. (2016). Tensorlog: A differentiable deductive database. *CoRR*, abs/1605.06523. pages 65

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425. pages 10

Fages, F. (1991). A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. *New Generation Computing*, 9(3):425–443. pages 10

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2012). Answer set solving in practice. 6:1–238. pages 1

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. pages 1070–1080. MIT Press. pages 2, 11

Grefenstette, E. (2013). Towards a formal distributional semantics: Simulating logical calculi with tensors. pages 58, 67

Kolda, T. G. and Bader, B. W. (2009). Tensor Decompositions and Applications. *SIAM REVIEW*, 51(3):455–500. pages 13

Lin, F. (2013). From satisfiability to linear algebra. Technical report. pages 65

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11 – 26. pages 1

Lloyd, J. and Shepherdson, J. (1991). Partial evaluation in logic programming. *The Journal of Logic Programming*, 11(3):217 – 242. pages 66

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., and Welling, J. (2015). Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*. pages 75

Parker, T. S. and Chua, L. O. (1989). *Practical Numerical Algorithms for Chaotic Systems*. Springer-Verlag, Berlin, Heidelberg. pages 2, 30

Ray, O. (2009). Nonmonotonic abductive inductive learning. 7:329–340. pages 75

Rocktäschel, T. and Riedel, S. (2017). End-to-end differentiable proving. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3788–3800. Curran Associates, Inc. pages 58, 65

Sakama, C., Inoue, K., and Sato, T. (2017). Linear Algebraic Characterization of Logic Programs. In *KSEM*, volume 10412 of *Lecture Notes in Computer Science*, pages 520–533. Springer. pages 2, 15, 18, 23, 32, 58, 65, 69, 74

Sakama, C., Nguyen, H., Sato, T., and Inoue, K. (EasyChair, 2018). Partial evaluation of logic programs in vector spaces. EasyChair Preprint no. 172. pages 66, 67

Sato, T. (2017a). Embedding Tarskian Semantics in Vector Spaces. pages 2, 58, 67, 68

Sato, T. (2017b). A linear algebraic approach to datalog evaluation. *TPLP*, 17:244–265. pages 68

Sato, T., Inoue, K., and Sakama, C. (2018). Abducing relations in continuous spaces. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1956–1962. International Joint Conferences on Artificial Intelligence Organization. pages 68, 69

Schölkopf, B., Smola, A. J., et al. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press. pages 72

Serafini, L. and Garcez, A. S. d. (2016). Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge. *CoRR*, abs/1606.04422. pages 65

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press. pages 73

Socher, R., Chen, D., Manning, C. D., and Ng, A. (2013). Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934. pages 65

Van Emden, M. H. and Kowalski, R. A. (1976). The Semantics of Predicate Logic as a Programming Language. *Journal ACM*, 23(4):733–742. pages 6, 7

Van Gelder, A., Ross, K. A., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):619–649. pages 45, 48

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, l. (2014). Embedding entities and relations for learning and inference in knowledge bases. pages 65