Supervisor: Dr. Maarten H. van Emden

Abstract

The work described in this dissertation is mainly a study of some ordinal-theoretic properties of logic programs that are related to the downward powers of their immediate-consequence functions. The downward powers for any program give rise to an interesting non-increasing sequence of interpretations, whose point of convergence is called the *downward closure ordinal* of that program. The last appearance of ground atoms that get eliminated somewhere in this sequence is called their *downward order.*

While it is well-known that there is no general procedure that can determine downward orders of atoms in any program, we present some rules for constructing such a procedure for a restricted class of programs.

Another existing result is that for every ordinal up to and including the least non-recursive ordinal (ω_1^{ck}), there is a logic program having that ordinal as its downward closure ordinal. However, the literature contains only a few examples of programs, constructed in an *ad hoc* manner, with downward closure ordinal greater than the least transfinite ordinal (ω). We contribute to bridging this wide gap between the abstract and concrete knowledge by showing the connection between some of the existing examples and the well-known concept of the order of a vertex in a graph. Using this connection and a convenient notation system for ordinals involving ground terms as bases, we construct a family $\{P_\alpha\}_{\alpha<\epsilon_0}$ of logic programs where ϵ_0 is the least fixpoint of the function $\lambda\beta[\omega^\beta]$ and any member P_α of the family has downward closure ordinal $\omega + \alpha$.

We also present an organization of a general transformation system, in which the objective is to search for transformations on syntax objects that satisfy pre-

established semantic constraints. As desired transformations are not always guaranteed to exist, we present necessary and sufficient conditions for their existence. In this framework, we proceed to give transformations on logic programs for the successor and addition operations on their downward closure ordinals.

Examiners:

Dr. M. H. van Emden
Supervisor
(Department of Computer Science)

Dr. C. G. Morgan
Outside Member
(Department of Philosophy)

Dr. M. R. Levy
Departmental Member
(Department of Computer Science)

Dr. G. G. Miller
Outside Member
(Department of Mathematics)

Dr. M. H. M. Cheng
Departmental Member
(Department of Computer Science)

Dr. M. A. Nait Abdallah
External Examiner
(Department of Computer Science)

Contents

List of Figures

Acknowledgements

I wish to express my gratitude to both the University of Victoria and the Advanced Systems Institute of British Columbia for their generous financial support over the years.

I would like to extend my appreciation to the members of my Examining Committee - Dr. van Emden, Dr. Morgan, Dr. Cheng, Dr. Miller, Dr. Levy and Dr. Nait Abdallah - for their useful comments and suggestions for the improvement of my dissertation.

My gratitude to my supervisor, Dr. Maarten H. van Emden, for his various suggestions of research topics and other valuable academic and personal help during my studies.

Finally, I would like to convey my deepest recognition to my former supervisor, Dr. William W. Wadge, for generating a life-long interest in me for areas such as logic, set theory, declarative languages etc., which led me to the choice of logic programming as my research area.

Chapter 1

Introduction

One of the ultimate goals of computer science has been to eliminate the gap between the specification of programs and their executable implementations. While each step toward the realization of this goal, namely the advancement from machine languages to assembly languages to high-order imperative languages and finally, to declarative languages has been significant in its own way, it was only with the advent of declarative languages that the goal seemed *almost* within grasp. This is because programs in declarative languages *are* executable specifications that, as a bonus, have mathematically elegant semantics.

Of the two broad categories of declarative languages, namely functional programming languages (based primarily on λ-calculus [11]) and logic programming languages (based on formal logic [9]), here we confine ourselves to studying some properties of the latter. In [17], Robinson proposed an important rule of inference, called resolution, that enables us to derive a contradiction from any inconsistent set of first-order formulas; this property of resolution is known as refutation completeness. Given some arbitrary set of axioms, whether or not a given formula is a theorem of those axioms can be determined by including the negation of that

formula in the original set of axioms and testing for the inconsistency of the resulting set. As resolution is easily mechanizable, refutation completeness makes it possible to use predicate logic as a programming language: a program in a logic programming language is viewed as a finite set of axioms of a theory and the computer's task is to determine whether or not a given formula is a logical consequence of those axioms.

A major drawback of resolution, however, is that at each step of the proof it can be applied in many different ways, thereby yielding many new facts of which only a few are relevant in order to arrive at the desired contradiction. As there is no easy method for controlling resolution to obtain only the relevant facts, any automatic inference mechanism based on resolution for full first-order logic tends to be inefficient. For practical reasons, it thus becomes necessary to restrict ourselves to some sublogic for which an efficient, yet refutationally complete rule of inference exists.

One such sublogic, called Horn logic [22], with the SLD-resolution rule of inference [14] meets these requirements. Though not quite as expressively powerful as full first-order logic, its expressive power is sufficient for it to be applied to a wide range of computational problems. By a *logic program* (or just *program*), in this dissertation, we mean programs with Horn logic as their underlying language.

We shall be particularly interested in studying certain ordinal-theoretic properties of logic programs. To do so, we shall associate ordinals with logic programs in two independent ways: first, we shall consider programs that axiomatize binary relations that are well orderings and second, we shall look at the downward closure ordinal of the immediate-consequence function associated with logic programs (all essential concepts are introduced later).

The next section of this chapter reviews some of the required background on logic programs.

Chapter 2 is motivated by a result of Blair [6] (see also Andreka and Nemeti [1]) that for every ordinal up to the least nonrecursive ordinal ω_1^{ck}, there exists a logic program that axiomatizes a well ordering of that order type. In this chapter we construct a family $\{Q_\alpha\}$ of logic programs indexed by ordinals up to ϵ_0 such that any member Q_α of this family axiomatizes a well ordering of type α. Our approach is based on constructing a convenient system of notations involving ground terms for such ordinals. This family of programs is then used to construct another family of programs in chapter 4 with transfinite downward closure ordinals.

In [22], van Emden and Kowalski give three distinct, yet equivalent semantics for logic programs: proof-theoretic, model-theoretic and fixpoint-theoretic. Of these, the last is achieved by associating with a program P, an immediate-consequence function T_P that maps a given set I of assumptions to the set of immediately obtainable consequences of I from P. The function T_P is shown to always have some fixpoints, of which the least (with respect to the partial order of set inclusion) is an intuitively straightforward characterization of the set of assertions denoted by P. The least fixpoint is then shown to be the ω-limit of the sequence

$$\langle T_P^\alpha(\emptyset)\rangle_{\alpha\geq 0}.$$

If B_P is the universal set of all assertions, then the sequence

$$\langle T_P^\alpha(B_P)\rangle_{\alpha\geq 0}$$

is of particular interest to us. It is well known (see, for example [15]) that this is a non-increasing sequence that converges to the greatest fixpoint of T_P. Members of the sets in this sequence are called *ground atoms* and the (possibly transfinite) value of α at which the sequence converges is called the *downward closure ordinal* of T_P, denoted $\mathrm{dco}(T_P)$.

The above sequence is strictly decreasing for values of index α up to and in-

cluding $\mathrm{dco}(T_P)$ and remains the same for all higher values of α. So some ground atoms appear in the sets of the sequence to a certain point and then disappear; other ground atoms appear in all the sets of the sequence. For ground atoms of the former kind, we let their *downward order* be the index of their last appearance in the sequence. Ground atoms of the latter kind do not have any downward order.

Chapter 3 deals with determining downward orders of ground atoms. While it is well known that there is no general procedure, which for every given program P and ground atom A will halt with an answer to whether or not A has a downward order, we attempt to give such a procedure for a limited class of programs and atoms. For atoms that have a downward order, our procedure halts with that order as its output.

Another important result by Blair [7] is that for every ordinal up to and including ω_1^{ck}, there is a logic program having that ordinal as its downward closure ordinal. Programs with finite downward closure ordinal can easily be constructed. The literature, for example [3] and [15], contains only a few examples of programs with transfinite downward closure ordinals, constructed in an *ad hoc* manner. Hence there is a painful contrast between what is known abstractly and what is known concretely. Chapter 4 attempts to soften this contrast by constructing a family $\{P_\alpha\}_{\alpha<\epsilon_0}$ of logic programs such that the downward closure ordinal of any one of its members P_α is $\omega + \alpha$. This family provides a general method for constructing programs with arbitrary transfinite downward closure ordinals.

Chapter 5 is concerned with transformations on logic programs with their downward closure ordinals in mind. It introduces an organization of a general transformation system, in which the objective is to search for syntactic transformations that satisfy certain semantic constraints. Since desired transformations do not always exist, it establishes necessary and sufficient conditions for their existence. In the setting of these general transformation systems, transformations are con-

structed for logic programs for the successor and addition operations on their downward closure ordinals.

Finally, chapter 6 presents some concluding remarks and directions for future work.

1.1 Theoretical Background

In this section we will review some of the required background on logic programs.

Definition A *first-order language* is a quadruple $\langle \Sigma, \Pi, V, \rho \rangle$, where

- Σ is a finite set whose members are called *function symbols*,
- Π is a finite set whose members are called *predicate symbols*,
- V is a denumerable set whose members are called *variables*, and
- $\rho: (\Sigma \cup \Pi) \rightarrow \omega$ is an *arity function* that assigns a natural number to each function and predicate symbol.

For simplicity, we assume Σ, Π and V to be pairwise disjoint and for reasons to be made clear shortly, assume the existence of at least one symbol c in Σ such that $\rho(c) = 0$. Such function symbols with zero arity are called *constants*. From here onward we use the term *language* for a first-order language $\langle \Sigma, \Pi, V, \rho \rangle$.

Definition The set of *terms* of a language is the smallest set T such that

- $V \subseteq T$, and
- if $f \in \Sigma$ and $\{t_1, \ldots, t_{\rho(f)}\} \subseteq T$ then $f(t_1, \ldots, t_{\rho(f)}) \in T$.

In other words, the set of terms is the smallest set that contains variables, constants and is closed under function symbol application.

Definition A term is said to be *ground* if no variables occur in it. The *Herbrand universe* of a language, denoted U, is the set of all ground terms of that language.

Clearly, $U \subseteq T$ and because of our assumption of at least one constant symbol in the language we have that $U \neq \emptyset$ It is worth noting that if, in addition, the language contains at least one function symbol with nonzero arity then the Herbrand universe is denumerable.

Definition The set A of *atoms* of a language is given by

$$\{q(t_1, \ldots, t_{\rho(q)}) \mid q \in \Pi, \{t_1, \ldots, t_{\rho(q)}\} \subseteq T\}.$$

Thus, an atom is a predicate symbol applied to an appropriate number of terms.

Definition An atom is said to be *ground* if all terms occurring in it are ground. The *Herbrand base* of a language, denoted B, is the set of all ground atoms of that language. A *Herbrand interpretation* is any subset of the Herbrand base.[1]

[1]Strictly speaking, an interpretation is a triple $\langle D, \sigma, \pi \rangle$ where D is a (nonempty) domain of discourse, σ maps an n-ary function symbol in Σ to some function with signature $D^n \rightarrow D$ and π maps every n-ary predicate symbol in Π to some subset of D^n. A Herbrand interpretation is an interpretation that has the Herbrand universe as its domain and maps any n-ary $f \in \Sigma$ to a function that maps a sequence $\langle t_1, \ldots, t_n \rangle$ of ground terms to the term '$f(t_1, \ldots, t_n)$'. Therefore, for simplicity we let a Herbrand interpretation be the set $\{q(t_1, \ldots, t_n) | q \in \Pi, \langle t_1, \ldots, t_n \rangle \in \pi(q)\}$ of ground atoms.

An example is in order. Consider the language given by

$$\Sigma = \{0, 1, \mathtt{nil}, \mathtt{cons}\},\ \Pi = \{\mathtt{append}\},\ V = \{X_1, X_2, \ldots\},$$
$$\rho(0) = \rho(1) = \rho(\mathtt{nil}) = 0,\ \rho(\mathtt{cons}) = 2,\ \rho(\mathtt{append}) = 3.$$

Then

$$T = \{\mathtt{nil}, X_1, \mathtt{cons}(\mathtt{nil}, 0), \mathtt{cons}(X_2, \mathtt{cons}(1, X_2)), \ldots\},$$
$$U = \{0, \mathtt{nil}, \mathtt{cons}(\mathtt{nil}, \mathtt{nil}), \mathtt{cons}(0, \mathtt{cons}(1, \mathtt{nil})), \ldots\},$$
$$A = \{\mathtt{append}(\mathtt{nil}, X_1, X_2), \mathtt{append}(\mathtt{cons}(1, X_1), X_1, X_3), \ldots\},$$
$$B = \{\mathtt{append}(\mathtt{nil}, \mathtt{nil}, \mathtt{nil}), \mathtt{append}(\mathtt{cons}(0, \mathtt{nil}), \mathtt{nil}, \mathtt{nil}), \ldots\}.$$

In all our examples we will use PROLOG's convention of symbols beginning with an upper case letter (like `A`, `X`, `Nat` etc.) for variables and all others as function and predicate symbols (see [9]). From the context we will always be able to tell function symbols apart from predicate symbols and also determine the arity of these symbols.

Definition A *clause* is of the form

$$A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$$

where each A_i and B_i is an atom.

The A_i's in the above definition are called *consequents* and the B_i's are known as *antecedents.*[2] A clause C is a *ground instance* of clause D if there is a function

[2]Each variable occurring in a clause is considered to be universally quantified at the outside. An intuitive reading of this clause is that if each antecedent is true then some consequent is true. Every first-order formula involving operators like negation, conjunction, quantifiers etc. can be converted (possibly after Skolemizations) to an equivalent clause. So this discussion is completely general.

$\theta: V \rightarrow U$ from the set of variables to the Herbrand universe such that C can be obtained by replacing each variable v in D by $\theta(v)$. Such a function θ is often called a *ground substitution.*

Definition A clause with exactly one consequent is called *definite*; a clause with no consequents is called *negative.* A *Horn clause* is either definite or negative.

As mentioned earlier, such a restriction to Horn clauses is sufficient for efficiency of an automatic inference mechanism, but that motivation does not concern us here. We are now in a position to define logic programs.

Definition A *logic program* (or just *program*) is a finite set of definite clauses.

As an example of a program consider the following set of clauses formed from the language given in the previous example:

```
append(nil,X,X) <-
append(cons(X,Y),Z,cons(X,C)) <- append(Y,Z,C)
```

which axiomatizes the append relation over lists constructed in the usual way from the function symbols nil and cons. We might now be interested in knowing whether the atom

```
append(cons(0,nil), cons(1,nil), cons(0,cons(1,nil)))
```

is a consequence of this program. A standard procedure is to add to the program the negative clause that has the above conjectured atom as its only antecedent and to turn on the SLD-resolution procedure. If the SLD-resolution procedure terminates with a refutation, the conjectured atom is a logical consequence. To

avoid getting sidetracked, we let the reader find out the details of SLD-resolution (see Lloyd [15]) and confirm that a refutation exists for the above atom.

Before proceeding to the following important definition, recall that a Herbrand interpretation is any subset of the Herbrand base.

Definition Let P be a program in a language with Herbrand base B. The *immediate-consequence function* $T_P: 2^B \rightarrow 2^B$ is defined as: for any Herbrand interpretation I,

$$\begin{aligned} T_P(I) = \{A \mid & \text{ there is a ground instance} \\ & A \leftarrow B_1, \ldots, B_n \\ & \text{of a clause in } P \text{ such that } \{B_1, \ldots, B_n\} \subseteq I\}. \end{aligned}$$

The immediate-consequence function maps one Herbrand interpretation to another. Intuitively, if each antecedent of some ground instance of a clause is in the argument interpretation then the consequent of the same ground instance of that clause is in the value interpretation. The reader familiar with universal quantifier elimination and *modus ponens* rules of inference will immediately recognize a similarity.

As it is a common error to conclude that $I \subseteq T_P(I)$, it is worth pointing out that it is not necessarily so. For instance, if P is the empty program and I is any non-empty interpretation then $T_P(I)$ is $\emptyset$. However, we have the following important property.

Proposition 1 *For any program P, T_P is monotone, i.e. $I \subseteq J$ implies $T_P(I) \subseteq T_P(J)$.*

A well-known theorem by Tarski (see Lloyd [15]) asserts that every monotone function over a complete lattice possesses a fixpoint. As the set of all Herbrand interpretations (power set of the Herbrand base) forms a complete lattice under the partial order of set inclusion (in fact a Boolean algebra), it follows that T_P has a fixpoint. It has been shown in [22] that the set of all ground atoms that are logical consequences of a program P is in fact the least fixpoint of T_P, denoted $\mathrm{lfp}(T_P)$. This is not surprising when one observes that $\mathrm{lfp}(T_P)$ is the union of $T_P^n(\emptyset)$ for all finite n, that is atoms obtainable by finite applications of *modus ponens* from the program containing ground instances of clauses in P. This motivates the following definition from [22, 3, 15].

Definition The *upward ordinal powers* of T_P are given by:

$$\begin{aligned} T_P \uparrow 0 &= \emptyset \\ T_P \uparrow \alpha &= T_P(T_P \uparrow (\alpha - 1)) \quad \text{if } \alpha \text{ is a successor ordinal} \\ &= \bigcup\{T_P \uparrow \beta | \beta < \alpha\} \quad \text{if } \alpha \text{ is a limit ordinal.} \end{aligned}$$

Informally stated, $T_P \uparrow \alpha$ is the Herbrand interpretation obtained by α applications of T_P to the empty Herbrand interpretation $\emptyset$. Although not immediately obvious, the following results are easily shown (see Lloyd [15]).

Proposition 2 **(a)** *The sequence* $\langle T_P \uparrow \alpha \rangle_{\alpha \geq 0}$ *is nondecreasing.*

(b) *There is an ordinal* δ *such that* $T_P \uparrow \delta = \mathrm{lfp}(T_P)$.

The upward ordinal powers can be viewed as a path from the bottom element $\emptyset$ to $\mathrm{lfp}(T_P)$ in the lattice of all Herbrand interpretations. As an example, let P be the program

```
even(0) <-
even(s(s(X))) <- even(X)
```

axiomatizing the **even** relation on representations of natural numbers using the constant **0** and the unary function symbol **s** for the successor operation. It can be seen that for any $n < \omega$,

$$T_P \uparrow n = \{\mathbf{even}(\mathbf{s}^{2i}(\mathbf{0})) \mid i < n\}$$

and that

$$T_P \uparrow \omega = \{\mathbf{even}(\mathbf{s}^{2i}(\mathbf{0})) \mid i < \omega\}.$$

Definition The least ordinal δ such that $T_P \uparrow \delta = \mathrm{lfp}(T_P)$ is called the *upward closure ordinal.*

The following well-known result is of interest.

Theorem 1 *The upward closure ordinal for any program is at most* ω.

It follows that the sequence $\langle T_P \uparrow \alpha \rangle_{\alpha \geq 0}$ converges no later than ω steps.

Note at this point that the equivalence of denotational, operational and fixpoint semantics simply means that for any ground atom A and program P, $P \models A$ iff SLΓ-resolution finds a refutation for the set $P \cup \{\leftarrow A\}$ of Horn clauses iff $A \in \mathrm{lfp}(T_P)$. These are all methods for characterizing the set of *positive* consequences of a logic program. As negative atoms do not occur in a program (their inclusion leads to ambiguous denotational semantics), there is no ground atom A such that $P \models \neg A$. However, if we assume that our program contains *all* the facts about our

world, then we are justified to say that $P \models \neg A$ iff $P \not\models A$. This assumption is known as the *closed world assumption* [8].

In general, there are three possible outcomes of an SLD-resolution procedure on the set $P \cup \{\leftarrow A\}$: it could terminate after having found a refutation (just in case $P \models A$), it could terminate after confirming the impossibility of a refutation, or it could not terminate at all. Clearly, under the closed world assumption, SLD-resolution is not complete with respect to negative consequences. But it is complete under a stronger definition of negative consequences, known as *negation by failure* (see Clark [8]), which defines $P \models \neg A$ iff SLD-resolution terminates on $P \cup \{\leftarrow A\}$ without finding a refutation. To obtain a fixpoint characterization of this set we are led to the following dual notion of the upward ordinal powers of T_P:

Definition The *downward ordinal powers* of T_P are given by:

$$\begin{aligned} T_P \downarrow 0 &= B \\ T_P \downarrow \alpha &= T_P(T_P \downarrow (\alpha - 1)) \quad \text{if } \alpha \text{ is a successor ordinal} \\ &= \bigcap\{T_P \downarrow \beta | \beta < \alpha\} \quad \text{if } \alpha \text{ is a limit ordinal.} \end{aligned}$$

Informally stated, $T_P \downarrow \alpha$ is the Herbrand interpretation obtained by α applications of T_P to the largest Herbrand interpretation (the entire Herbrand base). The downward ordinal powers approximate the greatest fixpoint of T_P, denoted $\text{gfp}(T_P)$, as mentioned in the following proposition.

Proposition 3 **(a)** *The sequence* $\langle T_P \downarrow \alpha \rangle_{\alpha \geq 0}$ *is nonincreasing.*

(b) *There is an ordinal* δ *such that* $T_P \downarrow \delta = \text{gfp}(T_P)$.

The downward ordinal powers can be viewed as a path from the top element B to $\text{gfp}(T_P)$ in the lattice of all Herbrand interpretations.

Definition The least ordinal δ such that $T_P \downarrow \delta = \text{gfp}(T_P)$ is called the *downward closure ordinal* (or just *dco*).

As expected, the dual counterpart of theorem 1 does not hold in general. That is, there exist logic programs whose downward closure ordinal exceeds ω. As an example, consider the program

```
p(f(X)) <- p(X)
q(a) <- p(X)
```

whose downward closure ordinal is $\omega + 1$. For $0 < n < \omega$, $T_P \downarrow n = \{\text{q(a)}\} \cup \{\text{p}(\text{f}^k(\text{a}))|n \leq k < \omega\}$. Thus $T_P \downarrow \omega = \{\text{q(a)}\}$. However, $T_P \downarrow (\omega + 1) = \emptyset = \text{gfp}(T_P)$.

The following proposition summarizes the fixpoint results mentioned thus far:

Proposition 4 *For any program P, $T_P \uparrow \omega = lfp(T_P) \subseteq gfp(T_P) \subseteq T_P \downarrow \omega$.*

Chapter 2

Axiomatizations of well orderings

One view of a logic program P is that of a set of axioms for a logical theory. The set of all clauses containing some n-ary predicate symbol, say $\mathtt{p}$, in their head characterizes the set of n-tuples of ground terms in the relation denoted by the symbol $\mathtt{p}$. An n-tuple $\langle t_1, \ldots, t_n \rangle$ of ground terms is in this relation iff the ground atom $\mathtt{p}(t_1, \ldots, t_n)$ is a logical consequence of P. The program can thus be treated as an axiomatization of the relations denoted by its predicate symbols.

An ordinal α can be viewed as the binary relation

$$\{\langle \beta, \delta \rangle \mid \beta < \delta < \alpha\}$$

over the ordinals smaller than α. An ordinal is said to be *recursive* if its underlying binary relation is recursive (see Rogers [18]). It is well-known (see, for example, Enderton [10]) that a relation (of any arity) on a countable set is recursive iff it is representable in some consistent finitely axiomatizable first-order theory. In fact, Blair, Andreka and Nemeti (see [6, 1, 2]) have shown a stronger result that such relations can be represented in a logic program.

Thus we have that for every recursive ordinal α, there is a logic program that axiomatizes a well ordering relation on some ground terms that is order-isomorphic

to the underlying binary relation of α. Such a relation is said to have *order type* α.

In section 2.1 we construct a family $\{Q_\alpha\}$ of programs indexed by ordinals up to ϵ_0, the least fixpoint of the function $\lambda\alpha[\omega^\alpha]$, such that any member Q_α of this family axiomatizes a well ordering of type α. Our approach is based on developing a convenient system of notations involving ground terms to represent ordinals up to ϵ_0. The family $\{Q_\alpha\}_{\alpha<\epsilon_0}$ of programs will then be used in the construction of another family of programs with transfinite downward closure ordinals in chapter 4. Finally, section 2.2 sketches a method for constructing bigger families of such programs by an iteration procedure for constructing systems of notations for larger recursive ordinals.

2.1 Axiomatization-based family of programs

In this section we consider axiomatizations of well orderings by logic programs.

Definition A program P is said to *axiomatize* a well ordering of order type α if P contains a unary predicate symbol $P^\odot$ and a binary predicate symbol $P^<$, such that the binary relation

$$\{\langle t_1, t_2\rangle \mid P \models P^\odot(t_1), P \models P^\odot(t_2), P \models P^<(t_1, t_2)\}$$

is a well ordering of order type α.

In other words, $P^<$ denotes some well ordering of order type α on ground terms in the predicate denoted by $P^\odot$.

Before constructing programs that axiomatize well orderings, we develop a system of notations for ordinals up to ϵ_0. The choice of ϵ_0 becomes clear after the

description of our notation system.

For representing natural numbers we use variable-free terms made from the constant 0 and the successor function symbol s, i.e. zero is represented by 0, one by s(0), two by s(s(0)) and so on. For any natural number n, we let $\bar{n}$ denote such a term representation of n. To represent ordinals[1] we use the following well-known result of their normal form expansions in base ω (see Sierpinski [20]):

Proposition 5 *Every ordinal number α, such that $0 < \alpha < \epsilon_0$, may be represented uniquely as*

$$\alpha = \omega^{\beta_1} c_1 + \omega^{\beta_2} c_2 + \cdots + \omega^{\beta_n} c_n$$

where n and $c_1, c_2, \ldots, c_n$ are non-zero natural numbers while $\beta_1, \beta_2, \ldots, \beta_n$ is a decreasing sequence of ordinals less than α.

As every ordinal number has a unique normal form, the representation of any ordinal number α, denoted $\lceil\alpha\rceil$, is the list of exponent-coefficient pairs appearing in the same order as in its normal form. Using the function symbol d to construct such pairs, $\lceil\alpha\rceil$ is given by the list

$$\langle \mathtt{d}(\lceil\beta_1\rceil,\bar{c_1}), \mathtt{d}(\lceil\beta_2\rceil,\bar{c_2}), \cdots, \mathtt{d}(\lceil\beta_n\rceil,\bar{c_n})\rangle.$$

By convention we let $\lceil 0\rceil$ be the empty list $\langle\rangle$. Note that a finite number $n > 0$ has different representations as a natural number and as an ordinal, since $\bar{n} = \mathtt{s}^n(\mathtt{0})$ whereas $\lceil n\rceil = \langle \mathtt{d}(\langle\rangle, \mathtt{s}^n(\mathtt{0}))\rangle$; also $\bar{0} = \mathtt{0}$ but $\lceil 0\rceil = \langle\rangle$.

As neither addition nor multiplication among ordinals is commutative, arithmetic becomes rather unfamiliar. After some simplification, it can be seen for example that the ordinal

$$(\omega + 1)2(\omega + 1)3(\omega + 1)4$$

[1] That is, there will be two representations for finite ordinals.

has the following normal form

$$\omega^3 4 + \omega^2 3 + \omega 2 + 1$$

and is represented by the list

$$\langle \mathtt{d}(\langle \mathtt{d}(\langle\rangle, \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0})))), \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))),$$
$$\mathtt{d}(\langle \mathtt{d}(\langle\rangle, \mathtt{s}(\mathtt{s}(\mathtt{0})))), \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0})))),$$
$$\mathtt{d}(\langle \mathtt{d}(\langle\rangle, \mathtt{s}(\mathtt{0}))), \mathtt{s}(\mathtt{s}(\mathtt{0}))),$$
$$\mathtt{d}(\langle\rangle, \mathtt{s}(\mathtt{0}))\rangle.$$

We allow lists to be nested to any finite depth. It may be verified that, with one level of nesting, this provides a representation for all ordinals up to (but not including) ω; with two levels of nesting, up to ω^ω; and so on. Thus, in the general case, we have a representation for all ordinals smaller than ϵ_0, which is the least fixpoint of the function $\lambda\alpha[\omega^\alpha]$.

We let the predicate `ord` be true of all lists that are representations of ordinals. The predicate `lto` specifies the less-than relation on ordinals, i.e. `lto`(X, Y) is true if the ordinal represented by X is less than that represented by Y. Representing lists as terms made up in the usual way from the constant `nil` and the binary functor '.', we can axiomatize `ord` as follows:

```
C1:  ord(nil);
C2:  ord(d(B,s(N)).nil) <- ord(B) & int(N);
C3:  ord(d(B1,s(N1)).d(B2,s(N2)).Rest) <-
                ord(B1) & int(N1) &
                ord(d(B2,s(N2)).Rest) &
                lto(B2,B1);
```

```
C4:  int(0);
C5:  int(s(X)) <- int(X);
```

Note that ord requires the coefficient fields to be non-zero and the pairs in the lists to be sorted in decreasing order of their exponent fields.

The $<$ relation on the ordinals induces a $<$ relation on their list representations such that $\lceil\alpha\rceil < \lceil\beta\rceil$ iff $\alpha < \beta$, for any ordinals $\alpha, \beta < \epsilon_0$. We have

$$\langle \mathtt{d}(\lceil\beta_1\rceil,\bar{c_1}),\cdots,\mathtt{d}(\lceil\beta_m\rceil,\bar{c_m})\rangle < \langle \mathtt{d}(\lceil\delta_1\rceil,\bar{d_1}),\cdots,\mathtt{d}(\lceil\delta_n\rceil,\bar{d_n})\rangle$$

if one of the following (mutually exclusive) cases hold:

- $m = 0, n > 0$;
- $m, n > 0$ and $\beta_1 < \delta_1$;
- $m, n > 0$ and $\beta_1 = \delta_1$ and $c_1 < d_1$;
- $m, n > 0$ and $\beta_1 = \delta_1$ and $c_1 = d_1$ and
  $\langle \mathtt{d}(\lceil\beta_2\rceil,\bar{c_2}),\cdots,\mathtt{d}(\lceil\beta_m\rceil,\bar{c_m})\rangle < \langle \mathtt{d}(\lceil\delta_2\rceil,\bar{d_2}),\cdots,\mathtt{d}(\lceil\delta_n\rceil,\bar{d_n})\rangle$.

These cases are directly translated to the following axioms for the lto predicate:

```
C6: lto(nil,X.Rest);
C7: lto(d(B1,N1).Rest1,d(B2,N2).Rest2) <- lto(B1,B2);
C8: lto(d(B,N1).Rest1,d(B,N2).Rest2) <- ltn(N1,N2);
C9: lto(d(B,N).Rest1,d(B,N).Rest2) <- lto(Rest1,Rest2);
```

The predicate ltn specifies the less-than ordering on natural numbers: ltn(X, Y) is true if the natural number X is less than the natural number Y. It is defined as.

```
C10: ltn(0,s(X));
C11: ltn(s(X),s(Y)) <- ltn(X,Y);
```

We are now in a position to construct a family $\{Q_\alpha\}_{\alpha<\epsilon_0}$ of logic programs such that any one member Q_α axiomatizes a well ordering of order type α. Clauses C1 through C11 form a basis for this family. In addition to these clauses, any member Q_α of this family contains the clauses:

$$\texttt{C12}: Q^{<}_{\alpha}(\texttt{X},\texttt{Y}) \leftarrow \texttt{lto(X,Y)};$$

$$\texttt{C13}: Q^{\circ}_{\alpha}(\texttt{X}) \leftarrow \texttt{ord(X)},\ \texttt{lto(X,}\alpha\texttt{)};$$

The following theorems are immediate:

Theorem 2 *$Q_\alpha \models Q^{\circ}_{\alpha}(\lceil\beta\rceil)$ iff $\beta < \alpha$.*

Theorem 3 *For all $\beta < \delta < \alpha$, $Q_\alpha \models Q^{<}_{\alpha}(\lceil\beta\rceil, \lceil\delta\rceil)$.*

In other words, the unary predicate $Q^{\circ}_{\alpha}(\texttt{X})$ is true if X is a representation of an ordinal smaller than α, and the binary predicate $Q^{<}_{\alpha}(\texttt{X},\texttt{Y})$ is true if the ordinal represented by X is smaller than the one represented by Y.

2.2 Larger families

We have presented a systematic way of constructing logic programs that axiomatize well orderings of order types up to ϵ_0, the least fixpoint of $\lambda\alpha[\omega^\alpha]$. Given Blair's result that there exist such programs for order types up to the least non-recursive ordinal (ω_1^{ck}), it is quite tempting to go beyond ϵ_0. Even though we have not considered that in this chapter, it seems to be a simple matter to arrive at closer approximations to ω_1^{ck}. The problem is essentially that of denoting ordinals by logic terms. We have presented a method to construct notations $\lceil\alpha\rceil$ for $\alpha < \epsilon_0$ given some notations $\bar{n}$ for $n < \omega$. This step can be iterated and increasingly large

initial segments of ordinals can be assigned notations by enlarging the base of their normal form expansions. For example, instead of ω, using ϵ_0 as the base yields a notation for all ordinals less than the least fixpoint of the function $\lambda\alpha[\epsilon_0^\alpha]$. Rogers [18] gives notation systems to go beyond even this value as follows. Let

$$\begin{aligned} \gamma_0 &= \epsilon_0, \\ \gamma_{n+1} &= \text{least ordinal not expressible as} \\ &\quad \text{a polynomial of } \gamma_n. \end{aligned}$$

Let us call a system of notations *maximal* if it assigns a notation to every recursive ordinal. Then, since for all n, $\gamma_n < \omega_1^{ck}$, any notation system for ordinals up to γ_n will fail to be maximal. A system of notations is said to be *recursively related* if there exists an effective procedure, which, when given any two representations in that system can tell us which one represents a smaller ordinal. A classic result by Kleene (see Rogers [18]) is that there is no maximal recursively related system of notations. As we need our system to be recursively related (to axiomatize the `lto` predicate), it follows that our method cannot be generalized to a maximal family of programs.

Chapter 3

Determining downward orders of ground atoms

Both the upward as well as the downward ordinal powers of T_P, defined in chapter 1, give rise to rather interesting sequences of interpretations of the program P.

The sequence $\langle T_P \uparrow \alpha \rangle_{\alpha \geq 0}$, obtained from the upward ordinal powers of T_P, was seen to be non-decreasing and one which converges at no later than ω steps to $\mathrm{lfp}(T_P)$. For every ground atom A in $\mathrm{lfp}(T_P)$, there is thus a finite point, called the *upward order* of A, when it first appears in the sequence. Ground atoms that do not occur in $\mathrm{lfp}(T_P)$ do not have an upward order. Operationally, the upward order of a ground atom A gives the minimum number of *modus ponens* inferences needed to deduce A from the program. Intuitively, it is a measure of the "distance" of a theorem A from the axiom set P and, in a sense, may be considered as the complexity of A with respect to P. It can easily be shown that there is an effective procedure that, for any given program P and ground atom $A \in \mathrm{lfp}(T_P)$, halts with the upward order of A as its output.

The sequence $\langle T_P \downarrow \alpha \rangle_{\alpha \geq 0}$, on the other hand, obtained from the downward

ordinal powers of T_P, was seen to be non-increasing and one that converges to $\text{gfp}(T_P)$. However, unlike the previous sequence, this sequence does not necessarily converge within ω steps. In fact, Blair [7] has shown that for some programs, this sequence might converge as late as ω_1^{ck} (the least non-recursive ordinal). Thus for every ground atom A in $\overline{\text{gfp}(T_P)}$, there is a well-defined, though not necessarily finite point, called the *downward order* of A, when it last appears in the sequence. Ground atoms contained in $\text{gfp}(T_P)$ do not have a downward order. Due to the transfinite nature of downward orders they do not have any straightforward operational interpretation. Intuitively, however, the downward order of a ground atom $A \in \overline{\text{gfp}(T_P)}$, is a measure of the minimum amount of work needed to determine that A is not a theorem of the axiom set P. Later we show that there is no effective procedure that, for any given program P and ground atom $A \in \overline{\text{gfp}(T_P)}$, halts with the downward order of A as its output.

In this chapter we address the problem of determining the downward orders for a limited class of programs and ground atoms. We consider programs with certain restricted structures of clauses constructed from unary predicate symbols and a function symbol $\mathtt{f}$. We develop a method that, for every predicate symbol $\mathtt{p}$ and constant $\mathtt{a}$, computes an ordinal expression $\mathcal{E}(k)$ for the downward order of any atom of the form $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$. The method is based on studying the structures of clauses contained in the program.

In section 3.1 we give a precise definition of downward orders of ground atoms and show that the downward order function is in general non-computable. In section 3.2 we determine downward orders of atoms for a carefully chosen example by studying the structures of its clauses. The rules employed to do so are then formalized in section 3.3. These rules can then be incorporated into an automatic procedure.

3.1 Orders of ground atoms

Let P be a program. The following proposition is straightforward:

Proposition 6 *For all α, β, if $\alpha \leq \beta$ then $T_P \uparrow \alpha \subseteq T_P \uparrow \beta$.*

It is thus clear that if a ground atom A is in $T_P \uparrow \alpha$, for some α, then for all $\beta > \alpha$, $A \in T_P \uparrow \beta$.

Definition For any $A \in \text{lfp}(T_P)$, the *upward order* of A is the least ordinal α such that $A \in T_P \uparrow \alpha$.

The existence of an upward order of atoms in $\text{lfp}(T_P)$ is guaranteed as the ordinals are well ordered. It can easily be shown that upward orders are always non-zero and finite. Moreover, the upward order of any atom $A \in \text{lfp}(T_P)$ is equal to the length of the shortest successful SLD-derivation (see Lloyd [15]) starting from the negative clause $\leftarrow A$. Thus the upward order of A can easily be determined by traversing an SLD-tree with $\leftarrow A$ at its root in breadth-first order until a successful SLD-derivation is detected.

We now develop a notion dual to the upward orders of ground atoms. The following proposition is also straightforward:

Proposition 7 *For all α, β, if $\alpha \leq \beta$ then $T_P \downarrow \alpha \supseteq T_P \downarrow \beta$.*

From the above proposition it follows immediately that if $A \notin T_P \downarrow \alpha$, for some α, then for all $\beta > \alpha$, $A \notin T_P \downarrow \beta$. However, the following important result is not obvious:

Proposition 8 *For any ground atom A, if there is an ordinal α such that $A \notin T_P \downarrow \alpha$ then there is a greatest ordinal β such that $A \in T_P \downarrow \beta$.*

Proof If $A \notin T_P \downarrow \alpha$ then by the well ordering property of ordinals there exists a least ordinal α' such that $A \notin T_P \downarrow \alpha'$. We know that $\alpha' \neq 0$ since $A \in B_P = T_P \downarrow 0$; so α' must be either a successor or a limit ordinal. The latter case can be excluded since then by the definition $T_P \downarrow \alpha' = \bigcap\{T_P \downarrow \beta \mid \beta < \alpha'\}$, we have that $A \notin T_P \downarrow \alpha'$ implies $A \notin T_P \downarrow \beta$ for some $\beta < \alpha'$, contradicting the minimality of α'. Thus the only possible case turns out to be $\alpha' = \beta + 1$ for some β, which is the desired maximal solution of $A \in T_P \downarrow \beta$. □

Definition For any $A \in \overline{\text{gfp}(T_P)}$, the *downward order* (or just *order*) of A is the greatest ordinal α such that $A \in T_P \downarrow \alpha$.

The existence of downward orders for atoms in $\overline{\text{gfp}(T_P)}$ is guaranteed by proposition 8. Unlike upward orders, downward orders are not necessarily finite. As an example, the atom `q(a)` in the program

```
p(f(X)) <- p(X)
q(a) <- p(X)
```

first introduced in chapter 1, can be seen to have downward order ω.

Let DO be the function that maps ground atoms in B_P to their downward orders. Clearly, DO is partial as it is not defined for elements in $\text{gfp}(T_P)$. In [19] it is shown that there exist programs for which $\overline{\text{gfp}(T_P)}$ is not recursively enumerable. Thus for such programs the problem of determining whether or not a ground atom has a downward order is not even semi-decidable. Moreover, there is no effective procedure that takes any $A \in \overline{\text{gfp}(T_P)}$ as input and is guaranteed to halt with $DO(A)$ as its output. The following is a precise formulation of this result:

Proposition 9 *There exists a program P for which DO is not computable over $\overline{gfp(T_P)}$.*

Proof (By contradiction) By results in [19] we know that there exists a program P for which $\overline{\text{gfp}(T_P)}$ is not r.e.. Suppose DO is computable for this program; i.e. there is an effective procedure E_P that, for any $A \in \overline{\text{gfp}(T_P)}$ terminates in $k_A > 0$ steps with $DO(A)$ as its output.

Let $\langle A_0, A_1, \cdots \rangle$ be some enumeration of the countable set B_P. (As B_P is r.e. such an enumeration can be generated.) Now consider a procedure R, which executes one step of E_P on A_0; A_0, A_1; A_0, A_1, A_2; ... in that order. In other words, R performs passes over the enumerated atoms and in the ith pass executes one step of E_P on each of the atoms A_0 through A_i. If in the process E_P terminates for some A_j $(0 \leq j \leq i)$ with $DO(A_j)$ as its output, R outputs A_j and omits A_j in the subsequent passes.

It is easily seen that R will enumerate the elements of $\overline{\text{gfp}(T_P)}$, but the existence of such a procedure R is a contradiction. □

The above proposition shows that there exists a program P for which there is no procedure E_P that takes any $A \in \overline{\text{gfp}(T_P)}$ as input and halts with $DO(A)$ as its output. But the proof assumes that $\overline{\text{gfp}(T_P)}$ is non-r.e.. However, there are many programs for which this is not the case, i.e. $\overline{\text{gfp}(T_P)}$ *is* r.e.. (In fact, despite their abundance, it is very difficult to construct examples of programs P for which $\overline{\text{gfp}(T_P)}$ is non-r.e.. The simplest one the author has come across was adapted from an undecidable propositional calculus in [21] and is about 16 clauses.) So for these programs DO might be computable. Any canonical program (first defined in [12], for which $\text{gfp}(T_P) = T_P \downarrow \omega$) is an example of this case because $\overline{T_P \downarrow \omega}$ is known to be r.e. for all programs.

3.2 An example of determining downward orders

Let us call a program *orderly* if its DO function is computable. In this section we address the problem of computing DO for some orderly programs. Note that given an orderly program P, we know that there is a procedure E_P to compute its DO function. But we may not have a single procedure E, which will do the job for any orderly program. At present we do not know if such a general procedure exists or not, but attempt to arrive at some guidelines here to construct one for programs with certain restricted kinds of clauses.

We start with an example. Let P be the following program taken from [15]:

```
1: p(a) <- p(X) & q(X);
2: p(f(X)) <- p(X);
3: q(b);
4: q(f(X)) <- q(X);
```

As for any program, $T_P \downarrow 0 = B_P$. $T_P \downarrow 1$ can be seen to be $T_P \downarrow 0 \setminus \{\mathtt{p(b)}, \mathtt{q(a)}\}$ as these two atoms cannot be unified with the head of any clause. $T_P \downarrow 2$ is similarly seen to be $T_P \downarrow 1 \setminus \{\mathtt{p(f(b))}, \mathtt{q(f(a))}\}$. In general, for any $0 < k < \omega$, $T_P \downarrow k$ is $T_P \downarrow (k-1) \setminus \{\mathtt{p}(\mathtt{f}^{k-1}(\mathtt{b})), \mathtt{q}(\mathtt{f}^{k-1}(\mathtt{a}))\}$. Thus the order of $\mathtt{p}(\mathtt{f}^k(\mathtt{b}))$ or $\mathtt{q}(\mathtt{f}^k(\mathtt{a}))$ is k. The limit of the finite downward powers of T_P is $T_P \downarrow \omega$, which is

$$\{\mathtt{p}(\mathtt{f}^k(\mathtt{a})) \mid k < \omega\} \bigcup \{\mathtt{q}(\mathtt{f}^k(\mathtt{b})) \mid k < \omega\}.$$

We can continue further in the same way so that for any $0 < k < \omega$, $T_P \downarrow (\omega + k) = T_P \downarrow (\omega + k - 1) \setminus \{\mathtt{p}(\mathtt{f}^k(\mathtt{a}))\}$. Thus the order of $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$ is $\omega + k$. $T_P \downarrow \omega 2$ is the fixpoint (both the greatest as well as least) of T_P containing $\mathtt{q}(\mathtt{f}^k(\mathtt{b}))$ for all $k < \omega$. These atoms, therefore, do not have an order.

Let us now try to analyze the reasoning employed in the above procedure for determining the orders of the atoms. In order to determine the order of an atom A, we clearly need to find the point in the decreasing sequence $\langle T_P \downarrow \alpha \rangle_{\alpha \geq 0}$ where the atom first falls out. This is best done by studying the pattern of atoms eliminated in this sequence. For instance, in the above example, the atoms $\{\mathtt{p(b)}, \mathtt{q(a)}\}$ drop after step 0, $\{\mathtt{p(f(b))}, \mathtt{q(f(a))}$ after step 1, etc.; in general, $\{\mathtt{p}(\mathtt{f}^k(\mathtt{b})), \mathtt{q}(\mathtt{f}^k(\mathtt{a}))\}$ after step k. Thus $T_P \downarrow \omega$ does not have any of these atoms and the procedure is continued thereafter.

Though there are programs for which no (observably) regular pattern exists, the above method is applicable only on ones for which such a pattern exists. Moreover, for such programs, orders of atoms can be computed by just studying the clauses in the program (rather than the pattern of eliminated atoms) and is thus easily mechanizable. Here we restrict ourselves to programs with unary predicate symbols and one unary function symbol f. We outline the mechanics of a procedure, which for any such program, given a predicate symbol p and a constant a, is able to compute an expression $\mathcal{E}(k)$ denoting the order of an atom of the form $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$.

The problem is essentially to fill a matrix whose rows correspond to predicate symbols and columns to constants. For our example, the matrix is:

	a	b
p		
q		

Any entry, for example the one for p and a, will finally contain an expression denoting the order of $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$.

Consider first the entry for p and b. Since any atom of the form $\mathtt{p}(\mathtt{f}^k(\mathtt{b}))$ unifies with the head of only the second clause, the order of the atom is seen from the structure of the clause to be $DO(\mathtt{p(b)}) + k$. Since p(b) does not unify with the

head of any clause, its order is 0; hence the order of $\mathtt{p}(\mathtt{f}^k(\mathtt{b}))$ is k. By a similar reasoning we can determine the order of $\mathtt{q}(\mathtt{f}^k(\mathtt{a}))$ to be k. Our partially filled matrix looks like:

	a	b
p		k
q	k	

Due to the fourth clause $DO(\mathtt{q}(\mathtt{f}^k(\mathtt{b}))) = DO(\mathtt{q}(\mathtt{b})) + k$, and since $\mathtt{q(b)}$ unifies with the head of the third clause that has an empty body (meaning $\mathtt{q(b)}$ does not have a downward order), $\mathtt{q}(\mathtt{f}^k(\mathtt{b}))$ does not have a downward order either. In our matrix, we write ∞ to indicate that the atom $\mathtt{q}(\mathtt{f}^k(\mathtt{b}))$ does not ever get eliminated in the sequence $\langle T_P \downarrow \alpha \rangle_{\alpha \geq 0}$ of interpretations of the program:

	a	b
p		k
q	k	∞

From the second clause, we have that $DO(\mathtt{p}(\mathtt{f}^k(\mathtt{a}))) = DO(\mathtt{p}(\mathtt{a})) + k$. Due to the first clause, $DO(\mathtt{p}(\mathtt{a}))$ is the maximum of the minimums of the $\lim_{k \to \omega}$ value of each filled entry in the columns; i.e.

$$\begin{aligned} DO(\mathtt{p}(\mathtt{a})) &= \max(\min(\lim_{k \to \omega} k), \min(\lim_{k \to \omega} k, \lim_{k \to \omega} \infty)) \\ &= \omega \end{aligned}$$

where ∞ is considered greater than all ordinals. The exact reason for computing the maximum of the minimums becomes clear in the next section. Thus $DO(\mathtt{p}(\mathtt{f}^k(\mathtt{a})))$ turns out to be $\omega + k$. The filled matrix becomes:

	a	b
p	$\omega + k$	k
q	k	∞

3.3 An automatic procedure

In the previous section we computed ordinal expressions for downward orders of ground atoms of the kind $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$ for an example program by informally studying the structure of the clauses in the program. Here we formalize the general rules for doing so for the kinds of clauses we came across. These rules can then be incorporated into an automatic procedure to compute the *DO* function for programs with such clauses.

As before, we assume that the program contains only unary predicate symbols and one unary function symbol $\mathtt{f}$, and will try to fill ordinal expressions in the matrix associated with the program. Let M be this matrix. The rows of M are labelled with the predicate symbols and the columns with the constants appearing in the program. Any entry of this matrix, say $M(\mathtt{q}, \mathtt{a})$, will finally contain an expression for the downward order of atoms of the form $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$. The variable k would occur free in the expressions.

We now give some rules for determining downward orders.

Rule 1 If an atom does not unify with the head of any clause of the program, its downward order is 0.

The above rule is immediate from the observation that such an atom will not appear in $T_P \downarrow 1$. The downward order of the atom $\mathtt{p}(\mathtt{b})$ (for the computation of the downward order of any atom of the form $\mathtt{p}(\mathtt{f}^k(\mathtt{b}))$ in the example of the previous section was determined by this rule.

Rule 2 If an atom unifies with more than one clause of the program then its downward order is the maximum of the downward orders derived from each of

those clauses.

The above rule is also immediate since an atom stays in the sequence $\langle T_P \downarrow \alpha \rangle_{\alpha \geq 0}$ of interpretations as long as there is some clause that enables it to do so. As an example of this rule, let the clauses

```
p(X) <- q(X)
p(a) <- r(b)
```

be the only ones in some program with the predicate symbol p occurring in their head. It is easily seen that $DO(\mathtt{p(a)})$ is the maximum of $DO(\mathtt{q(a)}) + 1$ and $DO(\mathtt{r(b)}) + 1$.

Rule 3 If an atom unifies with the head of a clause with no antecedents, it does not have a downward order. This is indicated in the matrix M by the symbol ∞.

The above rule follows from the observation that such an atom is in $\mathrm{lfp}(T_P)$, which is contained in $\mathrm{gfp}(T_P)$. Members of $\mathrm{gfp}(T_P)$ do not have a downward order. The downward order of the atom q(b) (for the computation of the downward order of any atom of the form $\mathtt{q}(\mathtt{f}^k(\mathtt{b}))$ in the example of the previous section was determined by this rule.

Rule 4 The downward order of the atom $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$ due to a clause of the form

$$\mathtt{p}(\mathtt{f}^n(\mathtt{X})) \leftarrow \mathtt{p}(\mathtt{X})$$

is $DO(\mathtt{p(a)}) + \lfloor k/n \rfloor$.

The above rule becomes immediate from studying the pattern of atoms of the kind $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$ eliminated in the downward ordinal powers of T_P. This rule was used for filling each of the entries in the matrix of the example in the previous section.

Rule 5 The downward order of a ground atom A due to a clause of the form

$$\begin{aligned} A \leftarrow\ & q_1^1(X_1)\ \&\ q_2^1(X_1)\ \&\ \ldots\ \&\ q_{n_1}^1(X_1)\ \& \\ & q_1^2(X_2)\ \&\ q_2^2(X_2)\ \&\ \ldots\ \&\ q_{n_2}^2(X_2)\ \& \\ & \vdots \\ & q_1^m(X_m)\ \&\ q_2^m(X_m)\ \&\ \ldots\ \&\ q_{n_m}^m(X_m) \end{aligned}$$

is

$$\min\{\max\{\min\{\lim_{k\to\omega} M(q_j^i, c) \mid 1 \le j \le n_i\} \mid c \in \mathcal{C}\} \mid 1 \le i \le m\}$$

where $\mathcal{C}$ is the set of constants appearing in the program and the X_i's are distinct variables.

Understanding the above rule requires some analysis of the clause. The outer min operation appears because of the conjunction between the groups of atoms for each X_i in the body. Similarly, the inner min operation appears because of the conjunctions between different atoms containing the same X_i. The max operation is due to the occurence of variables in the antecedents: a variable can be instantiated by a term $\mathtt{f}^k(c)$ for each c in $\mathcal{C}$. The lim operation appears because terms, although finite, can be arbitrarily long.

The above rule was used in determining the downward order of the atom $\mathtt{p}(\mathtt{a})$ in the example program of the previous section. To obtain a better understanding of this rule, consider the following clause in some program P with $\mathtt{a}$ and $\mathtt{b}$ as the only constants:

```
d(a) <- p(X) & q(X) &
        q(Y) &
        p(Z) & r(Z) & t(Z)
```

Due to this clause, the atom `d(a)` would appear in the sequence $\langle T_P \downarrow \alpha \rangle_{\alpha \geq 0}$ as long as there is some ground substitution for the variables `X`, `Y` and `Z` for which the ground instances of all atoms in the clause body appear collectively in the sequence. As `a` and `b` are assumed to be the only constants in P, each of the variables `X`, `Y` and `Z` can be bound either to a ground term of the form $\mathtt{f}^k(\mathtt{a})$ or of the form $\mathtt{f}^k(\mathtt{b})$. Consider `X` for the moment. Due to the conjunction between the atoms containing `X` in the clause body, namely `p(X)` and `q(X)`, these atoms contribute to the appearance of the atom `d(a)` in the downward ordinal power sequence as long as either atoms of the form $\mathtt{p}(\mathtt{f}^k(\mathtt{a}))$ and $\mathtt{q}(\mathtt{f}^k(\mathtt{a}))$ appear collectively or atoms of the form $\mathtt{p}(\mathtt{f}^k(\mathtt{b}))$ and $\mathtt{q}(\mathtt{f}^k(\mathtt{b}))$ appear collectively in that sequence. In other words, they contribute as long as

$$\begin{aligned}
&\max\{ \\
&\quad \min\{\lim_{k\to\omega} M(\mathtt{p},\mathtt{a}), \lim_{k\to\omega} M(\mathtt{q},\mathtt{a})\}, \\
&\quad \min\{\lim_{k\to\omega} M(\mathtt{p},\mathtt{b}), \lim_{k\to\omega} M(\mathtt{q},\mathtt{b})\} \\
&\}
\end{aligned}$$

steps of the sequence. Similarly, the atom `q(Y)` contributes as long as

$$\begin{aligned}
&\max\{ \\
&\quad \min\{\lim_{k\to\omega} M(\mathtt{q},\mathtt{a})\}, \\
&\quad \min\{\lim_{k\to\omega} M(\mathtt{q},\mathtt{b})\} \\
&\}
\end{aligned}$$

and the atoms containing the variable `Z` as long as

$$\max\{$$

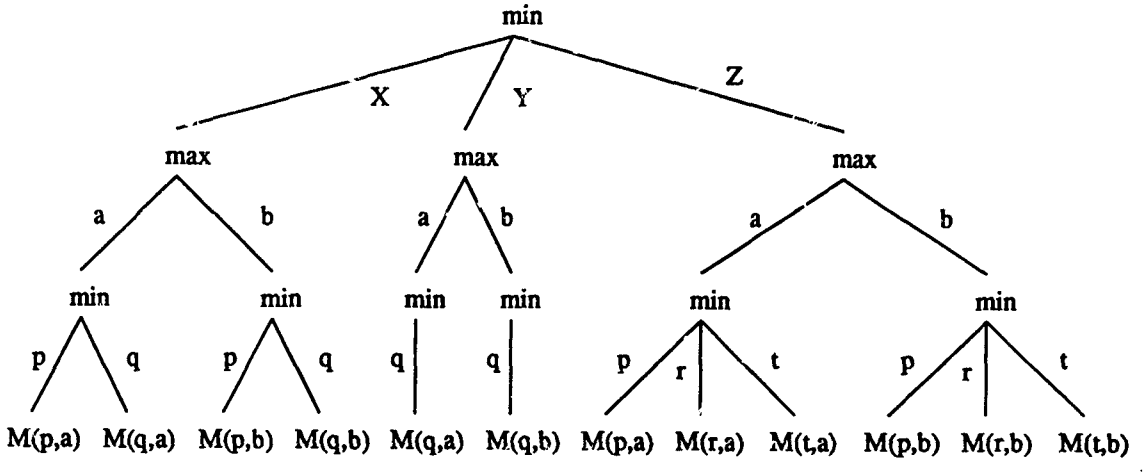

Figure 3.1: Downward order expression tree

$$\min\{\lim_{k\to\omega} M(\mathtt{p},\mathtt{a}), \lim_{k\to\omega} M(\mathtt{r},\mathtt{a}), \lim_{k\to\omega} M(\mathtt{t},\mathtt{a}),\},$$
$$\min\{\lim_{k\to\omega} M(\mathtt{p},\mathtt{b}), \lim_{k\to\omega} M(\mathtt{r},\mathtt{b}), \lim_{k\to\omega} M(\mathtt{t},\mathtt{b})\}$$
$$\}.$$

The downward order of the atom d(a) is thus the minimum of the above three expressions. Figure 3.3 gives a pictorial representation of this expression.

Although the above rule seems rather restrictive, in that the function symbol f does not appear in the body of the clause, the restriction is only for ease of understanding. In fact even if any atom $q_j^i(X_i)$ therein is replaced by the more general atom $q_j^i(f^{k_j^i}(X_i))$, it can be seen that the downward order of A remains exactly the same.

Chapter 4

Downward Closure Ordinal Based Family of Logic Programs

In chapter 3 we introduced the notion of downward orders of ground atoms of a program. In general, the downward order map, *DO*, was seen to be partial over the Herbrand base of the program. In this chapter, we study a more general aspect of a logic program, namely its *downward closure ordinal.* Much of the work described here first appeared in [4]. While a more precise definition of downward closure ordinals appears later, for now it suffices to view the downward closure ordinal of a program as the least ordinal greater than the downward orders of all ground atoms for which *DO* is defined.

Examples of programs with downward closure ordinals up to ω can easily be constructed. Blair has shown in [7] that for every ordinal up to and including the least non-recursive ordinal (ω_1^{ck}), there is a logic program having it as downward closure ordinal. However, given such an ordinal and Blair's proof, it is not straightforward to find a program having this ordinal as downward closure ordinal. There is a painful contrast between the richness, *in abstracto*, assured by Blair's

theorem and the meagreness of what is known concretely: the literature presents only a few isolated examples of programs with downward closure ordinal greater than ω, presented in an *ad hoc* manner. It is the purpose of this chapter to soften this contrast.

A basis of our approach is provided by the well-known notion of ordering the vertices of an acyclic directed graph by assigning to each vertex an ordinal number, which may or may not be finite. The other basis is a connection between graphs and logic programs provided by a variant of Kowalski's reachability representation of graphs. These fundamentals allow us to "explain" some of the published examples of logic programs having downward closure ordinal exceeding ω. More importantly, they suggest a family of logic programs having as downward closure ordinals all those ordinals for which some convenient notation system applies. For this purpose we use the notation system developed in chapter 2.

In Section 4.1 we review some of the theory on closure ordinals. In the next section we explain some of the examples by relating them to known concepts in graph theory. In Section 4.3 we prepare for a more general treatment by exploiting generally applicable representations of graphs by means of logic programs. Section 4.4 is devoted to the construction of a family of logic programs indexed by ordinals up to ϵ_0, the least fixpoint of the function $\lambda\alpha[\omega^\alpha]$. Each member P_α of this family represents a graph, and has $\omega + \alpha$ as downward closure ordinal.

4.1 Closure ordinals

We start by restating a definition that first appeared in chapter 1.

Definition The *upward closure ordinal* of T_P is the least ordinal α such that $T_P \uparrow \alpha = \mathrm{lfp}(T_P)$; the *downward closure ordinal* of T_P, denoted $\mathrm{dco}(T_P)$, is the

least ordinal α such that $T_P \downarrow \alpha = \text{gfp}(T_P)$.

As shown in Lloyd [15], both the upward and downward closure ordinals of T_P exist for any logic program P. Moreover, the upward closure ordinal is at most ω.

The downward closure ordinal, on the other hand, is known to exceed ω for a large class of programs. Consider for example the following program:

```
p(f(X)) <- p(X)
q(a) <- p(X)
```

whose downward closure ordinal was first seen in chapter 1 to be $\omega + 1$.

Certain classes of programs have downward closure ordinals of at most ω; for example, if in every clause the conclusion contains all variables that occur in the clause.

For any program P, it is the case that $T_P \uparrow \alpha \subseteq T_P \downarrow \beta$, for all α and β. An important class is that of determinate programs as defined in Blair [7]:

Definition A program P is *determinate* if $T_P \uparrow \omega = T_P \downarrow \omega$.

Proposition 10 *If P is determinate, $dco(T_P) \leq \omega$.*

Proof For any P, we have that $T_P \uparrow \omega = \text{lfp}(T_P) \subseteq \text{gfp}(T_P) \subseteq T_P \downarrow \omega$. Thus determinacy implies that $T_P \downarrow \omega$ is a (in fact, the only) fixpoint. The proposition then follows from the definition of dco. □

Determinacy of a program is a fixpoint-theoretic property. We define a stronger proof-theoretic property of programs using the notion of SLD-derivation, which is described in Lloyd [15].

Definition A program is *well founded* if no infinite SLD-derivation starts from a negative clause consisting of a single variable-free atomic formula.

Proposition 11 *Every well founded program is determinate.*

Proof Let P be a well founded program and $A \in B_P$, i.e. A is any variable-free atom. Since any SLD-derivation starting from the negative clause $\leftarrow A$ is finite, any SLD-tree with $\leftarrow A$ as the root is either finitely failed or contains a successful derivation. That is, A is either in the finite-failure set (equal to $B_P \setminus T_P \downarrow \omega$, see Lloyd [15]) or in the success set (equal to $T_P \uparrow \omega$). Therefore, $T_P \uparrow \omega = T_P \downarrow \omega$. □

As it is possible for a variable-free negative clause $\leftarrow A$ to begin a successful as well as an infinite SLD-derivation, the converse of proposition 11 does not hold in general. For example consider the program P:

```
q;
q <- q;
```

This program is determinate because

$$T_P \uparrow \omega = \{\mathtt{q}\} = T_P \downarrow \omega.$$

However, P is not well founded since the variable-free clause `<- q` has an infinite derivation, namely

$$\mathtt{<-\ q}, \mathtt{<-\ q}, \mathtt{<-\ q}, \ldots$$

For use in later sections we establish here the following result:

Proposition 12 *Let P be a logic program such that all predicate symbols in it have non-zero arity and every term occurring in the body of any of its clauses is a proper subterm of a term occurring in the head of the same clause. Then P is well founded.*

Proof Straightforward by structural induction on the terms occurring in any variable-free negative clause. □

The relation between a program's structure and its downward closure ordinal is not well understood. In the literature, a few isolated examples are exhibited as curiosities to show that the value of this function can exceed ω. The example shown above, first published in [3], is due in part to K.L. Clark and in part to H. Andreka and I. Nemeti. Here we discuss the other examples found in Lloyd [15].

Example 1 Let P be the program

```
p(f(X)) <- p(X);
q(a) <- p(X);
q(f(X)) <- q(X);
r(a) <- q(X);
r(f(X)) <- r(X);
s(a) <- r(X);
s(f(X)) <- s(X);
t(a) <- s(X);
t(f(X)) <- t(X);
```

Then we have

$$\begin{aligned}
T_P \downarrow 0 &= B_P, \\
T_P \downarrow \omega &= T_P \downarrow 0 \setminus \{\mathtt{p}(\mathtt{f}^k(\mathtt{a}))|k < \omega\}, \\
T_P \downarrow \omega 2 &= T_P \downarrow \omega \setminus \{\mathtt{q}(\mathtt{f}^k(\mathtt{a}))|k < \omega\}, \\
T_P \downarrow \omega 3 &= T_P \downarrow \omega 2 \setminus \{\mathtt{r}(\mathtt{f}^k(\mathtt{a}))|k < \omega\}, \\
T_P \downarrow \omega 4 &= T_P \downarrow \omega 3 \setminus \{\mathtt{s}(\mathtt{f}^k(\mathtt{a}))|k < \omega\},
\end{aligned}$$

$$\begin{aligned} T_P \downarrow \omega 5 &= T_P \downarrow \omega 4 \setminus \{\mathtt{t}(\mathtt{f}^k(\mathtt{a}))|k < \omega\}, \\ &= \emptyset, \\ &= \mathrm{gfp}(T_P). \end{aligned}$$

Thus $\mathrm{dco}(T_P)$ is $\omega 5$, since that is the least ordinal α such that $T_P \downarrow \alpha = \mathrm{gfp}(T_P)$.

Example 2 Let P be the program

```
p(a) <- p(X) & q(X);
p(f(X)) <- p(X);
q(b);
q(f(X)) <- q(X);
```

Then we have

$$\begin{aligned} T_P \downarrow n &= \{\mathtt{p}(\mathtt{f}^k(\mathtt{a}))|k < \omega\} \cup \{\mathtt{p}(\mathtt{f}^k(\mathtt{b}))|n \leq k < \omega\} \cup \\ &\quad \{\mathtt{q}(\mathtt{f}^k(\mathtt{a}))|n \leq k < \omega\} \cup \{\mathtt{q}(\mathtt{f}^k(\mathtt{b}))|k < \omega\}, \quad \text{for } n < \omega, \\ T_P \downarrow \omega &= \{\mathtt{p}(\mathtt{f}^k(\mathtt{a}))|k < \omega\} \cup \{\mathtt{q}(\mathtt{f}^k(\mathtt{b}))|k < \omega\}, \\ T_P \downarrow (\omega + n) &= \{\mathtt{p}(\mathtt{f}^k(\mathtt{a}))|n \leq k < \omega\} \cup \{\mathtt{q}(\mathtt{f}^k(\mathtt{b}))|k < \omega\}, \quad \text{for } n < \omega, \\ T_P \downarrow \omega 2 &= \{\mathtt{q}(\mathtt{f}^k(\mathtt{b}))|k < \omega\}, \\ &= \mathrm{gfp}(T_P). \end{aligned}$$

The above shows that $\mathrm{dco}(T_P)$ is $\omega 2$.

Example 3 Let P be the program

```
p(a) <- p(X);
p(f(X)) <- p(X);
q(b);
q(f(X)) <- q(X);
```

```
r(c) <- r(X) & q(X);
r(f(X)) <- r(X);
```

Then dco(T_P) is easily shown to be $\omega 2$. Lloyd [15] is really scraping the bottom of the barrel here: if we remove the clauses for the predicate symbol p, which do not affect dco(T_P), then we obtain the program of Example 2, up to renaming of symbols.

4.2 Graphs associated with uniconditional logic programs

We have seen some examples of programs with downward closure ordinal greater than ω. With these in mind one can, with a bit of tinkering, produce more. But that exercise may not make clear what the *mechanism* is; *why* the examples work. In this section we show that all uniconditional logic programs, that is, those where every clause has one condition, can be mapped to a graph in such a way that each vertex has an ordinal number associated with it having the property that the downward closure ordinal of the program has a simple relationship to the orders of the vertices as defined in graph theory. As it is easier to construct graphs in such a way that all successive ordinals up to a certain transfinite bound are associated with a vertex, this result suggests a way to construct *ad libitum* examples of logic programs with downward closure ordinal beyond ω.

Of course, we do not suggest that this be actually done. We present this result because it substantiates our claim that we now *understand* some of the published examples. Better still is to have a parameterized family of logic programs, like the one developed in chapter 2, with a downward closure ordinal closely related to the

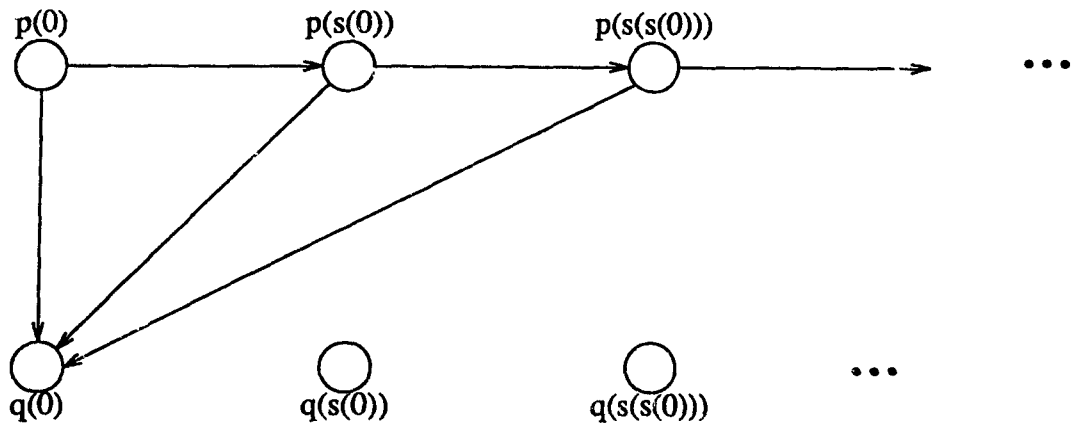

Figure 4.1: Graph of Example 1

parameter. This parameter is given as a term encoding ordinal numbers up to a certain bound. That is the topic of the next two sections.

The graph *associated with* a uniconditional logic program has as vertices elements of the Herbrand base, that is, ground atomic formulas constructed with symbols occurring in the program. There is an arc from `A` to `B` iff `B <- A` is a variable-free instance of a clause in the program. In Figure 4.1 we show the graph associated with one of the examples discussed in the previous section. We will now review known concepts in graph theory that translate directly to the downward closure ordinal of a uniconditional program.

Let $G = \langle V, E \rangle$ be a directed graph, where V is a (possibly infinite) set of vertices and $E \subseteq V \times V$ is a set of edges. The inverse graph of G, denoted G^{-1} is the graph G with all edges reversed, i.e. $G^{-1} = \langle V, E^{-1} \rangle$.

Let $R_G : 2^V \rightarrow 2^V$ be defined as

$$R_G(X) = \{v | \exists u \in X : \langle u, v \rangle \in E\}.$$

We call R_G the *reachability function* of G, because $R_G(X)$ is the set of vertices reachable in one step from a vertex in X. Clearly, R_G is monotonic and we have the following proposition.

Proposition 13 *If G is the graph associated with a uniconditional logic program P, R_G is the immediate-consequence function T_P.*

Definition The *upward ordinal function* X_G for the graph G maps ordinals to sets of vertices of G as follows:

$$\begin{aligned} X_G(0) &= \emptyset, \\ X_G(\alpha) &= \{x | R_G(\{x\}) \subseteq X_G(\alpha - 1)\}, \quad \text{if } \alpha \text{ is a successor ordinal;} \\ &= \bigcup\{X_G(\beta) | \beta < \alpha\}, \quad \text{if } \alpha \text{ is a limit ordinal.} \end{aligned}$$

A similar function was first defined in Berge [5], where it is called "ordinal function". Intuitively, $X_G(\alpha)$ is the set of all vertices v such that all paths in G^{-1} terminating at v are of order type less than α.

We find it useful to define the dual of Berge's upward ordinal function:

Definition The *downward ordinal function* Y_G for the graph $G = \langle V, E \rangle$ is as follows:

$$\begin{aligned} Y_G(0) &= V, \\ Y_G(\alpha) &= R_G(Y_G(\alpha - 1)), \quad \text{if } \alpha \text{ is a successor ordinal;} \\ &= \bigcap\{Y_G(\beta) | \beta < \alpha\}, \quad \text{if } \alpha \text{ is a limit ordinal.} \end{aligned}$$

In other words, $Y_G(\alpha)$ is the set of all vertices which terminate some path in G of order type α or greater. The following propositions are of interest.

Proposition 14 *For all α, $Y_G(\alpha) = V \setminus X_{G^{-1}}(\alpha)$.*

Proof (By transfinite induction) Suppose the result holds for all $\beta < \alpha$. The case for limit values of α is straightforward. However, if α is a successor ordinal, then

$$
\begin{aligned}
& Y_G(\alpha) \\
&\quad = R_G(Y_G(\alpha-1)) \\
&\qquad \text{by definition of } Y_G \\
&\quad = \{x | R_{G^{-1}}(\{x\}) \cap Y_G(\alpha-1) \neq \emptyset\} \\
&\qquad \text{by definition of } G^{-1} \\
&\quad = \{x | R_{G^{-1}}(\{x\}) \cap (V \setminus X_{G^{-1}}(\alpha-1)) \neq \emptyset\} \\
&\qquad \text{by the induction hypothesis} \\
&\quad = V \setminus \{x | R_{G^{-1}}(\{x\}) \subseteq X_{G^{-1}}(\alpha-1)\} \\
&\qquad \text{by simplification} \\
&\quad = V \setminus X_{G^{-1}}(\alpha) \\
&\qquad \text{by definition of } X_G. \square
\end{aligned}
$$

Proposition 15 *Y_G is non-increasing, i.e. if $\alpha \leq \beta$, then $Y_G(\beta) \subseteq Y_G(\alpha)$.*

Proposition 16 *Let $S \subseteq V$ be such that $S \subseteq R_G(S)$. Then $S \subseteq Y_G(\alpha)$, for all α.*

Proof Clearly $S \subseteq Y_G(0)$. Now suppose $S \subseteq Y_G(\beta)$, for all $\beta < \alpha$. If α is a successor ordinal then by the assumption on S and monotonicity of R_G we have $S \subseteq R_G(S) \subseteq R_G(Y_G(\alpha-1)) = Y_G(\alpha)$. If α is a limit ordinal, the result follows from the definition of $Y_G(\alpha)$. $\square$

Corollary If a vertex x occurs in a cycle then $x \in Y_G(\alpha)$, for all α.

Proof For any cycle S in V we have $S \subseteq R_G(S)$. $\square$

Definition The *upward order* of a vertex x is defined in Berge [5] as the smallest ordinal α such that $x \in X_G(\alpha)$, provided that there exists such α.

The above definition is similar to the definition appearing in chapter 2 of upeard orders for ground terms. It may be seen that the definition assigns an order to vertices from which no infinite path originates. Moreover, if it assigns an order α to a vertex then, for all $\beta < \alpha$ it also assigns order β to some vertex.

Before we can develop a notion dual to the upward order, we need the following result:

Proposition 17 *For any $x \in V$, if there is an ordinal α such that $x \notin Y_G(\alpha)$ then there is a greatest ordinal $\beta < \alpha$ such that $x \in Y_G(\beta)$.*

Proof Similar to that of proposition 8. □

Definition The *downward order* of a vertex x is the largest ordinal β such that $x \in Y_G(\beta)$, provided that there is an α such that $x \notin Y_G(\alpha)$.

Proposition 18 *Let $D(x)$ be the downward order of x in G and $U(x)$ be the upward order of x in G^{-1}. Then $U(x) = D(x) + 1$ for every $x \in V$ such that $x \notin Y_G(\alpha)$ for some α.*

Proof Follows from Proposition 14. □

In the remaining part of this dissertation, by *order* we mean the downward order of a vertex. We also denote this value by $Order(x)$, for any vertex x.

For certain graphs, *Order* fails to be a total function; for instance it is not defined for vertices in a cycle since such vertices occur in $Y_G(\alpha)$, for all ordinals α. Moreover, the range of *Order* is an initial segment of the ordinals, i.e. for any ordinal α if there is a vertex $u \in V$ such that $Order(u) = \alpha$, then for every ordinal $\beta < \alpha$ there is a vertex $v \in V$ such that $Order(v) = \beta$.

Definition For any vertex x, x^* is the set of all vertices from which there is a path to x.

Proposition 19 *If all vertices in x^* have an order, then the order of x is the least ordinal greater than all orders of vertices in x^*.*

Definition A graph G is *well founded* if G^{-1} does not contain any infinite paths.

Proposition 20 *If P is a well founded uniconditional program having G as its associated graph, then G is well founded.*

Proposition 21 *For any vertex x of a well founded graph, $Order(x)$ is defined iff there is no vertex y such that y occurs in a cycle and there is a path from y to x.*

Proof ($\Rightarrow$) Suppose such a vertex y exists. Then there is a cycle C containing y and a path P from y to x. Clearly, $C \cup P \subseteq R_G(C \cup P)$. By Proposition 16, $x \in Y_G(\alpha)$, for all α. Thus $Order(x)$ is not defined.

($\Leftarrow$) Suppose $Order(x)$ is not defined. We need to show that there is a $y \in x^*$ such that $y \in y^*$.

Let y_0 be x. By proposition 19, there is a vertex $y_1 \in y_0^*$ such that $Order(y_1)$ is not defined. By iterating the same argument we get an infinite sequence $\langle y_0, y_1, y_2, \cdots \rangle$

of vertices such that y_0 is x and for all i, $Order(y_i)$ is not defined and $y_{i+1} \in y_i^*$. If all the y_i's were distinct, they would constitute an infinite path in G^{-1}, thereby contradicting the well foundedness of G. Therefore, there exist m and n such that $m < n$ and $y_m = y_n$. Hence, $y_m \in y_m^* \subseteq y_0^*$. □

Proposition 22 *If α is any ordinal greater than all orders of vertices in G, then for all $\beta \geq \alpha$, $Y_G(\beta) = Y_G(\alpha)$.*

Proof ($\subseteq$) Since Y_G is monotonically non-increasing, we have that for all $\beta \geq \alpha$, $Y_G(\beta) \subseteq Y_G(\alpha)$.

($\supseteq$) Let $x \notin Y_G(\beta)$, for some $\beta \geq \alpha$. Then by Proposition 17, there is a maximum ordinal δ such that $x \in Y_G(\delta)$. By definition of order, δ is the order of x. As $\delta < \alpha$, by the assumption on α we have that $x \notin Y_G(\alpha)$. Therefore, $Y_G(\alpha) \subseteq Y_G(\beta)$. □

Theorem 4 *If P is a uniconditional program and G is its associated graph, then for all α, $T_P \downarrow \alpha = Y_G(\alpha) = V \setminus X_{G^{-1}}(\alpha)$.*

See Figure 4.2 for an example.

Corollary If G is the graph associated with a uniconditional program P, then $\mathrm{dco}(T_P)$ is the least ordinal greater than all orders of vertices in G.

We feel we have now unveiled the secret of some of the examples found in the literature where the downward closure ordinal is greater than ω. Specifically, here is a method to follow if another example is required. Take any acyclic graph G with at least one vertex of transfinite order, say, α. Name the nodes of G by variable-free atomic formulas. A uniconditional (possibly infinite) program P of

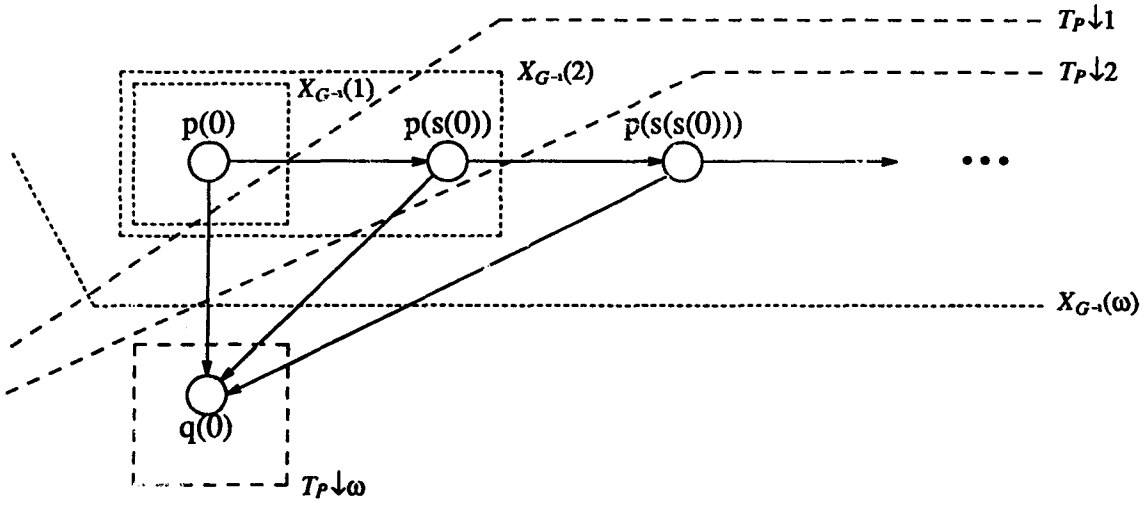

Figure 4.2: Ordinal powers compared with Berge's ordinal function

which G is the associated graph then has a downward closure ordinal greater than α. Of course we choose G and name the nodes in such a way that P has a finite, even a small number, of clauses.

Although we have an improvement over the existing situation, where only a few isolated examples of programs with downward closure ordinal exceeding ω were published, the above "method" is hardly satisfactory. It does not specify how to get from an infinite graph G to a finite, preferably small, logic program having G as associated graph. This problem is addressed in the next section.

4.3 Graph representations

In this section we consider representations of graphs by logic programs. When G is the graph associated with a uniconditional program P, P is similar to a graph representation due to Kowalski [13] that we call *unary representation*. According to it, given a graph G whose vertices are labelled by variable-free terms, the clause

$$\mathtt{r}(\sigma) \leftarrow \mathtt{r}(\tau)$$

is in P iff G has an edge directed from the vertex labelled by τ to the vertex labelled by σ. Note that since there is a one-one correspondence between the edges in G and the clauses in P, P can be infinite. The greatest fixpoint for the unary representation is easily seen to be the empty set and the following proposition follows immediately from the corollary to theorem 4:

Proposition 23 *The downward closure ordinal of the unary representation of a graph is the least ordinal greater than all orders of vertices in the graph.*

Kowalski also uses what we call the *binary representation* of a graph, where a variable-free atom $\mathtt{arc}(\tau.\sigma)$ is interpreted as saying that the graph contains an

edge directed from the vertex labelled by τ to the vertex labelled by σ. A binary representation P of a graph G is an axiomatization in Horn clauses of the `arc` relation. The clauses in P can either be all variable-free or, more interestingly, they may contain variables, in which case P can be finite if the edge set of G is recursive. In its general form, P is a binary representation of a graph when $\mathtt{arc}(\tau, \sigma) \in T_P \uparrow \omega$ iff the graph contains an edge directed from τ to σ.

Unfortunately, we do not have a result equivalent to proposition 23 for a binary representation of a graph. This is due to the fact that, unlike the unary representation, a graph may have many binary representations, which have different downward closure ordinals. On one extreme, if the binary representation contains only variable-free unit clauses for the `arc` predicate, its downward closure ordinal is 1; however on the other extreme, there may exist binary representations with higher, even transfinite, downward closure ordinals.

We find it useful to combine Kowalski's two graph representations. A *combined representation* is obtained by adding the following clauses to a binary representation:

```
K1:  r(X) <- r(Y) & arc(Y,X);
K2:  r(X) <- p(Y);
K3:  p(s(X)) <- p(X);
```

assuming that the predicate symbols `r` and `p` do not occur in the binary representation but the function symbol `s` does. The intuition behind the clause `K1` of the combined representation is that if a vertex Y is reachable and there is an arc from Y to a vertex X, then X is reachable.

The reason for including clauses `K2` and `K3` in a combined representation becomes clear in the proof of theorem 5(b), but for an intuitive understanding, first

consider a combined representation, say R, without these two clauses. Assuming that the corresponding binary representation contains only variable-free unit clauses defining the arc relation, from proposition 23 it may be seen that $\mathrm{dco}(T_R)$ will be $1+\alpha$, where α is the least ordinal greater than all orders of vertices in the graph. In general if the downward closure ordinal of the binary representation is β, $\mathrm{dco}(T_R)$ will be at most $\beta+\alpha$; it will not always be equal to $\beta+\alpha$ because the r-atoms will start disappearing in the sequence $\langle T_R \downarrow \delta\rangle_{\delta\geq 0}$ before all the unwanted arc-atoms have disappeared. To ensure that it is equal to $\beta+\alpha$ we need to retain all the r-atoms just until all the unwanted arc atoms have disappeared. For this reason, we will be particularly interested in determinate binary representations, so that all the unwanted arc-atoms disappear within the first ω steps of the sequence $\langle T_R \downarrow \delta\rangle_{\delta\geq 0}$. Clauses K2 and K3 are added in R to retain all the r-atoms until $T_R \downarrow \omega$. The downward closure ordinal of T_R then turns out to be $\omega+\alpha$.

Definition Let P be a combined representation of a graph. The clauses with the predicate symbols r or p in their heads are *kernel* clauses; all other clauses of P are called *non-kernel.*

Definition If P is a combined representation of a graph, $\hat{P}$ denotes the set of all non-kernel clauses of P.

Definition For any Herbrand interpretation I and predicate symbol p, the p-*component* of I, denoted $I \diamond \mathrm{p}$, is $\{\mathrm{p}(t_1,\ldots,t_n) | \mathrm{p}(t_1,\ldots,t_n) \in I\}$.

Proposition 24 *Let P be a combined representation of a graph. Then*

(a) *for all $\alpha \geq \omega$, $(T_P \downarrow \alpha) \diamond \mathrm{p} = \emptyset$,*

(b) *for all α, $\{T_{\hat{P}} \downarrow \alpha, (T_P \downarrow \alpha) \diamond \mathrm{r}, (T_P \downarrow \alpha) \diamond \mathrm{p}\}$ is a disjoint partition of $T_P \downarrow \alpha$.*

Proof (a) Straightforward as K3 is the only clause in P with the symbol p in its head.

(b) Straightforward since $\hat{P} \subseteq P$, they share the same Herbrand universe U_P and the symbols r and p do not occur in $\hat{P}$. □

Theorem 5 *Let P be a combined representation of a graph $G = \langle V, E \rangle$ such that $\hat{P}$ is determinate. Then*

(a) $(T_P \downarrow \omega) \diamond \mathtt{arc} = \{\mathtt{arc}(\tau,\sigma) | \langle \tau, \sigma \rangle \in E\}$,

(b) $(T_P \downarrow \omega) \diamond \mathtt{r} \supseteq \{\mathtt{r}(\sigma) | \sigma \in V\}$,

(c) *for all* $\alpha > 0$, $(T_P \downarrow (\omega + \alpha)) \diamond \mathtt{r} = \{\mathtt{r}(\sigma) | \sigma \in Y_G(\alpha)\}$.

Proof (a) By proposition 24(b) and the fact that $\hat{P}$ is determinate, we have

$$(T_P \downarrow \omega) \diamond \mathtt{arc} = (T_{\hat{P}} \downarrow \omega) \diamond \mathtt{arc} = (T_{\hat{P}} \uparrow \omega) \diamond \mathtt{arc}.$$

The result follows since $\hat{P}$ is a binary representation of G.

(b) Due to clause K3, for all $n < \omega$, $(T_P \downarrow n) \diamond \mathtt{p} \neq \emptyset$. The result follows due to clause K2.[1]

(c) (By transfinite induction) By theorem 5(b) we have

$$\begin{aligned} (T_P \downarrow \omega) \diamond \mathtt{r} &\supseteq \{\mathtt{r}(\sigma) | \sigma \in V\} \\ &= \{\mathtt{r}(\sigma) | \sigma \in Y_G(0)\}. \end{aligned}$$

Induction hypothesis: For all $\beta < \alpha$,

$$(T_P \downarrow (\omega + \beta)) \diamond \mathtt{r} \supseteq \{\mathtt{r}(\sigma) | \sigma \in Y_G(\beta)\}.$$

[1] In fact, clauses K2 and K3 are included in the combined representation only to ensure that the set $\{\mathtt{r}(\sigma) | \sigma \in V\}$ is contained in $T_P \downarrow \omega$. The importance of this becomes clear in the proof of part (c).

Induction step: If α is successor ordinal then

$(T_P \downarrow (\omega + \alpha)) \diamond \mathbf{r}$

$= (T_P(T_P \downarrow (\omega + \alpha - 1))) \diamond \mathbf{r}$

by definition of $T_P \downarrow (\omega + \alpha)$

$= (T_P(T_{\hat{P}} \downarrow (\omega + \alpha - 1) \cup (T_P \downarrow (\omega + \alpha - 1)) \diamond \mathbf{r} \cup (T_P \downarrow (\omega + \alpha - 1)) \diamond \mathbf{p})) \diamond \mathbf{r}$

by proposition 24(b)

$= (T_P(T_{\hat{P}} \downarrow (\omega + \alpha - 1) \cup (T_P \downarrow (\omega + \alpha - 1)) \diamond \mathbf{r})) \diamond \mathbf{r}$

by proposition 24(a)

$= (T_P(T_{\hat{P}} \downarrow \omega \cup (T_P \downarrow (\omega + \alpha - 1)) \diamond \mathbf{r})) \diamond \mathbf{r}$

by proposition 10

$= \{\mathbf{r}(\sigma) | \exists \tau : \mathbf{arc}(\tau, \sigma) \in T_{\hat{P}} \downarrow \omega \text{ and } \mathbf{r}(\tau) \in (T_P \downarrow (\omega + \alpha - 1)) \diamond \mathbf{r}\}$

by definition of T_P on clause K1, and proposition 5

$= \{\mathbf{r}(\sigma) | \exists \tau : \mathbf{arc}(\tau, \sigma) \in T_{\hat{P}} \downarrow \omega \text{ and } \tau \in Y_G(\alpha - 1)\}$

by induction hypothesis and $\tau \in V$

$= \{\mathbf{r}(\sigma) | \exists \tau : \langle \tau, \sigma \rangle \in E \text{ and } \tau \in Y_G(\alpha - 1)\}$

by proposition 24(b) and theorem 5(a)

$= \{\mathbf{r}(\sigma) | \sigma \in Y_G(\alpha)\}$

by definition of Y_G.

If α is a limit ordinal then

$(T_P \downarrow (\omega + \alpha)) \diamond \mathbf{r}$

$= (\bigcap \{T_P \downarrow \beta | \beta < \omega + \alpha\}) \diamond \mathbf{r}$

by definition of $T_P \downarrow (\omega + \alpha)$

$= (T_P \downarrow \omega \cap \bigcap \{T_P \downarrow (\omega + \beta) | \beta < \alpha\}) \diamond \mathbf{r}$

by simplification

$$= (T_P \downarrow \omega) \diamond \mathbf{r} \cap \bigcap\{(T_P \downarrow (\omega + \beta)) \diamond \mathbf{r} | \beta < \alpha\}$$

by further simplification

$$= (T_P \downarrow \omega) \diamond \mathbf{r} \cap \bigcap\{\{\mathbf{r}(\sigma) | \sigma \in Y_G(\beta)\} | \beta < \alpha\}$$

by induction hypothesis and induction step for successors

$$= (T_P \downarrow \omega) \diamond \mathbf{r} \cap \{\mathbf{r}(\sigma) | \sigma \in Y_G(\alpha)\}$$

by definition of $Y_G(\alpha)$

$$= \{\mathbf{r}(\sigma) | \sigma \in Y_G(\alpha)\}$$

by theorem 5(b).□

Theorem 6 *Let P be a combined representation of a non-empty graph G such that $\hat{P}$ is determinate. Then $dco(T_P) = \omega + \alpha$, where α is the least ordinal greater than all orders of vertices in G.*

Proof ($\leq$) For $\beta \geq \alpha$ we have

$$T_P \downarrow (\omega + \beta + 1)$$

$$= T_{\hat{P}} \downarrow (\omega + \beta + 1) \cup (T_P \downarrow (\omega + \beta + 1)) \diamond \mathbf{r} \cup (T_P \downarrow (\omega + \beta + 1)) \diamond \mathbf{p}$$

by proposition 24(b)

$$= T_{\hat{P}} \downarrow (\omega + \beta + 1) \cup (T_P \downarrow (\omega + \beta + 1)) \diamond \mathbf{r}$$

by proposition 24(a)

$$= T_{\hat{P}} \downarrow (\omega + \beta) \cup (T_P \downarrow (\omega + \beta + 1)) \diamond \mathbf{r}$$

by proposition 10

$$= T_{\hat{P}} \downarrow (\omega + \beta) \cup \{\mathbf{r}(\sigma) | \sigma \in Y_G(\beta + 1)\}$$

by theorem 5(c)

$$= T_{\hat{P}} \downarrow (\omega + \beta) \cup \{\mathbf{r}(\sigma) | \sigma \in Y_G(\beta)\}$$

by proposition 22

$$= T_{\hat{P}} \downarrow (\omega + \beta) \cup (T_P \downarrow (\omega + \beta)) \diamond \mathbf{r}$$

by theorem 5(c)

$$= T_P \downarrow (\omega + \beta)$$

by proposition 24.

Thus, $\mathrm{dco}(T_P) \leq \omega + \alpha$.

($\geq$) By the condition on α, for all $\beta < \alpha$, there is a vertex x in G such that $x \in Y_G(\beta)$ and $x \notin Y_G(\alpha)$. By theorem 5(c), $\mathbf{r}(x) \in T_P \downarrow (\omega + \beta)$ but $\mathbf{r}(x) \notin T_P \downarrow (\omega + \alpha)$. Hence, $\mathrm{dco}(T_P) \geq \omega + \alpha$. □

4.3.1 An example of combined representation

Consider any graph containing exactly one vertex of order α, for each ordinal $\alpha < \epsilon_0$. It is easily seen that all such graphs have the same transitive closure, denoted by W, which has the property that an edge directed from vertex u to v exists in W iff $Order(u) < Order(v)$. In this section we construct a combined representation of W.

Any combined representation of W will contain the kernel clauses:

```
K1:  r(X) <- r(Y) & arc(Y,X);
K2:  r(X) <- p(Y);
K3:  p(s(X)) <- p(X);
```

along with the non-kernel clauses axiomatizing the `arc` relation, which depend upon the structure of W.

Since W contains exactly one vertex of each order less than ϵ_0, we can represent vertices by their orders using the notation system developed in chapter 2.

W contains an edge from vertex u to vertex v iff $Order(u) < Order(v)$. This gives rise to the following Horn clause D1 for the arc relation:

```
D1:  arc(X,Y) <- ord(X) & ord(Y) & lto(X,Y);
```

where the ord and lto predicates were introduced in chapter 2. We use clauses C1-C11 developed therein for these predicates.

Thus the combined representation of W contains the kernel clauses K1-K3, clause D1 for arc and clauses C1-C11 for ord and lto.

4.4 A family of logic programs

In this section we construct a family $\{P_\alpha\}_{\alpha<\epsilon_0}$ of logic programs, such that dco(T_{P_α}) is $\omega + \alpha$. The members of this family are combined representations of graphs containing vertices with transfinite order.

As a basis for this family we consider the graph W introduced in the previous section, which contains exactly one vertex of each order less than ϵ_0 and has the property that from any vertex there are edges to all vertices with higher orders. Let W_α be the graph obtained by adding, in W, an edge from the vertex with order α to itself. This extra edge introduces a cycle containing only that vertex, thereby causing that vertex, and vertices which in W had a higher order, not to have an order. Then the program P_α is the combined representation of W_α.

Thus in addition to the clauses K1-K3, D1 and C1-C11 for the combined representation of W, P_α contains the non-kernel clause

$$\texttt{D2 : arc}(\lceil\alpha\rceil, \lceil\alpha\rceil);$$

Theorem 7 *For $\alpha > 0$, $dco(T_{P_\alpha}) = \omega + \alpha$.*

Proof $\hat{P}_\alpha$ $(= \{\texttt{D1}, \texttt{D2}, \texttt{C1}, \ldots, \texttt{C11}\})$ can be verified to meet the conditions of proposition 12. Thus by proposition 11, it is determinate. Proposition 21 tells us that a vertex has an order in W_α iff the corresponding vertex has an order $< \alpha$ in W. So α is the least ordinal greater than all orders of vertices in W_α. The result follows from theorem 6. □

Chapter 5

Downward Closure Ordinal Based Transformations

After establishing a relationship between the syntax and semantics of formal systems in general, and logic programs in particular, one is naturally led to the issue of studying the relationship between transformations on the syntax objects and changes to their semantic counterparts. In principle, this study can be carried out in two opposite directions. For a given syntactic transformation it is often useful to determine its effects on the semantics of the transformed objects. Alternatively, it is more practical to start with an intended semantic change and determine whether or not a corresponding transformation on the syntax objects exists for it; moreover, if such a transformation exists, finding one is almost always necessary. We start with simple examples of both of these approaches.

For an example of the first approach, let $\mathcal{P}$ be the set of all logic programs in some fixed language. $\mathcal{P}$ is our set of syntax objects. Since all programs in $\mathcal{P}$ are based on the same language, they all have the same Herbrand Base B. Let $\mathcal{T}$ be the set of all functions with 2^B as their domain and range. $\mathcal{T}$ is our set of semantic

objects. As usual, for any program $P \in \mathcal{P}$. let $T_P \in \mathcal{T}$ be the immediate consequence function of P. In [16], Mancarella and Pedreschi have studied the effects of certain syntactic operations on elements of $\mathcal{P}$ on their immediate consequence functions. The simplest of these is the union operation $\oplus$ defined as follows: for any $P_1, P_2 \in \mathcal{P}$,

$$P_1 \oplus P_2 \stackrel{\text{def}}{=} P_1 \cup P_2$$

where the $\cup$ operation, as expected, treats P_1 and P_2 as sets of clauses. Mancarella and Pedreschi then proceed to show that

$$T_{P_1 \oplus P_2} = \lambda I. T_{P_1}(I) \cup T_{P_2}(I).$$

In other words, the immediate consequences of any interpretation $I \subseteq B$ from the program $P_1 \oplus P_2$ are exactly those obtainable from either P_1 or P_2. This can be seen to follow immediately from the definitions of the immediate consequence function and the $\oplus$ transformation.

For an example of arriving at a syntactic transformation corresponding to a desired semantic change, let $\mathcal{E}$ be the set of expressions over natural numbers involving the infix addition operator + and $\mathcal{N}$ be the set of natural numbers. Let $I : \mathcal{E} \rightarrow \mathcal{N}$ be an interpretation map that assigns values to expressions according to the standard laws of arithmetic. Clearly, I is many-to-one, as for example, the expressions "6+1" and "2+3+2" in $\mathcal{E}$ are mapped to the same object 7 in $\mathcal{N}$. Now let $S : \mathcal{N} \rightarrow \mathcal{N}$ be the successor function that maps every number in $\mathcal{N}$ to the next higher number. A transformation $T : \mathcal{E} \rightarrow \mathcal{E}$ *corresponds* to S if for any expression $e \in \mathcal{E}$,

$$I(T(e)) = S(I(e)),$$

i.e. T maps any expression in $\mathcal{E}$ to an expression with one higher value. It is easily seen that many transformations correspond to S. An example is,

$$T_1(e) \stackrel{\text{def}}{=} \text{“1 + ”} \bowtie e,$$

where $\bowtie$ is the string concatenation operator. Another transformation

$$T_2(e) \stackrel{\text{def}}{=} e \bowtie \text{“}+1\text{”}$$

can also be seen to have the desired property. Yet another transformation is

$$T_3(e) \stackrel{\text{def}}{=} \begin{cases} \text{“}4+4\text{”} & \text{if } e = \text{“}4+3\text{”} \\ T_2(e) & \text{otherwise} \end{cases}$$

which differs from T_2 only for the argument “4+3”. Similarly, we can construct many more transformations corresponding to S. We will see later that the number of such transformations is directly related to the degree of many-to-one-ness of the interpretation I.

5.1 Transformation Systems

The above example of arithmetic expressions is a particular instance of general transformation systems involving a syntax and a semantic domain with an interpretation map associating their elements. The objective is to find transformations that correspond to the desired changes in the semantics. In this section we study the conditions under which such transformations exist.

Let S and V be, respectively, the sets of syntax objects and their semantic values. Let $I : S \rightarrow V$ be an interpretation map and $C : V \rightarrow V$ be some function. Given an object $a \in S$, we are interested in determining an object $b \in S$, such that $I(b) = C(I(a))$. In figure 5.1 it can be seen that if such an object b exists, then $b \in I^{-1}(C(I(a)))$. If I is many-to-one, in general $I^{-1}(C(I(a)))$ will contain more than one element. A transformation $T : S \rightarrow S$ will correspond to C if for all $a \in S$, $T(a) \in I^{-1}(C(I(a)))$.

Proposition 25 *A transformation T that corresponds to C exists iff*

$$C(Range(I)) \subseteq Range(I).$$

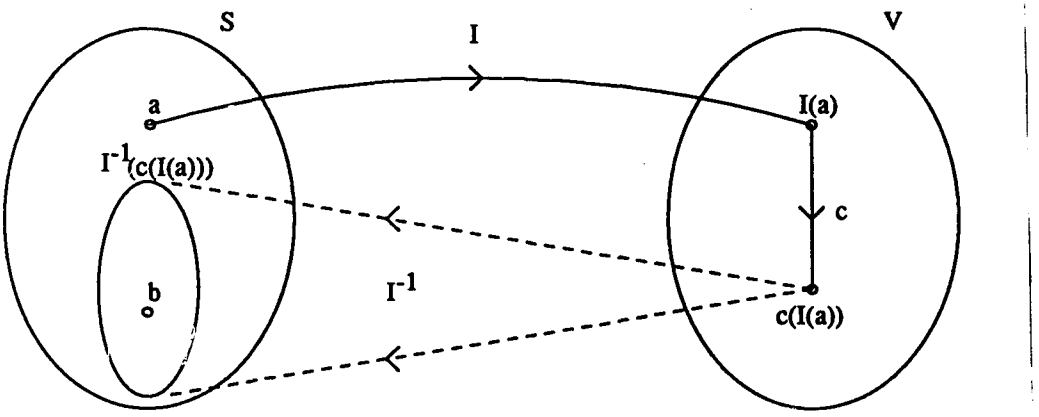

Figure 5.1: A transformation system

Proof ($\Rightarrow$) Suppose such a transformation T exists. Then for any $a \in S$, $T(a) \in I^{-1}(C(I(a)))$. Thus the set $I^{-1}(C(I(a)))$ is nonempty, i.e. $C(I(a)) \in \text{Range}(I)$. Since a was arbitrarily chosen, we have that $C(\text{Range}(I)) \subseteq \text{Range}(I)$.

($\Leftarrow$) Suppose $C(\text{Range}(I)) \subseteq \text{Range}(I)$. Let a be any element in S. Then $C(I(a)) \in \text{Range}(I)$, i.e. $I^{-1}(C(I(a)))$ is nonempty. The existence of a transformation corresponding to C follows from the Axiom of Choice. □

Having established this result for unary maps T and C, we can easily generalize it to n-ary maps:

Proposition 26 *Let sets S, V and functions $I : S \to V$ and $C : V^n \to V$ be given. Then a transformation $T : S^n \to S$ such that for all sequences $\langle a_1, a_2, \ldots, a_n \rangle \in S^n$,*

$$T(\langle a_1, a_2, \ldots, a_n \rangle) \in I^{-1}(C(\langle I(a_1), I(a_2), \ldots, I(a_n) \rangle))$$

exists iff $C((Range(I))^n) \subseteq Range(I)$.

Proof Similar to that of Proposition 25. □

5.2 Program transformations

In this section we attempt to discover transformations on logic programs in the general framework described in the previous section. Instead of studying their immediate consequence functions we will only be interested in their downward closure ordinals. Thus our semantic objects will be the ordinals that are candidates for being the downward closure ordinals of programs.

Let $\mathcal{P}$ be the set of all logic programs and $\mathcal{W}$ be the set of all ordinals up to and including the least nonrecursive ordinal ω_1^{ck}. Blair showed in [7] that the downward closure ordinal of any program is at most ω_1^{ck}, and also that for any ordinal up to and including ω_1^{ck}, there is a program with that ordinal as its downward closure ordinal. Let dco: $\mathcal{P} \rightarrow \mathcal{W}$ be the downward closure map for programs. As more than one program can have the same downward closure ordinal, the dco function is many-to-one. Moreover, from Blair's result, we have that dco is onto $\mathcal{W}$.

Proposition 27 *For any operation $S : \mathcal{W}^n \rightarrow \mathcal{W}$, there exists a corresponding transformation $T : \mathcal{P}^n \rightarrow \mathcal{P}$.*

Proof Since dco is onto $\mathcal{W}$, it meets the condition of Proposition 26. □

From the above Proposition we know that for any operation on our ordinal set $\mathcal{W}$, there exists a corresponding transformation on our program set $\mathcal{P}$. We now attempt to discover such transformations for some operations.

Identity

We start with the identity map S on $\mathcal{W}$, i.e. for all $\alpha \in \mathcal{W}$, $S(\alpha) = \alpha$. While any dco preserving transformation would do, clearly the identity transformation T, for which for any $P \in \mathcal{P}$, $T(P) = P$, is the simplest one suitable for this S.

Successor

Now let us consider the first interesting case by defining $S(\alpha) = \alpha + 1$, for all $\alpha \in \mathcal{W}$. So we need to come up with a transformation T, such that for any $P \in \mathcal{P}$, $\mathrm{dco}(T(P)) = \mathrm{dco}(P) + 1$.

Our approach in constructing the transformation T is based on the observation that $\mathrm{dco}(P)$ is directly governed by the ground atoms in the language of P that have a downward order. We let the Herbrand Base $B_{T(P)}$ of $T(P)$ have a unique extra atom A' for every A in B_P such that if A has a downward order then the downward order of A' is one more than that of A. Such a construction works only for programs P for which $\mathrm{dco}(P)$ is a successor ordinal, but due to its simplicity we will be content with it and leave the construction of a completely general transformation as future work.

For any program $P \in \mathcal{P}$, let $T(P)$ be the program obtained by adding to P the clause

$$q'(X_1, X_2, \ldots, X_n) \leftarrow q(X_1, X_2, \ldots, X_n)$$

for each n-ary predicate symbol q in P. We assume that the symbol q' does not occur in P and that variables X_1 through X_n are distinct.

Proposition 28 $dco(T(P)) = \begin{cases} dco(P) + 1 & \textit{if } dco(P) \textit{ is a successor ordinal} \\ dco(P) & \textit{otherwise} \end{cases}$

Proof *(Case 1)* Suppose $\mathrm{dco}(P)$ is a successor ordinal. Then there is a ground atom A such that

(a) $A \in T_P \downarrow \alpha$, for all $0 \leq \alpha < \mathrm{dco}(P)$, and

(b) $A \notin T_P \downarrow \alpha$, for all $\alpha \geq \mathrm{dco}(P)$.

Let S be the set of ground atoms with the above two properties. Consider any $A \in S$ of the form $q(t_1, t_2, \ldots, t_n)$, and let A' be $q'(t_1, t_2, \ldots, t_n)$. Due to the extra clause for q' present in P', we have

(a) $A' \in T_{T(P)} \downarrow \alpha$, for all $0 \leq \alpha \leq \mathrm{dco}(P)$, and

(b) $A' \notin T_{T(P)} \downarrow \alpha$, for all $\alpha > \mathrm{dco}(P)$.

Moreover, it is easily seen that for all $\alpha > \mathrm{dco}(P) + 1$,

$$T_{T(P)} \downarrow \alpha = T_{T(P)} \downarrow (\mathrm{dco}(P) + 1).$$

Hence $\mathrm{dco}(T(P)) = \mathrm{dco}(P) + 1$.

(Case 2) Suppose $\mathrm{dco}(P)$ is a limit ordinal. Then for any ground atom A of the form $q(t_1, t_2, \ldots, t_n)$ such that $A \notin T_P \downarrow \mathrm{dco}(P)$, there is an $\alpha < \mathrm{dco}(P)$ such that $A \in T_P \downarrow \alpha$ and for some $\alpha < \beta < \mathrm{dco}(P)$, $A \notin T_P \downarrow \beta$. Thus the ground atom A' of the form $q'(t_1, t_2, \ldots, t_n)$ is such that $A' \in T_{T(P)} \downarrow \alpha$ and $A' \notin T_{T(P)} \downarrow (\beta + 1)$. Thus $A \notin T_{T(P)} \downarrow \mathrm{dco}(P)$. Therefore $\mathrm{dco}(T(P)) = \mathrm{dco}(P)$. □

As an example, let be the program

```
p(f(X)) <- p(X)
q(a) <- p(X)
```

that first appeared in chapter 1. It was seen that $\mathrm{dco}(P)$ is $\omega + 1$. After applying the above transformation on P, we get $T(P)$ to be the program

```
p(f(X)) <- p(X)
q(a) <- p(X)
p'(X) <- p(X)
q'(X) <- q(X)
```

It may be verified that for $1 < n < \omega$,

$$T_{T(P)} \downarrow n = \{\mathtt{p}(\mathtt{f}^k(\mathtt{a})) \mid n \leq k < \omega\} \cup \{\mathtt{q}(\mathtt{a})\} \cup \{\mathtt{p}'(\mathtt{f}^k(\mathtt{a})) \mid n-1 \leq k < \omega\} \cup \{\mathtt{q}'(\mathtt{a})\}.$$

And

$$\begin{aligned} T_{T(P)} \downarrow \omega &= \{\mathtt{q}(\mathtt{a})\} \cup \{\mathtt{q}'(\mathtt{a})\} \\ T_{T(P)} \downarrow \omega + 1 &= \{\mathtt{q}'(\mathtt{a})\} \\ T_{T(P)} \downarrow \omega \dot{+} 2 &= \emptyset \end{aligned}$$

Thus $\mathrm{dco}(T(P)) = \omega + 2 = \mathrm{dco}(P) + 1$.

Addition

Let us now consider the binary map $S : \mathcal{W}^2 \rightarrow \mathcal{W}$ for the addition operation on ordinals. We will develop a corresponding transformation, which for any two programs P and Q with disjoint languages would construct a program R such that $\mathrm{dco}(R) = \mathrm{dco}(P) + \mathrm{dco}(Q)$.

The program R is constructed from P and Q in a way that its Herbrand Base B_R is isomorphic to the set $B_P \cup (B_P \times B_Q)$. Ground atoms $A \in B_P$ that have a downward order in P still have the same downward order in the program R.

For each such atom A with downward order α, and any atom $A' \in B_Q$ that has a downward order β in Q, the tuple $\langle A, A' \rangle$ has downward order $\alpha + \beta$ in R. Due to all such atoms, $\mathrm{dco}(R)$ becomes $\mathrm{dco}(P) + \mathrm{dco}(Q)$.

We now define the construction of R. Let it be the program obtained by adding to P the following sets of clauses:

(1) For every k-ary predicate symbol p in P and n-ary predicate symbol q in Q, the clause

$$q(p'(X_1, X_2, \ldots, X_k), Y_1, Y_2, \ldots, Y_n) \leftarrow p(X_1, X_2, \ldots, X_k)$$

where p' is a new function symbol and the X_i's and Y_i's are distinct variables.

(2) For every k-ary predicate symbol p in P and the clause

$$q_0(t_0^1, t_0^2, \ldots, t_0^{n_0}) \leftarrow$$
$$q_1(t_1^1, t_1^2, \ldots, t_1^{n_1}), \ldots, q_m(t_m^1, t_m^2, \ldots, t_m^{n_m})$$

in Q, the clause

$$q_0(p'(X_1, X_2, \ldots, X_k), t_0^1, t_0^2, \ldots, t_0^{n_0}) \leftarrow$$
$$q_1(p'(X_1, X_2, \ldots, X_k), t_1^1, t_1^2, \ldots, t_1^{n_1}),$$
$$\vdots$$
$$q_m(p'(X_1, X_2, \ldots, X_k), t_m^1, t_m^2, \ldots, t_m^{n_m})$$

where p' is a new function symbol and the X_i's are distinct variables.

Theorem 8 *If $dco(P)$ is a successor ordinal, $dco(R) = dco(P) + dco(Q)$.*

Proof Let $\mathrm{dco}(P)$ be a successor ordinal $\alpha+1$. Then α is the maximum ordinal such that there are ground atoms $A \in B_P$ with downward order α. As $P \subseteq R$, any such $A \in B_R$ also has downward order α. Let any such A be of the form $p(t_1,\ldots,t_k)$. Due to clauses of the first kind, we have that

$$\{q(p'(t_1,\ldots,t_k),u_1,\ldots,u_n)|q(u_1,\ldots,u_n) \in B_Q\} \subseteq T_R \downarrow (\alpha+1).$$

Due to clauses of second kind, $\mathrm{dco}(R)$ is the least ordinal greater than the elements of the set

$$\{DO(q(p'(t_1,\ldots,t_k),u_1,\ldots,u_n))|q(p'(t_1,\ldots,t_k),u_1,\ldots,u_n) \in \overline{\mathrm{gfp}(T_R)}\}$$

which is the least ordinal greater than the elements of the set

$$\{(\alpha+1)+DO(q(u_1,\ldots,u_n))|q(u_1,\ldots,u_n) \in \overline{\mathrm{gfp}(T_P)}\}$$

or in other words $\mathrm{dco}(P)+\mathrm{dco}(Q)$. □

Similar to the successor transformation, the above transformation works only if $\mathrm{dco}(P)$ is a successor ordinal. As before, due to its simplicity we are content with it and leave the construction of a completely general transformation as future work.

5.3 General transformations

Having found transformations corresponding to the successor and addition operations on ordinals it is quite tempting to find transformations corresponding to other operations like multiplication, exponentiation etc.. However, such an *ad hoc* approach will be limited to only the operations considered and it would be much more significant to develop a general transformation construction scheme for the operations described below.

The ordinal $\alpha + \beta$ can, in a sense, be obtained by taking the successor of α, β times; $\alpha\beta$ can be obtained by adding α to itself, β times; analogously, α^β can be obtained by multiplying α to itself, β times; and so on. This is made precise by the following sequence of operations:

$$
\begin{aligned}
f_0(\alpha,\beta) = \quad \alpha+\beta \quad &= \underbrace{s(s(\cdots s(}_{\beta \text{ times}}\alpha)\cdots)) \\
f_1(\alpha,\beta) = \quad \alpha\beta \quad &= \underbrace{f_0(f_0(\cdots f_0(}_{\beta \text{ times}}\alpha,\alpha)\cdots,\alpha),\alpha) \\
f_2(\alpha,\beta) = \quad \alpha^\beta \quad &= \underbrace{f_1(f_1(\cdots f_1(}_{\beta \text{ times}}\alpha,\alpha)\cdots,\alpha),\alpha) \\
&\vdots
\end{aligned}
$$

where s denotes the successor operation. For any $n > 0$, $f_n(\alpha,\beta)$ is obtained by β applications of f_{n-1} on α and itself.

A general transformation scheme would be able to find a transformation T_n corresponding to any function f_n. An even more ambitious project would be to develop a transformation meta-scheme, which would find a transformation scheme for any given unary seed function f_0, but for now we leave general transformations as directions for future work.

Chapter 6

Conclusions

The main emphasis of the work described in this dissertation has been on studying certain ordinal-theoretic properties of logic programs. In doing so, we associated ordinals with logic programs in two independent ways: first, we considered programs that axiomatize well orderings and, second, we looked at the downward closure ordinals of logic programs.

Given Blair's result that for every recursive ordinal α, there is a program that axiomatizes a well ordering of type α, chapter 2 constructs a family of such programs indexed by ordinals up to ϵ_0. Its main contributions are a convenient system of notations involving ground terms for such ordinals and a Horn clause axiomatization of the less-than relation on ordinals smaller than ϵ_0. This chapter also indicates a method to expand this system of notations for larger ordinals, which would lead to the construction of larger families of such programs. However, it is shown that no such expansion of the system of notations would lead to a maximal family of programs that would be indexed by all ordinals up to ω_1^{ck}. Though the existence of such a maximal family of programs is guaranteed by Blair's result, at present we do not know if one can be constructed. Search for such a family is one

of the directions for future work indicated by this chapter.

Just as any ground atom A contained in $\mathrm{lfp}(T_P)$, for any program P, has an upward order that is a measure of the minimum amount of work needed to determine that A is a logical consequence of P, any ground atom A contained in $\overline{\mathrm{gfp}(T_P)}$ has a downward order that is a measure of the minimum amount of work needed to determine that A is *not* a logical consequence of P. In chapter 3 we see that unlike upward orders, downward orders need not necessarily be finite. In general, it is not always possible to determine the downward order of atoms in $\overline{\mathrm{gfp}(T_P)}$. This chapter presents some rules for achieving this for a restricted class of programs by studying the syntactic structure of its clauses. The rules are given for a few simple kinds of clause structures, thus leaving a significant scope for expansion for other kinds of clauses. However, the chapter shows the non-existence of any set of rules that would work for all programs.

Chapter 4 contains some of the most interesting results of this dissertation. It presents a family of logic programs indexed by ordinals up to ϵ_0, such that the downward closure ordinal of any of its member P_α is $\omega + \alpha$. The family is based on the one constructed in chapter 2 and provides a much needed bridge for the gap between the meagreness of the concrete examples contained in the literature and the richness of Blair's result about the existence of such programs up to ω_1^{ck}. This chapter establishes a two-way connection between graphs and logic programs by giving ways to represent one using the other. The connection provides for a better understanding of logic programs by making use of known concepts in graph theory.

Transformation on syntax objects is a topic of a very general nature that can be applied to both arithmetic and non-arithmetic expressions, programs and many other syntax domains. Chapter 5 studies transformation systems in which the objective is to search for transformations that satisfy given semantic constraints. It

establishes necessary and sufficient conditions under which desired transformations exist in a system. This makes it possible to know beforehand whether a search has a possibility of being successful. Within this framework, transformations are constructed on logic programs for the successor and addition operations on their dow .ward closure ordinals. However, these transformations are constructed in an *ad hoc* manner. It is much more desirable to construct a general transformation scheme for all the operations in the Ackermann hierarchy mentioned toward the end of the chapter. This is one of the main directions for future work indicated by this chapter.

Bibliography

[1] H. Andreka and I. Nemeti. The generalized completeness of horn predicate logic as a programming language. *Acta Cybernetica*, 4, 1978.

[2] K. R. Apt. *Introduction to Logic Programming.* Technical Report TR-87-35, Department of Computer Sciences, University of Texas at Austin, July 1988.

[3] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, July 1982.

[4] R. Bagai, M. Bezem, and M. H. van Emden. On downward closure ordinals of logic programs. *Fundamenta Informaticae*, March 1990.

[5] C. Berge. *The Theory of Graphs.* John Wiley and Sons, 1962.

[6] H. A. Blair. Decidability in the herbrand base. manuscript (presented at the Workshop on Foundations of Deductive Databases and Logic Programming, Washington, D.C., August 1986).

[7] H. A. Blair. The recursion-theoretic complexity of the semantics of predicate logic as a programming language. *Information and Control*, July–September 1982.

[8] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, Plenum Press, NY, 1978.

[9] W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, 1984.

[10] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[11] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and Lambda-Calculus.* Cambridge University Press, 1986.

[12] J. Jaffar and P. J. Stuckey. Canonical logic programs. *Journal of Logic Programming*, 2, 1986.

[13] R. Kowalski. *Logic for Problem Solving.* North-Holland, 1979.

[14] R. A. Kowalski. Predicate logic as a programming language. *Information Processing*, 1974.

[15] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[16] P. Mancarella and D. Pedreschi. An algebra of logic programs. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, 1988.

[17] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, January 1965.

[18] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* MIT Press, 1987.

[19] J. C. Shepherdson. Negation in logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, 1988.

[20] W. Sierpinski. *Cardinal and Ordinal Numbers.* Polish Scientific Publishers, 1965.

[21] A. Urquhart. The undecidability of entailment and relevant implication. *The Journal of Symbolic Logic*, December 1984.

[22] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, October 1976.