ACCEPTED
FACULTY OF GRADUATE STUDIES

DEAN

DATE 91/09/11

Intensional Logic Programming

by

Mehmet Ali Orgun

B.Sc., Hacettepe University, 1982
M.Sc., Hacettepe University, 1985

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

Dr. W. W. Wadge, Supervisor (Department of Computer Science)

Dr. M. H. van Emden, Departmental Member (Department of Computer Science)

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

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

Dr. L. Robertson, Outside Member (Department of Physics)

Dr. A. Borning, External Examiner (Department of Computer Science)

Supervisor: Dr. William W. Wadge

Abstract

This dissertation presents an investigation of logic programming based on intensional logic. Through intensional logic, the notion of dynamic change and the ability to reason about context-dependent properties can be brought back into logic programming without any extra-logical or non-logical features. In intensional logic, the meaning of an expression depends on an implicit context. Temporal logic is a special case of intensional logic where the set of contexts models a collection of moments in time. Intensional logic programs are a set of logical axioms interpreted as statements true at all contexts.

We can investigate the meaning of programs written in an intensional language either by focusing on the language or by studying the general properties of intensional logic programming to identify the conditions under which those properties are satisfied. This dissertation discusses both approaches with more emphasis on the more general one. The temporal language Chronolog is an instance of intensional logic programming, suitable for modeling time-varying aspects of certain problems and non-terminating computations. The semantics of Chronolog programs are developed in terms of temporal Herbrand interpretations.

The dissertation introduces an intensional semantics based on Scott's neighborhood semantics. We identify several important semantic properties of intensional operators, namely, monotonicity, universality, conjunctivity and finitariness. An intensional logic program logic enjoys the minimum model semantics and its fixpoint characterization provided that intensional operators of the underlying logic have these properties. We show that the theory can be applied to existing logic programming languages based on diverse temporal logics, modal logic and interval logic. The

theory can be utilized to design an intensional logic programming language with the desired properties.

Due to non-determinism involved in logic programming, predicates do not represent single-valued relations. Choice predicates are an extension of intensional logic programming, through which non-deterministic dataflow-style of computations can be modeled. A choice predicate in principle acts like a dataflow node with multiple input lines, which arbitrarily selects one of its inputs as output. We provide the semantics of intensional logic programs with choice predicates in terms of minimal models. We also investigate how the expressiveness of intensional logic programming can be improved.

Examiners:

Dr. W. W. Wadge, Supervisor (Department of Computer Science)

Dr. M. H. van Emden, Departmental Member (Department of Computer Science)

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

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

Dr. L. Robertson, Outside Member (Department of Physics)

Dr. A. Borning, External Examiner (Department of Computer Science)

Contents

List of Figures

Acknowledgements

I would like to thank my supervisor and mentor, Professor Bill Wadge, who has been a continuing source of inspiration and encouragement during the development of this research. He introduced me to the wonderful world of intensional logic. Many thanks go to the members of the functional programming group, especially Weichang Du, Gordon Stuart and Senhua Tao, for many brain-storming discussions we have had together. I am also indebted to my supervisory committee members for their careful reading of my dissertation. I was financially supported by a University of Victoria Graduate Fellowship during my studies. I am also grateful to the Computer Science Department of the University of Victoria, whose facilities were used in the preparation of this dissertation.

My stay in Victoria has been a very pleasant one. I would like to thank to my friends in Victoria for providing instant relief from daily grind. Without them, this dissertation would have been completed much sooner. I hope our roads will cross again.

My deepest thanks go to my parents for their ever-lasting love, support and encouragement, since I started school so many years ago.

Chapter 1

Introduction

1.1 Problems with Imperative Languages

Since almost all of today's computer architectures are based on concepts invented by von Neumann and others in the mid 40's, the quest for better programming languages has produced a variety of imperative languages, all of which reflect the very same concepts at different abstraction levels. However, imperative languages have failed to give programmers the ability to reason about their programs in a mathematical way, because they lack simple axioms and rules of inference required for verification. Therefore imperative languages are not logical formalisms, rather a syntactic representation of operational aspects of the underlying architectures.

This lack of formalism spawned another research area, verifying the correctness of programs culminating from the early works of Floyd [Flo67] and Hoare [Hoa69]. First-order (predicate) logic has played an important role in this field, and more recently non-classical logical formalisms have been proposed for verifying concurrent programs, including temporal logic [MP81], modal logic [FL77] and algorithmic and dynamic logic [Pra80]. For a thorough account of the developments in program verification and other approaches to formalizing programming, we refer the reader

to the literature, e.g., see [Man74], [MP81], [Pra80], and [Har84].

In the mean time, developments in VLSI technology have produced novel multiprocessor architectures with high-speed parallel computation potential. Another defect of imperative languages is that they are inherently sequential, and therefore cannot effectively exploit parallelism offered by those new architectures. In an imperative language, programmers have to specify what parts of a program can be performed in parallel, and how concurrent parts can communicate with each other. In other words, it is the programmers' responsibility to discover parallelism.

All of these problems can be attributed to the concepts imperative languages are based upon: A program written in an imperative language specifies, step by step, how a computation is to be performed, not what is to be computed; program constructs basically correspond to state transformations in the underlying von Neumann architecture.

Declarative languages are proposed as one solution to the problems of von Neumann (imperative) languages. Declarative languages, as opposed to imperative languages, are logical formalisms with simple denotational semantics; they are not based on any specific architecture and enable programmers to specify what is to be computed. Declarative languages do not have side-effects and are insensitive to the ordering of programming constructs during execution, except the ordering imposed by the data-dependencies in programs. Therefore in principle they are more amenable to parallel implementations than imperative languages.

Declarative languages can be broadly grouped under two categories:

- Functional languages,
- Logic programming languages.

In functional languages, the basic building blocks are functions. By combining functions through the usual mathematical method of function composition, more complex program constructs can be formed.

1.2 Logic Programming

Since our main interest is logic programming languages, we will provide a more detailed but informal treatment of logic programming. Logic is a formal system concerned with two concepts: truth and provability. Given a set of logical axioms, we are interested in what follows from it or what its logical consequences are, and how they can be deduced from the given set by the rules of inference provided in the logic. Gödel's completeness theorem forms the bridge between the notion of logical consequence, which refers to truth, and the notion of proof, which is purely syntactical. Truth is defined in terms of interpretations and structures, while provability is defined in terms of rules of inference and proof systems.

The goal of mechanical theorem proving is to find algorithmic methods for finding proofs of logical statements from a given set of logical axioms. However, Church's undecidability theorem asserts that there is no such algorithmic method that always terminates reporting either success or failure. Regardless of theoretical limitations imposed by Church's theorem and the combinatorial nature of proofs, mechanical theorem-proving has flourished after Robinson's discovery of unification and resolution inference rule for clausal logic [Rob65]. Several simplifications of Robinson's resolution rule have been proposed in order to reduce the complexity of mechanical proofs even further [CL73].

Then Colmerauer [C+73] and Kowalski [Kow74] suggested the use of logic as a programming language, rather than employing it as a formal system in the background. In logic programming, a program is considered as a set of logical axioms formalising our knowledge and assumptions about some problem in the form of a set of Horn clauses. Then the problem statement is also formalised in logic as a goal statement (query) to be proved from the given set of axioms by using a proof procedure called SLD-resolution. The most widely used logic programming language Prolog is based on Horn logic and was implemented in the early 70's by Colmerauer

and his colleagues [BM73].

We will illustrate the basic ideas behind logic programming. The following logic program specifies the addition relation add with successor functions by a set of Horn clauses:

```
add(X,0,X).
add(X,s(Y),s(Z)) <- add(X,Y,Z).
```

Here all variables in each program clause are assumed to be universally quantified. Program constructs such as `add(X,0,X)` are called atoms. The first clause unconditionally says that the result of adding `0` to any number is the same number. The second clause is a conditional assertion. The left-hand side of a conditional statement is called the conclusion of the statement, and the right-hand side its premise.

Kowalski [Kow74] showed that logic as a programming language has a procedural interpretation as well as a logical interpretation. Horn clauses in a logic program can be regarded as procedure definitions and computation is initiated by an initial query, that is, a Horn clause without a conclusion. For instance, `<- add(s(0),s(0),N)` is a query with one atom stating that there is a number that is the sum of the two numbers represented by `s(0)` and `s(0)`.

To prove the atom `add(s(0),s(0),N)` from the program, we first try to match the atom with the conclusions of the Horn clauses by unification. The atom does not match the conclusion of the first clause, since `0` and `s(0)` cannot be unified; but it matches that of the second clause when `X` is unified with `s(0)`, `Y` with `0` and `s(Z)` with `N`. This results in a reduction of the query by the premise of the matching clause, `<- add(s(0),0,Z)`. Then the conclusion of the first clause `add(X,0,X)` matches the atom `add(s(0),0,Z)` when `X` unified with `s(0)` and `X` with `Z`. After reducing the new query statement by the premise of the first clause, an empty query is reached, representing a success. Then the answer to the original query is `<- add(s(0),s(0),s(s(0)))` where `N` is replaced by `s(s(0))`.

The meaning of a logic program is the set of all ground atoms that are the logical consequences of the program [vEK76]; a ground atom does not contain any uninstantiated variable. SLD-resolution is a sound and complete proof procedure for logic programs with respect to the meaning of logic programs (see [Llo84].) Since the atom `add(s(0),s(0),s(s(0)))` can be proved from the above program, by the soundness of SLD-resolution, it is a logical consequence of the program.

1.3 Extensions to Logic Programming

The ultimate goal of logic programming is that we only specify what is to be solved, and we should not be bothered with control issues and implementation details [Kow74] [Kow79]. However, many implementations of logic programming employ non-logical and extra-logical features which violate some of the very principles logic programming is based upon.

Logic programs specify relations, and in this sense there may be many alternative solutions for a given query. For instance, consider the program given previously and the query `<- add(X,Y,s(0))`. There are two different solutions to the query i.e., the ground atoms `add(s(0),0,s(0))` and `add(0,s(0),s(0))` from the first and second clauses respectively. Since in logic programming we have no control over which solution an implementation can produce, a non-logical feature called the cut operator is introduced, which has the effect of pruning the search space while trying to find a solution. Then the completeness of an implementation is destroyed [Llo84] in favour of efficiency.

First-order logic cannot naturally express dynamic properties of certain problems such as simulation and database updates. Therefore the notion of change is brought back into logic programming by some other non-logical features in the form of system predicates such as **assert** and **retract** with much more harmful effects than the

cut operator, destroying the soundness of an implementation along the way. In this case, the pure logical reading of a logic program is no longer relevant, because mixed uses of these non-logical constructs dynamically alter the program itself.

Some deliberately incomplete parallel implementations of logic programming introduce the concept of a *committed choice*, a cleaned-up cut operator [CG81] [Sha87]. In concurrent logic programming languages such as Relational Language [CG81], Parlog [CG86], Concurrent Prolog [Sha87], and GHC [Ued86], a program clause takes the following form:

$$A\text{<-}G_0, \ldots, G_{m-1} \mid B_0, \ldots, B_{n-1}. \quad m, n \geq 0$$

where "|" is called the commit operator; and all of A, G_i's and B_i's are atoms. Here $G_0 \ldots G_{m-1}$ are called *guards* of the clause. When there is more than one alternative clause for proving an atom, all alternative clauses are tried in parallel until one of them reaches its commit operator before the others. Then that clause is committed and all the other clauses are eliminated. Of course, there is no guarantee that the committed clause will lead to a solution for the query.

Parallel logic programming languages offer dataflow modularity and potential for exploiting the computational power of multi-processor architectures, but they also introduce many other non-logical features besides the committed choice mechanism such as annotations to guide the implementation and builtin predicates to find all solutions to a given query. Almost all parallel logic programming languages model dataflow style of non-terminating computations or perpetual processes by employing infinite data structures such as streams. With respect to the minimum model semantics, the set of logical consequences of a given infinitary logic program does not contain any infinitary objects at all. Moreover, the compactness of first-order logic dictates that all proofs are done in finitely many steps. Therefore a pure logical reading of a concurrent logic program is far from what the program is intended to specify and non-logical features can only be described operationally

[Sar87] [GCLS88].

The following infinitary logic program from [vENA84] specifies a non-terminating computation:

```
list-succ(U.X,s(U).Y) <- list-succ(X,Y).
```

Here `list-succ(L1,L2)` means that `L1` and `L2` are infinite sequences of natural numbers in successor notation and `L2` is, element for element, the successor of `L1`. As shown in [vENA84], the set of logical consequences of the above program is empty, and yet given a query like `<- list-succ(0.Z,Z)`, an implementation of infinitary logic programming produces an infinite sequence of numbers `s(0).s(s(0))...` for `Z`, one at a time, and never terminates.

There have been many approaches to developing declarative semantics of infinitary logic programming languages. Most of these approaches employ the greatest fixpoint techniques and Herbrand interpretations based on a Herbrand universe extended with infinitary terms, e.g., see [vENA84] [Llo84] [NA85] and [Mur88]. Some recent approaches such as [GCLS88] and [dBK88] have provided a denotational/compositional semantics for subsets of committed choice languages inspired by the denotational semantics of parallel imperative languages, justifying that these committed-choice languages are better understood operationally. Each approach has its own merits; but the conclusion is that we are dealing with a non-logical programming paradigm [Wad85] [GCLS88] [HA88].

The current trend in the concurrent/infinitary logic programming community is that complete implementations are out of the question and non-logical features are necessary to improve the expressiveness and efficiency of logic programming languages [CG86] [Sha87] [GCLS88]. In our opinion, incomplete implementations are tolerable to some extent; but the notion of logical consequence is essential. Thus the solution lies not in extending logic programming with extra-logical and non-logical features, but rather in employing more powerful logics which will enable us

to model the notion of dynamic change and non-terminating computations gracefully. As mentioned earlier, some non-classical logics have already been employed successfully in reasoning about programs in the form of algorithmic, temporal and dynamic logic; see Harel [Har84] for a short survey of these logics. Then why not use non-classical logics to write programs in the first place?

1.4 Intensional Logic Programming

This dissertation presents an investigation of logic programming based on intensional logic, hence the term "intensional logic programming (ILP)". Through intensional logic, the notion of dynamic change and the ability to reason about context-dependent properties can be brought back into logic programming more naturally, and, more importantly, without any extra-logical or non-logical features. The dissertation will mainly focus on the model-theoretic issues of intensional logic programming, and therefore it will not address any proof-theoretical issues or implementation techniques. More specifically, we are interested in exploring the general conditions under which an intensional logic programming language is acceptable as a programming formalism.

Intensional logic studies context-dependent properties [Mon74]. Therefore, in intensional logic, the meaning of an expression cannot be determined without knowledge of the context of its use. Depending on the target application, the context may be a moment in time, a node in a tree structure, a point in an n-dimensional space and so on. The meaning of a logical statement is in fact a set of truth values indexed by the elements of a given collection of possible contexts ("possible worlds" in the terminology invented by Kripke). The family is also called the *intension* of the expression and each object in the family denotes the *extension* of the expression in a particular possible world. An intensional logic is equipped with intensional

operators through which context-dependent properties can be expressed. Temporal logic [RU71] can be regarded as an instance of intensional logic where the set of possible worlds models a collection of moments in time.

Intensional logic programs are regarded as a set of logical axioms or assertions, based on an extension of Horn logic, interpreted as statements true in all possible worlds. We will adopt the clausal notation of Kowalski [Kow74] for intensional logic programs. In the dissertation, we will give several examples of intensional logic programming including the temporal language Chronolog proposed by Wadge [Wad85] [Wad88]. Chronolog is based on a temporal logic where the collection of moments in time is the set of natural numbers. In Chronolog, we can model non-terminating computations. For instance, the following Chronolog program [MF89] specifies the simulation of a traffic light, which starts with a **green** light, and then goes from **green** to **amber**, from **amber** to **red** and from **red** to **green** and so on.

```
first light(green).
next light(amber) <- light(green).
next light(red) <- light(amber).
next light(green) <- light(red).
```

The temporal operator **first** refers to the initial moment in time and **next** the next moment in time. Given the query **<- light(X)**, an implementation of Chronolog produces the answers **light(green)** at time 0, **light(amber)** at time 1, **light(red)** at time 2, **light(green)** at time 3 and so on.

At any given moment in time, the answer to the query is a logical consequence of the program. Since the query is open-ended, i.e., not fixed to any moment in time, the implementation tries to prove it for each moment of time in turn, hence runs forever, producing all answers to the query. This way, non-terminating computations can be modeled within a logical framework. There is no need to employ infinitary structures in the object language such as streams, because a stream may

be represented by a time-varying predicate such as `light`. Note that at any given moment in time, only one of the ground atoms `light(green)`, `light(amber)` and `light(red)` is true of the program.

In contrast, we can write an infinitary logic program to simulate the operations of the traffic light in question, in which the `light` predicate is intended to represent the infinite list `[green,amber,red,green,...]`.

```
light([green|L]) <- switch([green|L]).
switch([green,amber|L]) <- switch([amber|L]).
switch([amber,red|L]) <- switch([red|L]).
switch([red,green|L]) <- switch([green|L]).
```

Given the query `<- light(L)`, an implementation of infinitary logic programming produces the colours in the infinite list `[green,amber,red,green,...]`, one at a time, and never terminates. However, these partial answers are not justified by the minimum model semantics. The set of logical consequences of this program is empty and therefore the intended meaning of the program is not what it denotes! In other words, the declarative meaning of the program is no longer relevant to explain its procedural behavior. Such programs compromise the credibility of logic programming as declarative programming.

There are many non-classical logic programming languages, based on different kinds of intensional logic: Molog [dC86] is based on user-elected modal logics and it is suitable for epistemic reasoning and knowledge representation. The simple language described in [Sak87] is also based on modal logic. Templog [AM87] and Temporal Prolog [Gab87] are based on temporal logic, very similar to that of Chronolog. Both of these languages can be used for temporal reasoning and modeling temporal databases. InTense [MF89] is based on a multi-dimensional intensional logic with temporal and spatial dimensions where a possible world is just a point in a time-space hyperfield. Tokio [AFMo86] is based on interval temporal logic, and it can

be used in hardware specification and verification. Of course, there are many other non-classical logics which can be used in logic programming.

To the best of our knowledge, there are very few attempts at developing model-theoretical semantics for these non-classical languages. Baudinet [Bau88] [Bau89] has independently provided a van Emden-Kowalski style semantics for Templog programs and shown the completeness of temporal logic programming. Balbiani et al [BdCH88] has developed the declarative semantics of some instances of modal logic programming in Molog, based on a tree-like semantics.

In this dissertation, we will study the semantic problems of intensional logic programming languages from a unifying point of view. In other words, we will not focus on any specific intensional language, instead, we will aim at isolating the general properties of intensional logic programs, and then discovering the conditions under which these properties may be satisfied.

1.5 A Preview of the Dissertation

We will now provide a preview of the dissertation.

Chapter 2 summarizes the preliminary material for the dissertation. After introducing the mathematical notation which will be used throughout, we define syntactical and semantical properties of first-order logic. Logic programming is based on a special kind of logic called Horn logic. The model-theoretical semantics of (Horn) logic programs developed by van Emden and Kowalski [vEK76] is based on Herbrand interpretations and the notion of the least (minimum) Herbrand model. We will briefly outline the syntax and semantics of intensional logic and introduce the notion of an intensional interpretation which assigns meanings to all elements of an intensional language at all possible worlds.

In Chapter 3, we will introduce several ILP languages including the temporal

language Chronolog [Wad85] [Wad88]. Chronolog is suitable for modeling time-varying properties and non-terminating computations and deserves special attention for its simplicity. We will also address the model-theoretic semantics of temporal logic programs, and, in particular, show that Chronolog programs enjoy the least (minimum) model semantics based on the notion of a canonical temporal ground atom and temporal Herbrand models [OW88a]. We will point out that the semantics of temporal logic programs is not general enough to extend to arbitrary intensional languages. This chapter will also outline a set of rules of inference which can be used to devise a proof procedure for temporal logic programming.

Chapter 4 will introduce a generalized intensional semantics for intensional logic. Intensional semantics will provide us with an abstract characterization of intensional logic, which can be exploited further to develop a language-independent model theory for intensional logic programs. The denotations of intensional operators will be abstracted as functions over intensions, i.e., elements of

$$[\, \mathit{intensions} \times \mathit{intensions} \times \ldots \times \mathit{intensions} \rightarrow \mathit{intensions} \,]$$

where an intension is a function from possible worlds to truth values. We will also adopt "neighborhood semantics" for intensional operators, developed by Scott [Sco70] and Montague [Mon74] as an alternative to Kripke-style semantics. We will define several important properties of intensional operators, namely, universality, monotonicity, conjunctivity, finitariness and continuity [OW89b]. We will show that each property has a certain model-theoretical consequence for the neighborhood semantics.

In Chapter 5, we will develop a generalised model theory for intensional logic programs, in the style of van Emden-Kowalski [vEK76], based on intensional Herbrand interpretations. We will show that intensional logic programs of a given intensional logic enjoy the minimum model semantics provided that intensional operators of the logic meet the semantic properties given in Chapter 4. We will elaborate how

each property affects the semantics of intensional logic programs: universality and monotonicity are related to model existence; conjunctivity to the model-intersection property; monotonicity and finitariness to the least fixpoint semantics. We will also show that our results can be applied to the temporal languages Chronolog [Wad85], Templog [AM87] and Temporal Prolog [Gab87]; the intensional language InTense [MF89]; the modal language Molog [dC86]; and the interval language Tokio [AFMo86].

Chapter 6 presents how non-determinism can be modeled in intensional logic programming. As mentioned earlier, logic programming is inherently non-deterministic since there may be more than one possible solution to a given query. When we want to model a dataflow style of stream-oriented computations, it is desirable to have exactly one solution to the query chosen arbitrarily among all possible solutions (if any). Committed choice languages offer dataflow modularity to some extent [Sha87], but cannot guarantee single-valued solutions in model-theoretical terms.

Wadge [Wad85] [Wad88] proposed an extension to intensional logic programming called "choice predicates". Choice predicates are associated with each predicate appearing in a given intensional logic program, and represent arbitrary but definite single-valued relations at each possible world extracted from those the original predicates represent. Therefore they offer a dataflow style of stream-oriented communication within intensional logic programming. However, the minimum model semantics is no longer valid for programs with choice predicates. In fact, the meaning of such a program can be characterised in terms of "minimal models"; therefore choice predicates are a very controlled extension of intensional logic programming beyond Horn logic.

In Chapter 7, we will consider how to improve the expressive power of intensional logic programming from within. We will show that intensional program clauses can be used to define intensional operators; therefore programmers are free to extend the

underlying intensional logic through the tools already available in the language. An intensional logic program with intensional operator definitions can be transformed into an "equivalent" program without new intensional operators and with possibly more program clauses than the original program. We will describe a transformation procedure and show its correctness. When recursive definitions are allowed, the fixpoint techniques have to be employed. However, recursive definitions require the use of an infinitary intensional logic (with countable conjunctions and disjunctions).

Chapter 8 summarizes the main results and contributions of the dissertation. We will also point out possible extensions of the concepts investigated in the dissertation and other issues where further research is profitable.

Chapter 2

Background

This chapter is devoted to the preliminary material needed for the development of the theory of intensional logic programming. After introducing some mathematical notations, the syntax and semantics of first-order logic will be defined; logic programming is based on a special kind of logic, called Horn logic. Then the model-theoretical semantics of logic programs will be summarized, including the fixpoint characterisation of the minimum Herbrand model. A brief overview of the syntax and semantics of intensional logic will follow. We will give the formal definition of an intensional interpretation which assigns meanings to all elements of an intensional language at all possible worlds. We will also outline how the semantics of intensional operators can be defined in Kripke-style semantics for intensional logic.

2.1 Mathematical Notation

In this thesis we use the von Neumann set-theoretic representation of natural numbers. Then the set of natural numbers is denoted by ω and is the set $\{0, 1, 2, 3, \ldots\}$. The number 0 is the emptyset, i.e., $0 = \emptyset$. For any given number $n \in \omega$, n is the set $\{0, 1, \ldots, n-1\}$.

We use the following set-building notation: The set $\{x \mid C(x)\}$ is the collection of all elements x that satisfy the condition $C(x)$. The set of all subsets of a given set S (the power set of S) is denoted by $P(S)$ and is the collection $\{X \mid X \subseteq S\}$. Given a set S, a selective subset of S with respect to some condition is the set $\{x \in S \mid C(x)\}$.

The Cartesian product of two sets S_0 and S_1 is denoted by $S_0 \times S_1$ and is the set of ordered pairs $\{< x, y > \mid x \in S_0 \text{ and } y \in S_1\}$. Given any finite number of sets $S_0, \ldots, S_{n-1}$, the Cartesian product $S_0 \times \ldots \times S_{n-1}$ is the set of ordered n-tuples $\{< x_0, \ldots, x_{n-1} > \mid x_i \in S_i \text{ for all } i \in n\}$, and also denoted by $\prod_{i \in n} S_i$. If $S = S_i$ for all $i \in n$, we write S^n for $\prod_{i \in n} S_i$.

Any subset R of $S_0 \times S_1$ is called a binary relation from S_0 to S_1. Given a set S, any subset R of $S \times S$ is said to be a relation over S. Then we say that

- R is *reflexive* iff $< x, x > \in R$ for all $x \in S$;
- R is *transitive* iff $< x, y > \in R$ and $< y, z > \in R$ implies $< x, z > \in R$;
- R is *symmetrical* iff $< x, y > \in R$ implies $< y, x > \in R$.

If D and S are sets, a function from D to S is a binary relation f where for any given $x \in D$, there is a unique element $y \in S$ with $< x, y > \in f$. The set of all functions from D to S is denoted by $[D \rightarrow S]$. An n-ary function f is an element of $[D_0 \times \ldots \times D_{n-1} \rightarrow S]$ where $D_0, \ldots, D_{n-1}$ and S are sets. Given two functions $f \in [S \rightarrow V]$ and $g \in [D \rightarrow S]$, the composition of the functions f and g is a function in $[D \rightarrow V]$ denoted by $f \circ g$ where for all $x \in D$, $f \circ g(x) = f(g(x))$.

Given two sets I and S, an I-indexed sequence is any function $s \in [I \rightarrow S]$ denoted by $\{s_\alpha\}_{\alpha \in I}$. The set I is called the index set of S. If S is a set of sets, $S = \{s_\alpha\}_{\alpha \in I}$ is said to be a family of sets. Then the Cartesian product of a family S is denoted by $\prod_{\alpha \in I} s_\alpha$, or simply $\prod S$.

The cardinality of a given set S is denoted by $\mathtt{card}(S)$. If there exists a surjective function $I \in [\omega \rightarrow S]$, the set S is called *countable*; if $\mathtt{card}(S) \in \omega$, we say that S is

finite. Note that $\mathbf{card}(n) = n$ for any $n \in \omega$.

2.2 First-Order Logic

First-order logic comprises two parts: syntax and semantics. The syntax is about what the elements of the logic are and how they can be formed, while the semantics say what those elements denote.

2.2.1 Syntactical Properties

The underlying language of first-order logic consists of logical connectives such as $\neg$, $\wedge$ and $\forall$, auxiliary symbols such as "(" and ")", a countable set of variables, a countable set of function symbols and a countable set of predicate symbols. We do not differentiate between the meta-language and object language symbols for parentheses. We now give the formal definitions of *terms* and *formulas.*

Definition 2.1 *A term is defined inductively as follows:*

- *All variables are terms.*
- *If $e_0 \ldots e_{n-1}$ are terms and f is an n-ary function symbol, then $f(e_0 \ldots e_{n-1})$ is a term.*

Nullary (0-ary) function symbols are called constants. Predicate symbols take terms as their arguments to form the basis case of formulas (atomic formulas.) Atomic formulas are also called atoms.

Definition 2.2 *A formula is defined inductively as follows:*

- *Let p be an n-ary predicate symbol and $e_0 \ldots e_{n-1}$ terms. Then $p(e_0 \ldots e_{n-1})$ is an atomic formula (or simply an atom.)*
- *If A and B are formulas, so are $A \wedge B$, $\neg A$.*

- *If A is a formula and x is a variable, then $(\forall x)A$ is a formula.*

The symbol $\forall$ is called the universal quantifier. We regard $\wedge$ and $\neg$ as primitives in the language and introduce other logical connectives in terms of $\wedge$ and $\neg$ as follows: Let A and B be formulas.

- $A \vee B =_{def} \neg(\neg A \wedge \neg B)$ (read as A or B.)
- $A \rightarrow B =_{def} \neg A \vee B$ (read as A implies B.)
- $A \leftrightarrow B =_{def} (A \rightarrow B) \wedge (B \rightarrow A)$ (read as A iff B.)

The existential quantifier ($\exists$) can be defined in terms of the universal quantifier ($\forall$) and negation $\neg$: $(\exists x)A =_{def} \neg(\forall x)\neg A$.

A variable is *free* in a formula if it is not within the scope of a binding quantifier. For instance, the variable x is free in the formula $p(x) \wedge (\forall y)q(x, y)$, but the variable y is not. Below is the formal definition of the function `free` which, given an element E of L, returns the set of all free variables that occur in E.

Definition 2.3 *The function* `free` *is defined inductively as follows:*

- $\texttt{free}(x) = \{x\}$ *where x is a variable.*
- $\texttt{free}(f(e_0, \ldots, e_{n-1})) = \bigcup_{i \in n} \texttt{free}(e_i)$ *where $f(e_0, \ldots, e_{n-1})$ is a term.*
- $\texttt{free}(p(e_0, \ldots, e_{n-1})) = \bigcup_{i \in n} \texttt{free}(e_i)$ *when $p(e_0, \ldots, e_{n-1})$ is an atomic formula.*
- $\texttt{free}(\neg A) = \texttt{free}(A)$ *where A is a formula.*
- $\texttt{free}(A \wedge B) = \texttt{free}(A) \cup \texttt{free}(B)$ *where A and B are formulas.*
- $\texttt{free}((\forall x)A) = \texttt{free}(A) - \{x\}$ *where A is a formula.*

Let e be a term. We say that e is a ground term if $\texttt{free}(e) = \emptyset$.

Let $p(e_0 \ldots e_{n-1})$ be an atomic formula. We say that $p(e_0 \ldots e_{n-1})$ is a ground atom if $\texttt{free}(p(e_0 \ldots e_{n-1})) = \emptyset$.

Definition 2.4 *A substitution θ is a finite set of the form $\{x_0/e_0, \ldots, x_{n-1}/e_{n-1}\}$ where each x_i is a variable, and each e_i is a term. Furthermore, all x_i's are distinct variables and for all $i \in n$, $x_i \neq e_i$.*

The notion of a substitution extends to formulas. Note that the following definition is still informal, since it does not explicitly specify how substitutions are performed. A rigorous definition can be found in [Gal86].

Definition 2.5 *Let $\theta = \{x_0/e_0, \ldots, x_{n-1}/e_{n-1}\}$ be a substitution and E be a term or quantifier-free formula. Then $E\theta$ is an instance of E obtained from E by simultaneously replacing every occurrence of x_i by e_i for all $i \in n$. If $\texttt{free}(E\theta) = \emptyset$, $E\theta$ is said to be a ground instance of E; and θ a ground term substitution.*

We will also need the notion of a *formula substitution* described as follows: Let A and P be formulas. Then $A\{P/B\}$ is a formula obtained from A by replacing all occurrences of P by B. The definition of formula substitution is given below.

Definition 2.6 *A formula-substitution ϑ is a finite set of pairs of the form $\{P_0/B_0, \ldots, P_{n-1}/B_{n-1}\}$ where all P_i's and B_is are formulas. Furthermore, all P_i's are distinct formulas and for all $i \in n$, $P_i \neq B_i$.*

Note that the following definition does not explicitly specify how formula substitutions are performed; see [Seg82] and [Gal86] for a more detailed treatment of this topic.

Definition 2.7 *Let $\vartheta = \{P_0/B_0, \ldots, P_{n-1}/B_{n-1}\}$ be a formula-substitution and A be a formula. Then $A\vartheta$ is a formula obtained from E by simultaneously replacing every occurrence of P_i by B_i for all $i \in n$.*

2.2.2 Interpretations, Semantics

For a given first-order language, the semantics of its elements are defined with respect to interpretations and satisfaction relation $\models$. In Tarski's model-theoretical approach, the semantic procedure consists of supplying a domain of discourse and an interpretation which assigns meanings to variables, function and predicate symbols by the following:

Definition 2.8 *An interpretation I of a first-order language L comprises a non-empty set $\mathbf{D}$, called the domain of the interpretation, over which the variables range, together with for each variable, an element of $\mathbf{D}$; for each n-ary function symbol, an element of $[\mathbf{D}^n \rightarrow \mathbf{D}]$; and for each n-ary predicate symbol, an element of $P(\mathbf{D}^n)$.*

The fact that a formula A is true in interpretation I will be denoted as $\models_I A$; $\models$ is called the satisfaction relation and it extends an interpretation upwards to all elements of a given language. Furthermore, $\models A$ means that A is true in any interpretation, i.e., A is valid. Then the definition of $\models$ in terms of interpretations is given as follows.

Definition 2.9 *The semantics of elements of a first-order language L are given inductively by the following, where I is an interpretation of L, and A and B are formulas.*

(a) $I(f(e_0, \ldots, e_{n-1})) = I(f)(I(e_0), \ldots, I(e_{n-1})) \in \mathbf{D}$ *where* $f(e_0, \ldots, e_{n-1})$ *is a term.*

If v is a variable, then $I(v) \in \mathbf{D}$.

(b) *For any n-ary predicate p and terms $e_0, \ldots, e_{n-1}$, $\models_I p(e_0, \ldots, e_{n-1})$ iff $< I(e_0), \ldots, I(e_{n-1}) > \in I(p)$.*

(c) $\models_I \neg A$ *iff* $\not\models_I A$.

(d) $\models_I A \wedge B$ *iff* $\models_I A$ *and* $\models_I B$.

(e) $\models_I (\forall x)A$ *iff* $\models_{I[d/x]} A$ *for all* $d \in \mathbf{D}$ *where* $I[d/x](x) = d$.

Here the interpretation $I[d/x]$ agrees with I on everything except possibly the assignment to the variable x.

2.3 Logic Programs

Any formula in first-order logic can be transformed into an equivalent one in clausal form; for details, see [Gal86]. Clauses play an important role in logic programming.

Definition 2.10 *A clause is a formula of the form*

$$(\forall x_0)\ldots(\forall x_{k-1})(\neg B_0 \vee \ldots \vee \neg B_{n-1} \vee A_0 \vee \ldots \vee A_{m-1})$$

where all A_i*'s and* B_i*'s are atomic formulas, and* $\texttt{free}(\neg B_0 \vee \ldots \vee \neg B_{n-1} \vee A_0 \vee \ldots \vee A_{m-1}) = \{x_0, \ldots, x_{k-1}\}$.

We will adopt a special clausal notation from [Kow79]. Throughout, a clause will be denoted by

$$A_0, \ldots, A_{m-1} \leftarrow B_0, \ldots, B_{n-1}$$

where all variables are assumed to be universally quantified. The set $\{A_0, \ldots, A_{m-1}\}$ is the conclusion (head) of the clause; the set $\{B_0, \ldots, B_{n-1}\}$ is the premise (body) of the clause.

In logic programming, a special form of clauses are of interest. A clause C is called a *Horn clause* if it is either of the form

$$A \leftarrow B_0, \ldots, B_{n-1} \ \ (n \geq 0),$$

or of the form

$$\leftarrow B_0, \ldots, B_{n-1} \ \ (n > 0).$$

where A and all B_i's are atomic formulas. All Horn clauses of the first form are called *definite clauses*, or program clauses. All Horn clauses of the second form are called *goal clauses*, or queries.

The right-hand side of any given program clause is called the *body* or premise of the clause and the left-hand side of the clause is called the *head* or conclusion of the clause. The informal semantics of a program clause $A \leftarrow B_0, \ldots, B_{n-1}$ is defined as follows: for each variable assignment, if all of $B_0, \ldots, B_{n-1}$ are true, then A is true. (It follows from the definition of the semantics of the universal quantifier $\forall$.) A clause of the form $A \leftarrow$ is an unconditional assertion.

A logic program consists of the conjunction of a set of program clauses regarded as axioms representing our knowledge and assumptions about some problem.

Definition 2.11 *A logic program is a finite set of program clauses.*

We are interested in those interpretations of a logic program which make the program true, i.e., I is a model of $\mathcal{P}$ iff $\models_I \mathcal{P}$. The semantics of the conjunction $\wedge$ implies that a logic program is true in an interpretation I iff all clauses in the program are true in I.

Definition 2.12 *Let $\mathcal{P}$ be a logic program and let I be an interpretation of $\mathcal{P}$. Then I is a model for $\mathcal{P}$ iff I is a model for each clause in $\mathcal{P}$, that is, $\models_I \mathcal{P}$ iff for all clauses $C_i \in \mathcal{P}$, $\models_I C_i$.*

2.4 Models of Logic Programs

Van Emden and Kowalski [vEK76] developed an elegant formalisation of model-theoretical semantics of logic programs. There the meaning of a given logic program is defined in terms of the least (minimum) Herbrand model. In the following, we will summarize their results from different sources including [vEK76], [AvE82] and [Llo84].

Herbrand Interpretations

We will now study Herbrand interpretations. Note that if a given logic program has no nullary function symbols (constants), we just add an arbitrary one into the domain of Herbrand interpretations.

Definition 2.13 *Let $\mathcal{P}$ be a logic program. The Herbrand universe $U_{\mathcal{P}}$ of $\mathcal{P}$ is the set of all ground terms which can be constructed out of constants and functions that appear in $\mathcal{P}$.*

Definition 2.14 *Let $\mathcal{P}$ be a logic program. The Herbrand base $B_{\mathcal{P}}$ of $\mathcal{P}$ is the set of all ground atoms which can be constructed out of predicates that appear in $\mathcal{P}$ with ground terms in $U_{\mathcal{P}}$ as arguments.*

A Herbrand interpretation satisfies a certain property: The domain of Herbrand interpretations is the Herbrand universe, and all ground terms literally represent themselves.

Definition 2.15 *Let $\mathcal{P}$ be a logic program and I be an interpretation of $\mathcal{P}$. Then I is called a Herbrand interpretation of $\mathcal{P}$ if the domain of I is $U_{\mathcal{P}}$ and for all $e \in U_{\mathcal{P}}$, $I(e) = e$.*

A Herbrand interpretation I of $\mathcal{P}$ can be identified with a subset H of the Herbrand base $B_{\mathcal{P}}$ by the following: For any Herbrand interpretation, the corresponding subset of the Herbrand base is the set of all ground atoms that are true with respect to the interpretation. Then

$$< e_0, \ldots, e_{n-1} > \in I(p) \text{ iff } p(e_0, \ldots, e_{n-1}) \in H$$

Note that there is a one-to-one correspondence between Herbrand interpretations and subsets of the Herbrand base of a given program. We will refer to any subset of $B_{\mathcal{P}}$ as a Herbrand interpretation. We say that H is a model of $\mathcal{P}$ iff $\models_I \mathcal{P}$ for any I corresponding to H, and $\models_I \mathcal{P}$ iff $\models_I C$ for all clauses $C \in \mathcal{P}$.

Definition 2.16 *Let $\mathcal{P}$ be a logic program and I be a Herbrand interpretation of $\mathcal{P}$. Consider a program clause $C \in \mathcal{P}$; then I is a model of C iff $\models_I C_i$ for all ground instances C_i of C.*

This definition is just a reformulation of the semantics of the universal quantifier $\forall$ for program clauses and Herbrand interpretations.

If $(A \leftarrow B_0, \cdots, B_{n-1})$ is a ground instance of a clause in $\mathcal{P}$ and I is a Herbrand interpretation of $\mathcal{P}$, we say that $\models_I (A \leftarrow B_0, \cdots, B_{n-1})$ iff $A \in I$ or for some $i \in n$, $B_i \notin I$.

The following lemma justifies that Hebrand interpretations are sufficient for clausal logic. It is a consequence of the downward Löwenheim-Skolem theorem [CK73] which states if a set of formulas is true in some model, then it is true in a countable model (a model whose domain is at most countable.) For any given set of clauses, the corresponding Herbrand universe is at most countable.

Lemma 2.1 *Let $\mathcal{S}$ be a set of clauses. Then $\mathcal{S}$ has a model iff it has a Herbrand model.*

The following lemma follows from lemma 2.1.

Lemma 2.2 *Let $\mathcal{P}$ be a logic program and $A \in B_{\mathcal{P}}$. Then $\mathcal{P} \cup \{\neg A\}$ is unsatisfiable iff no Herbrand model of $\mathcal{P}$ satisfies $\mathcal{P} \cup \{\neg A\}$.*

The Minimum Herbrand Model

Logic programs have a very important property, that is, they are consistent, and the family of Herbrand models of a given logic program is closed under intersection. Therefore a logic program singles out a particular Herbrand interpretation, called the least (minimum) Herbrand model.

Lemma 2.3 *Let $\mathcal{P}$ be a logic program; then $B_{\mathcal{P}}$ is a model of $\mathcal{P}$.*

Lemma 2.4 [Llo84] *Let $\mathcal{P}$ be a logic program and $M = \{I_\alpha\}_{\alpha \in S}$ be a non-empty family of Herbrand models for $\mathcal{P}$. Then $\cap M =_{def} \cap_{\alpha \in S} I_\alpha$ is a Herbrand model for $\mathcal{P}$.*

The following theorem follows from lemmas 2.3 and 2.4.

Theorem 2.5 [vEK76] *Let $\mathcal{P}$ be a logic program. Then $\mathcal{P}$ has a minimum Herbrand model $M_\mathcal{P}$, which is the intersection of all Herbrand models of $\mathcal{P}$.*

The minimum Herbrand model contains the least amount of information which makes the program true and it coincides with the set of ground atoms that are logical consequences of the program. Any implementation of logic programming must construct the minimum Herbrand model of a given logic program. This is how the correctness of an implementation is defined.

Theorem 2.6 [vEK76] *Let $\mathcal{P}$ be a logic program. Then $M_\mathcal{P} = \{A \in B_\mathcal{P} \mid \mathcal{P} \models A\}$.*

2.5 The Fixpoint Semantics of Logic Programs

There is a certain mapping, due to [vEK76], which is used to characterize the fixpoint semantics of logic programs. After a brief introduction to fixpoints, we will summarize some of the results from [vEK76], [AvE82] and [Llo84].

Monotonic Mappings, Fixpoints

Let L be a complete lattice with an ordering relation $\sqsubseteq$, the greatest lower bound operation $\sqcap$, and the least upper bound operation $\sqcup$. For any given subset $S = \{X_\alpha\}_{\alpha \in I}$ of L, we define $\sqcap S =_{def} \sqcap_{\alpha \in I} X_\alpha$ and $\sqcup S =_{def} \sqcup_{\alpha \in I} X_\alpha$.

Definition 2.17 *Let L be a complete lattice and $T \in [L \to L]$. Then we say that T is monotonic iff for all $X, Y \in L$, $X \sqsubseteq Y$ implies $T(X) \sqsubseteq T(Y)$.*

Monotonic mappings over complete lattices have very nice properties: They have fixpoints and furthermore the set of fixpoints of a monotonic mapping T is equivalent to the set of its pre-fixpoints.

Theorem 2.7 (Knaster-Tarski fixpoint theorem.) [Llo84] *Let L be a complete lattice and $T \in [L \to L]$ be monotonic. Then T has a least fixpoint, lfp(T), and a greatest fixpoint, gfp(T). Furthermore, $lfp(T) = \sqcap\{X \mid T(X) = X\} = \sqcap\{X \mid T(X) \sqsubseteq X\}$, and $gfp(T) = \sqcup\{X \mid T(X) = X\} = \sqcup\{X \mid T(X) \sqsupseteq X\}$.*

The following notation will be frequently used for ordinal powers of a given mapping up to ω.

Definition 2.18 *Let L be a complete lattice and $T \in [L \to L]$ be monotonic. Then we define*

$$T \uparrow 0 = \sqcap L,$$
$$T \uparrow n = T(T \uparrow (n-1)),$$
$$T \uparrow \omega = \sqcup_{n \in \omega} T \uparrow n.$$

Let S be a subset of L. Then we say that S is a *chain* if it is a total order with respect to $\sqsubseteq$, i.e., for all $X, Y \in S$, either $X \sqsubseteq Y$ or $Y \sqsubseteq X$. If S is countable, it will be called an ω-chain, and written as $S =< X_n >_{n \in \omega}$. We say that S is a *downwards* chain if it is a total order with respect to $\sqsupseteq$. Note that a chain is *upwards* by default.

Definition 2.19 *Let L be a complete lattice and $T \in [L \to L]$. Then we say that T is continuous iff for all chains $< X_n \in L >_{n \in \omega}$, $T(\sqcup_{n \in \omega} X_n) = \sqcup_{n \in \omega} T(X_n)$.*

Monotonicity guarantees the existence of the least and greatest fixpoints of a given mapping, while continuity guarantees that the least fixpoint of a mapping can be approximated from below. This is implied by the Kleene recursion theorem. Note that here continuity implies monotonicity.

Lemma 2.8 (Kleene.) [Llo84] *Let L be a complete lattice and $T \in [L \to L]$. If T is continuous, then the least fixpoint of T, denoted as lfp(T), is equal to $T \uparrow \omega$.*

The Mapping $T_{\mathcal{P}}$

The mapping $T_{\mathcal{P}}$, the one step modus ponens function originally defined in [vEK76], provides the basis for the fixpoint semantics and establishes the connection between the model-theoretical and operational semantics of logic programs. Let $\mathcal{F}(\mathcal{P})$ denote the set of Herbrand interpretations of a given logic program $\mathcal{P}$.

Definition 2.20 *Let $\mathcal{P}$ be a logic program and H be a subset of $B_{\mathcal{P}}$. Then the mapping $T_{\mathcal{P}} \in [\mathcal{F}(\mathcal{P}) \rightarrow \mathcal{F}(\mathcal{P})]$ where $T_{\mathcal{P}}(H)$ is a Herbrand interpretation of $\mathcal{P}$ given as*

$$T_{\mathcal{P}}(H) = \{A \mid A \leftarrow B_0, \ldots, B_{n-1} \text{ is a ground instance of a clause in } \mathcal{P} \text{ and } \{B_0, \ldots, B_{n-1}\} \subseteq H\}$$

Let $\mathcal{P}$ be a logic program. Then $\mathcal{F}(\mathcal{P})$ is a complete lattice with the ordering relation $\subseteq$. The least and greatest upper bounds of a given subset S of $\mathcal{F}(\mathcal{P})$ are $\cap S$ and $\cup S$ respectively. The following lemma states that $T_{\mathcal{P}}$ is continuous and therefore its least fixpoint can be approximated from below by lemma 2.8.

Lemma 2.9 [vEK76] *Let $\mathcal{P}$ be a logic program. Then $T_{\mathcal{P}}$ is continuous.*

Herbrand models of a given logic program can be characterised in terms of $T_{\mathcal{P}}$.

Lemma 2.10 [vEK76] *Let $\mathcal{P}$ be a logic program and I be a Herbrand interpretation of $\mathcal{P}$. Then I is a model of $\mathcal{P}$ iff $T_{\mathcal{P}}(I) \subseteq I$.*

Lemma 2.10 proves the following equivalence for Herbrand interpretations (models): $\{I \mid T_{\mathcal{P}}(I) \subseteq I\} = \{I \mid \models_I \mathcal{P}\}$. We have that $T_{\mathcal{P}}$ is continuous, and therefore $lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$ by lemma 2.8. According to the Knaster-Tarski fixpoint theorem, the least fixpoint of $T_{\mathcal{P}}$ is given as $lfp(T_{\mathcal{P}}) = \cap\{I \mid T_{\mathcal{P}}(I) \subseteq I\}$, which in turn implies that $M_{\mathcal{P}} = lfp(T_{\mathcal{P}})$. Then we have the theorem given below which states that the least fixpoint of $T_{\mathcal{P}}$ coincides with the minimum Herbrand model:

Theorem 2.11 [vEK76] *Let $\mathcal{P}$ be a logic program. Then $M_{\mathcal{P}} = lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$.*

2.6 Intensional Logic

Intensional logic is the study of context-dependent properties [Mon74] [HC68]. In intensional logic, the meanings of expressions depend on an implicit context, abstracted away from the object language. Temporal logic [RU71] can be regarded as an instance of intensional logic where the collection of contexts is interpreted as a collection of moments in time. The collection of contexts is also called the *universe* or the set of *possible worlds*, and denoted by $\mathcal{U}$. Depending on the target application, a possible world may be a moment in time, a node in a tree structure, a point in a hyper-field and so on. The set of possible worlds $\mathcal{U}$ is not a disorganised collection. For instance, in temporal logic, $\mathcal{U}$ can be regarded as a linearly ordered set with respect to the $\leq$ relation. In the following, we do not impose any restrictions on the choice of the universe and its elements.

An intensional logic is equipped with intensional operators through which elements from different contexts can be combined. The underlying language is obtained from a first-order language by extending it with formation rules for intensional operators as follows. Let IOP denote the set of intensional operators of the language. Let $\bigtriangledown \in IOP$ be an n-ary intensional operator, and $A_0, \ldots, A_{n-1}$ are formulas. Then $\bigtriangledown(A_0, \ldots, A_{n-1})$ is also a formula. The definition of the function `free` (definition 2.3) can be extended to cover formulas of the form $\bigtriangledown(A_0, \ldots, A_{n-1})$ by the following:

- $\texttt{free}(\bigtriangledown(A_0, \ldots, A_{n-1})) = \bigcup_{i \in n} \texttt{free}(A_i)$.

2.6.1 Intensional Interpretations

Let IL denote the underlying intensional language of an intensional logic. An intensional interpretation of IL basically assigns meanings to all elements of IL at all possible worlds in $\mathcal{U}$. An intensional interpretation can also be viewed as a collection

of first-order interpretations (Tarskian structures) associated with possible worlds in $\mathcal{U}$. Then the formal definition of an intensional interpretation can be given as follows.

Definition 2.21 *An intensional interpretation I of an intensional language IL comprises a non-empty set $\mathbf{D}$, called the domain of the interpretation, over which the variables range, together with for each variable, an element of $\mathbf{D}$; for each n-ary function symbol, an element of $[\mathbf{D}^n \rightarrow \mathbf{D}]$; and for each n-ary predicate symbol, an element of $[\mathcal{U} \rightarrow P(\mathbf{D}^n)]$.*

Note that the denotations of variables and function symbols are *extensional* (a.k.a. *rigid*), that is, independent of the elements of $\mathcal{U}$. This is not generally so in intensional logic; but quite satisfactory for the theory of intensional logic programs.

The fact that a formula A is true at world w in some intensional interpretation I will be denoted as $\models_{I,w} A$. All formulas of *IL* are *intensional*, that is their meanings may vary depending on the elements of $\mathcal{U}$. The definition of the satisfaction relation $\models$ in terms of intensional interpretations is partially given as follows. Let $I(E)$ denote the value (in $\mathbf{D}$) that I gives an *IL* term E.

Definition 2.22 *The semantics of elements of IL are given inductively by the following, where I is an intensional interpretation of IL, $w \in \mathcal{U}$, and A and B are formulas of IL.*

(a) $I(f(e_0, \ldots, e_{n-1}) = I(f)(I(e_0), \ldots, I(e_{n-1})) \in \mathbf{D}$ *where* $f(e_0, \ldots, e_{n-1})$ *is a term.*

If v is a variable, then $I(v) \in \mathbf{D}$.

(b) *For any n-ary predicate p and terms $e_0, \ldots, e_{n-1}$, $\models_{I,w} p(e_0, \ldots, e_{n-1})$ iff $< I(e_0), \ldots, I(e_{n-1}) > \in I(p)(w)$.*

(c) $\models_{I,w} \neg A$ *iff* $\not\models_{I,w} A$.

(d) $\models_{I,w} A \wedge B$ *iff* $\models_{I,w} A$ *and* $\models_{I,w} B$.

(e) $\models_{I,w} (\forall x)A$ *iff* $\models_{I[d/x],w} A$ *for all* $d \in \mathbf{D}$.

Furthermore, $\models_I A$ means that A is true in I (at all worlds) and $\models A$ means that A is true in any interpretation. Keep in mind that $\models$ relation is defined relative to *IL*.

This definition is by no means complete. We must define the semantics of intensional operators of the language. For instance, consider the classical intensional (modal) operators *necessary* $\Box$ and *possible* $\Diamond$. In Kripke-style semantics for intensional logic, the meanings of $\Box$ and $\Diamond$ are determined by an *accessibility* relation R over $\mathcal{U}$. Informally, $\Box A$ is true at a world w iff A is true at all worlds accessible from w; and $\Diamond A$ is true at a world w iff A is true at some world accessible from w. More formally,

- $\models_{I,w} \Box A$ iff $\models_{I,v} A$ for all $v \in \mathcal{U}$ where $< w, v > \in R$
- $\models_{I,w} \Diamond A$ iff $\models_{I,v} A$ for some $v \in \mathcal{U}$ where $< w, v > \in R$

where I is an intensional interpretation, and $w \in \mathcal{U}$. Note that $\Diamond A$ and $\neg\Box\neg A$ are logically equivalent; in other words, $\Diamond$ is the dual of $\Box$.

If R is reflexive, transitive and symmetrical, i.e., $R = \mathcal{U} \times \mathcal{U}$, this gives a Kripke-style semantics for modal logic *S5* [HC68]. Then we can simplify the semantics of *S5* modalities $\Box$ and $\Diamond$ as follows:

- $\models_{I,w} \Box A$ iff $\models_{I,v} A$ for all $v \in \mathcal{U}$
- $\models_{I,w} \Diamond A$ iff $\models_{I,v} A$ for some $v \in \mathcal{U}$

However, the traditional Kripke approach is too restrictive, because it limits us to a dual pair of intensional operators. We could extend it in the obvious way, by allowing a family of dual pairs with its own accessibility relation; see [Gol87].

This is better, but still not truly general, because, as Scott [Sco70] and others have pointed out, there are many natural intensional operators that cannot be defined in terms of an accessibility relation alone. Since we would like to develop a language-independent theory for intensional logic programming, there is no reason why we should restrict ourselves to those logics for which a Kripke-styke semantics is possible.

There are more general approaches to the semantics of intensional logic, including "neighborhood" semantics of Scott [Sco70] and Montague [Mon74]. For a detailed exposition of more general approaches and their relative strengths, we refer the reader to the literature, e.g., see [Woj88] and [BS84]. Neighborhood semantics will provide us with an abstract characterisation of intensional operators which we can exploit further to study the mathematical properties of intensional logics under discussion. Later in the thesis, we will use neighborhood semantics as the basis of our theory, but we will also make use of Kripke-style of semantics for illustrative purposes.

Chapter 3

Intensional/Temporal Logic Programming

In this chapter, we will first formally define intensional logic programs based on a given intensional logic, and then introduce particular intensional logic programming languages, including the temporal language Chronolog originally proposed by Wadge [Wad85]. Each language will be tailored to suit a certain kind of application. For instance, Chronolog is intended for modeling the notion of dynamic change in time and expressing dataflow style of non-terminating computations. We will pay special attention to temporal logic programming (TLP) and especially to Chronolog for its simplicity.

We will later develop the declarative semantics of temporal logic programs of Chronolog in the style of van Emden and Kowalski [vEK76], based on the notion of canonical temporal ground atoms and temporal Herbrand interpretations. We will in particular show that the minimum temporal Herbrand model of a temporal logic program exists and can also be characterized as the least fixpoint of a certain mapping between the temporal Herbrand interpretations of the program. However, this declarative semantics lacks the generality required to consider intensional logic

programming languages based on arbitrary intensional logics. We will also outline several rules of inference for temporal logic programs, which can be used to devise a proof procedure for TLP.

3.1 Intensional Logic Programs

Let *IL* denote an intensional logic equipped with a set of intensional operators, and a set of possible worlds denoted by $\mathcal{U}$. We start by defining an intensional logic program to be a set of intensional Horn clauses. The basic building blocks in an intensional logic program are called *intensional units.*

Definition 3.1 *An intensional unit is defined inductively as follows.*

- *All atomic formulas are intensional units.*
- *If* $A_0 \ldots A_{n-1}$ *are intensional units and* $\bigtriangledown$ *is an n-ary intensional operator, then* $\bigtriangledown(A_0 \ldots A_{n-1})$ *is an intensional unit.*

Throughout, we will adopt the clausal notation [Kow79] for intensional logic programs. All variables in a given clause are assumed to be universally quantified. For convenience, we will use upper-case letters for variables, and lower-case letters for function and predicate symbols.

Definition 3.2 (Intensional Horn clauses.) *Let* A *and* $B_0 \ldots B_{n-1}$ *be intensional units.*

- *An intensional program clause is a clause of the form* $A \leftarrow B_0, \ldots, B_{n-1}$ $(n \geq 0)$.
- *An intensional goal clause is of a clause of the form* $\leftarrow B_0, \ldots, B_{n-1}$ $(n > 0)$.
- *An intensional Horn clause is either an intensional program clause or an intensional goal clause.*

The informal semantics of an intensional program clause $A \leftarrow B_0, \ldots, B_{n-1}$ is defined as follows: at all worlds $w \in \mathcal{U}$, if all of $B_0, \ldots, B_{n-1}$ are true at w, then A is true at w; the difference from logic programs being the notion of truth at all possible worlds.

An intensional logic program therefore consists of the conjunction of a set of intensional program clauses regarded as assertions true at all worlds in $\mathcal{U}$.

Definition 3.3 *An intensional logic program is a finite set of intensional program clauses.*

In the following, we will describe three intensional logic programming languages including a temporal language, Chronolog originally proposed by Wadge [Wad85], as well as two multi-dimensional intensional languages. In each case, the underlying intensional logic will be briefly explained by defining a set of possible worlds over which the values of formulas vary and a set of intensional operators of the language. We will give sample intensional logic programs including one for Conway's game of life.

3.2 Chronolog — Temporal Logic Programming

Temporal logic can be regarded as a special case of intensional logic where the set of possible wolds $\mathcal{U}$ models a collection of moments in time, usually discrete, linearly ordered, and without a last moment. The temporal logic of the temporal language Chronolog [Wad85] has two temporal operators, `first` and `next`, inspired by those of the dataflow language Lucid [WA85], which refer to the initial and the next moment in time respectively. Here the collection of moments in time is the set ω of natural numbers.

A temporal interpretation I basically assigns meanings to all elements of the language at all moments in time in ω (for details, see definition 2.22.) To define the

semantics of the temporal operators **first** and **next**, let $R_f = \{< t, 0 > | t \in \omega\}$ and $R_n = \{< t, t+1 > | t \in \omega\}$ be the accessibility relations associated with **first** and **next**. Formally, the semantics of formulas of the form **first**A and **next**A are defined as follows:

- $\models_{I,t}$ **first** A iff $\models_{I,x} A$ for all $< t, x > \in R_f$.
- $\models_{I,t}$ **next** A iff $\models_{I,x} A$ for all $< t, x > \in R_n$.

where I is a temporal interpretation, and $t \in \omega$. It is not hard to see that **first** and **next** are the necessity operators corresponding to the accessibility relations R_f and R_n (both **first** and **next** are universal-type operators). Notice that these relations are single-valued; they are functions, namely, $\lambda t.0$ and $\lambda t.t+1$. This means that **first** and **next** are also the possibility operators corresponding to these relations. They are therefore selfdual: for example, $\neg$**first**$\neg A \leftrightarrow$ **first**A. Thus we can further simplify the semantics of **first** and **next** as follows.

- $\models_{I,t}$ **first** A iff $\models_{I,0} A$.
- $\models_{I,t}$ **next** A iff $\models_{I,t+1} A$.

We now give an example to illustrate how a Chronolog program works. The following Chronolog program from [Wad88] defines the predicate `fib` which is true of $t+1^{th}$ Fibonacci number at time t for all $t \in \omega$. In other words, it represents the infinite stream of Fibonacci numbers over the Chronolog time.

```
first fib(0).
first next fib(1).
next next fib(N) <- next fib(X), fib(Y), N is X+Y.
```

Read all clauses as assertions true at all moments in time. Thus the first two clauses define the first two Fibonacci numbers as `0` and `1`; the last clause defines the current Fibonacci number as the sum of the previous two.

Temporal logic programming has the potential for specifying non-terminating computations naturally. For instance, a query like `<- fib(X)` may trigger an attempt to prove `fib(X)` at all moments in time, since it actually stands for an infinite series of queries, `<- first fib(X)`, `<- first next fib(X)`, `<- first next next fib(X)`, and so forth. At any moment in time, the answer to the query is a ground atom of the form `first next`^{t} `fib`(e) where e is the $t+1^{th}$ Fibonacci number.

It is possible to specify non-terminating computations in infinitary (concurrent) logic programming languages such as Concurrent Prolog [Sha87] and Parlog [CG86]. Shapiro [Sha87] gives the following program in Concurrent Prolog which specifies the predicate `fib` as true of the infinite list of all Fibonacci numbers.

```
fib([0,1,L]) <- fibtest([0,1,L]).
fibtest([X,Y,Z|L]) <- Z is X+Y, fibtest([Y,Z|L]).
```

Given a query like `<- fib(L)`, an implementation of infinitary logic programming produces an infinite sequence of partial answers to the query, i.e., the initial segments of the infinite list of Fibonacci numbers, and never terminates. However, these partial answers are not justified by the least (minimum) model semantics. As mentioned in [vEdLF82] and [vENA84], the minimum Herbrand model over the usual Herbrand base of such a program is empty, and the intended meaning of this program can be modeled by the greatest fixpoint semantics, based on the notion of Herbrand universe extended with infinite terms. In contrast, Chronolog programs have the minimum model semantics owing to the abstraction of the notion of time in the underlying temporal logic.

Temporal logic programs could be transformed into ordinary Horn logic programs by adding an extra time parameter to *every* predicate; then we would have to simulate the effect of temporal operators **first** and **next** by explicitly manipulating these time parameters. For instance, the following logic program is the result of transforming the temporal program that defines the predicate `fib`.

```
fib(0,0).
fib(next(0),1).
fib(next(next(T)),N) <- fib(next(T),X), fib(T,Y), N is X+Y.
```

where `0` is the initial moment in time and `next`(X) is the successor of X in the corresponding binary relation for the temporal operator `next`. Here `fib`(t, n) means n is the $t + 1^{th}$ Fibonacci number.

However, this transformation is not correct. For instance, the clause `p(X) <-` gets transformed into `p(T,X) <-` which is incorrect, because there are no restrictions on what an implementation can substitute for `T` and `X`. Therefore a correct transformation must treat time parameters differently by introducing an extra domain predicate `time` with an obvious interpretation. Then the correct transformation of the clause would be `p(T,X) <- time(T)` with a complete axiomatisation of `time`. The transformed program above is just like a non-list solution to producing Fibonacci numbers in which the binary predicate `fib` represents Fibonacci numbers paired with their ranks.

But the transformational approach is not really practical. There are several problems: First, the extra time parameters make the programs harder to read and write, and introduce extra opportunities for errors. Second, because of the introduction of an extra domain predicate, the search space for any given query becomes larger. Last, the reduced program would be run on a general purpose logic programming interpreter which would not treat added time parameters differently when looking for a solution for a given query. People use temporal/modal logics to single out important concepts and to investigate their properties, even though, in many cases, they can be reduced to ordinary predicate logic with a similar technique. (Note that some non-classical logics do not admit first-order translations; thus the transformational appproach is not always feasible. One such example is the multi-dimensional logic where the universe is taken as $\mathcal{Z}^\omega$.) Therefore we advocate using non-classical

logics to write programs in the first place.

We claim that Chronolog is also suited for modeling the notion of dynamic change. The following Chronolog program from [OW88a] involves states and dynamic relationships between states. Let the binary predicate cpu represent the state of a cpu with the following properties. The first parameter of cpu shows the status of the cpu (either idle or the current job's name) and the second parameter of cpu indicates how many more units of time the current job will run. The cpu is available at the next moment if either its status is idle or the execution of the current job has ended.

```
first cpu(idle,0).
next cpu(idle,0) <- cpu(S,0), job-queue([]).
next cpu(X,N) <- cpu(S,0), job-queue([[X,N]|R]).
next cpu(S,N) <- cpu(S,s(N)).
```

The first clause in the program says that the cpu is available at time 0. The second clause says that the cpu becomes idle at the next moment if the job queue is currently empty and the cpu is available. The rest of the clauses say that the cpu is busy at the next moment if either (1) the job queue is not empty and the cpu is available or (2) the current job requires more cpu time. Let us extend this program with the following definition of the job-queue where no new job arrivals are considered.

```
first job-queue([[a,s(0)],[b,s(s(0))]]).
next job-queue(L) <- cpu(S,0), job-queue([J|L]).
next job-queue(L) <- cpu(S,s(N)), job-queue(L).
next job-queue([]) <- job-queue([]).
```

The first clause says that initially there are two jobs in the queue with time units 1 and 2 respectively. The second clause says that if the cpu is about to run the first job in the queue, it is removed from the queue at the next moment. If the cpu

has not finished running the current job, the queue will not change (third clause.) According to the extended program, cpu is true of <idle,0> at time 0; <a,s(0)> at time 1 and so forth.

Besides its application to non-terminating computations and modeling dynamic relationships, Chronolog can be applied to a wide range of problems: Rolston [Rol87] proposes the use of Chronolog to alleviate the frame problem. Mitchell and Faustini [MF89] use Chronolog to model simulation tasks. Orgun and Wadge [OW91] suggest that Chronolog is suitable to solve mutual exclusion problems, and give the dining philosophers problem as an example; they also claim that Chronolog extended with modular logic programming of Fitting [Fit87] can be used to specify objects with internal memory.

3.3 Spatial Logic Programming

We now introduce another ILP language, which is based on a kind of two-dimensional (spatial) logic. The underlying logic has $\mathcal{Z} \times \mathcal{Z}$ as the set of possible worlds $\mathcal{U}$ where $\mathcal{Z}$ is the set of integers, and six intensional operators. We regard $\mathcal{Z} \times \mathcal{Z}$ as a collection of (x,y)-coordinates of a plane (or grid) with an absolute reference point $< 0,0 >$, analogous to 0 in the temporal logic of Chronolog. We are not going to make use of accessibility relations in defining the semantics of intensional operators of the logic.

Intensional (spatial) operators are **side**, **edge**, **north**, **south**, **west** and **east**. The semantics of these operators are given in terms of the satisfaction relation $\models$ as follows. Let A be a formula, $< x, y > \in \mathcal{U}$ and I a spatial interpretation.

- $\models_{I,<x,y>}$ **side** A iff $\models_{I<0,y>} A$.
- $\models_{I,<x,y>}$ **west** A iff $\models_{I<x-1,y>} A$.
- $\models_{I,<x,y>}$ **east** A iff $\models_{I<x+1,y>} A$.

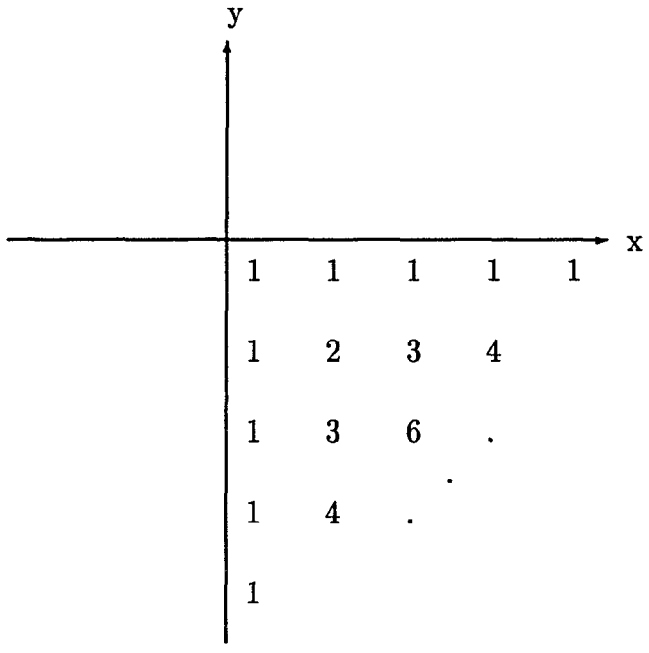

Figure 3.1: Pascal's triangle on the south-east quadrant

Similarly, the operators **edge**, **north** and **south** operate on the y-coordinate of a given world. The analogous operator of the temporal operator **first** is simply **side edge** (or **edge side**); **side edge** takes any world to the origin, i.e., $< 0, 0 >$. Note that the accessibility relations associated with these operators are functions like those of **first** and **next**. Therefore all of the spatial operators are selfdual as well.

We will now show how spatial logic programming can be used to specify such spatial relations on a 2-dimensional plane as Pascal's triangle. The following spatial logic program defines the predicate **pascal**.

```
side edge pascal(1).
side pascal(X) <- side north pascal(X).
edge pascal(X) <- edge west pascal(X).
pascal(X) <- north pascal(Y), west pascal(Z), X is Y+Z.
```

If these are the only axioms for the **pascal** predicate, Pascal's triangle is constructed

on the south-east quadrant, whose apex is at $< 0,0 >$, i.e., at $< 0,0 >$, `pascal(1)` is true from the first clause. Figure 3.1 shows an approximate graphic representation of Pascal's triangle as specified by the `pascal` predicate.

This language is in fact more expressive (powerful) than Chronolog, because any Chronolog program can be rewritten as a spatial program by replacing all temporal operators by their spatial counterparts over one of the spatial dimensions.

3.4 3-Dimensional Logic Programming

Let us combine the two languages we have introduced in the preceeding sections to obtain a (3-dimensional) ILP language [OW88b]. The underlying logic now has $\mathcal{Z} \times \mathcal{Z} \times \omega$ as the set of possible worlds ($\mathcal{U}$), and obvious analogs of all the intensional operators defined previously. A triple $< x, y, z > \in \mathcal{U}$ is interpreted as representing the coordinates of some world where $< x, y >$ refers to the location of the world on a plane and z refers to a moment in time. Of course, all intensional operators work on their respective coordinates.

Let I be a 3-dimensional interpretation, $< x, y, z > \in \mathcal{U}$, and A be a formula. Then the semantics of intensional operators of the language are given in terms of the satisfaction relation $\models$ as follows.

- $\models_{I,<x,y,z>}$ **first** A iff $\models_{I,<x,y,0>} A$.
- $\models_{I,<x,y,z>}$ **next** A iff $\models_{I,<x,y,z+1>} A$.
- $\models_{I,<x,y,z>}$ **side** A iff $\models_{I<0,y,z>} A$.
- $\models_{I,<x,y,z>}$ **west** A iff $\models_{I<x-1,y,z>} A$.

and so on. The obvious analog of the temporal operator **first** can be obtained as **first side edge** (or any combination of these three operators.)

Now we will give an example of 3-dimensional intensional logic programming from [OW88b]. Perhaps Conway's game of life is one of the best examples which require relative references to the neighbours of a point in space at different moments in time. The game involves a (possibly infinite) plane divided into grids. Inside each grid (or cell) resides an organism that may become alive or dead depending on the status of its immediate neighbours in the surrounding cells on the plane. The game starts with an initial configuration on the plane in which some of the organisms are alive.

Supposing the initial configuration is defined elsewhere, the program given below describes all relationships and state changes in the game. All program clauses are interpreted as assertions true at all worlds.

```
next organism(alive) <-
     neighbours(L),count-alive(L,2).
next organism(alive) <-
     organism(alive),neighbours(L),count-alive(L,3).
next organism(dead) <- neighbours(L),lonely(L).
next organism(dead) <- neighbours(L),overcrowded(L).
neighbours([X1,X2,X3,X4,X5,X6,X7,X8]) <-
     north west organism(X1),north organism(X2),
     north east organism(X3),east organism(X4),
     south east organism(X5),south organism(X6),
     south west organism(X7),west organism(X8).
lonely(L) <- count-alive(L,X),X < 2.
overcrowded(L) <- count-alive(L,X),X > 3.
count-alive([],0).
count-alive([alive|L],N) <- count-alive(L,X),N is X+1.
count-alive([dead|L],X) <- count-alive(L,X).
```

We will briefly explain what the first five clauses in the program mean. The first clause says that an organism will become alive at the next moment if exactly two of its neighbours are alive at the current moment no matter if it is alive or dead. This clause also covers the case where the birth of an organism will occur at the next moment if it is dead and exactly two of its neighbours are alive at the current moment. The second clause says that an alive organism will continue to live at the next moment if exactly three of its neighbours are alive at the current moment. The next two clauses state that an organism will become dead at the next moment if it is lonely (less than two neighbours are alive) or the surrounding area is overcrowded (more than three neighbours are alive). The fifth clause simply bundles up the status of the neighbours of a given cell in a list for further use. The rest of the clauses define some auxiliary predicates. According to the program, note that at any world exactly one of the atoms `organism(alive)` and `organism(dead)` is true.

3.5 Semantics of Temporal Logic Programming

We can design an intensional logic programming language for a given problem domain by identifying a set of possible worlds (contexts) and by providing a set of context-switching operators. The next logical step is to investigate the meaning of programs written in such a language. There are two alternative ways to go about doing this: (1) we focus on the language in question and provide a semantics for it, and (2) we investigate general properties of intensional languages and develop a theory which can be applied to different intensional languages. In this section, we will focus on the temporal language Chronolog. The more general approach will be discussed later.

In this section, we will develop the model-theoretic semantics of temporal logic programs of Chronolog by adopting the notion of a canonical temporal ground atom.

'Thus the term "temporal logic program" will always refer to programs in Chronolog. Preliminary results of this section have first appeared in the following research reports: [OW88a] and [OW91]. Th issues concerning the operational semantics of temporal logic programs will not be addressed, because developing operational semantics is not within the scope of this dissertation and furthermore implementation problems are discussed in another forthcoming dissertation by David Rolston of Arizona State University [Rol86].

3.5.1 The Underlying Temporal Logic

From here on, the underlying temporal language in which temporal logic programs are written wil' _e referred to as *TL*. We begin with a standard first-order language and extend it with two temporal operators `first` and `next`. We add new formation rules: if A is a formula, so are `first`A and `next`A. Note that the temporal operators `first` and `next` are applied to formulas, not to terms of the language.

As said before, the value of a formula (or an element) of *TL* depends on an implicit time parameter. Since here ω is considered as the collection of moments in time, the value of a formula is regarded as a collection of truth values indexed by the elements of ω. A temporal interpretation assigns meanings to all basic elements of the language, i.e., function symbols, predicate symbols and variables, and a satisfaction relation completes this picture by extending temporal interpretations upward to all elements of the language, e.g., terms, formulas, etc.

Axioms of Temporal Logic

The following are axioms of the underlying logic related to the temporal operators `first` and `next`, and they are considered as statements true at all moments in time. These axioms state some important properties of the temporal operators. Let A and B be formulas of the language.

A1. $\texttt{first}\ \texttt{first}\, A \leftrightarrow \texttt{first}\, A$.

A2. $\texttt{next}\ \texttt{first}\, A \leftrightarrow \texttt{first}\, A$.

A3. $\texttt{first}(A \wedge B) \leftrightarrow (\texttt{first}\, A) \wedge (\texttt{first}\, B)$.

A4. $\texttt{next}(A \wedge B) \leftrightarrow (\texttt{next}\, A) \wedge (\texttt{next}\, B)$.

A5. $\texttt{first}(\neg A) \leftrightarrow \neg(\texttt{first}\, A)$.

A6. $\texttt{next}(\neg A) \leftrightarrow \neg(\texttt{next}\, A)$.

A7. $\texttt{first}(\forall X)(A) \leftrightarrow (\forall X)(\texttt{first}\, A)$.

A8. $\texttt{next}(\forall X)(A) \leftrightarrow (\forall X)(\texttt{next}\, A)$.

The first two axioms, A1 and A2, are concerned with the temporal operator cancellation rules: they say that when applied to a formula of the form first$A$ (or to initial truths), both **first** and **next** are redundant. Axioms A3 through A6 capture the fact that the Boolean connectives work pointwise in time. And axioms A7 and A8 reflect the fact that the values of individual variables range over extensions (data values), not intensions (time-varying values).

3.5.2 Models of Temporal Logic Programs

This section focuses on the model-theoretic semantics of temporal logic programs. The theoretical foundations developed for logic programming [vEK76] [Llo84] will be followed and the discussion will be restricted to (temporal) Herbrand interpretations and models. Since all variables in a temporal Horn clause are assumed to be universally quantified, the meaning of the clause with respect to some interpretation I is independent of what I assigns to variables.

According to the definition of intensional logic programs, an intensional (temporal) unit of temporal logic consists of an atomic formula with a number (possibly

zero) of temporal operators **first** and **next** applied to it. The following are temporal units:

next p(X,f(X)), **next first first q(a)**, **r(X)**

We use upper-case characters for variables, and lower-case characters for predicates, constants and function symbols. A temporal logic program consists of the conjunction of a set of definite temporal clauses (or a set of program clauses).

We are interested in those interpretations I of a temporal logic program $\mathcal{P}$ which makes the program true at all moments in time, i.e., I is a model of $\mathcal{P}$ iff for all $t \in \omega$, $\models_{I,t} \mathcal{P}$. From here on, we denote this fact by $\models_I \mathcal{P}$. Since a temporal logic program $\mathcal{P}$ is the conjunction of a set of temporal program clauses, we also say that I is a model for $\mathcal{P}$ iff I is a model for each clause in $\mathcal{P}$.

Temporal Herbrand Interpretations

In logic programming theory [Llo84], Herbrand interpretations are regarded as subsets of the Herbrand base of a given logic program. In temporal logic, the meaning of a formula depends on the elements of ω, and therefore, in general, temporal Herbrand interpretations do not enjoy a set-theoretical representation. On the other hand, the temporal logic of Chronolog has the temporal operators **first** and **next** through which a set-theoretical representation of temporal Herbrand interpretations can be restored.

Throughout, we write $\mathbf{next}^t$ for t successive applications of **next**. We will now introduce the notion of a canonical temporal atom.

Definition 3.4 *A canonical temporal atom is a formula of the form* $\mathbf{first}\ \mathbf{next}^t A$ *for some* $t \geq 0$, *where* A *is an atomic formula.*

The meaning of a canonical temporal atom in a temporal interpretation is fixed over the elements of ω owing to the application of **first**. Let **first next nextp** be a

canonical temporal atom and I be a temporal interpretation. Then, for any $t \in \omega$, $\models_{I,t}$ `first next nextp` iff $\models_{I,2}$ p.

The definition of Herbrand universe of a given temporal logic $\mathcal{P}$ program coincides with that of an ordinary logic program. In other words, the Herbrand universe $U_{\mathcal{P}}$ of $\mathcal{P}$ is the set of all ground terms which can be constructed out of constants and functions that appear in $\mathcal{P}$. The domain of a temporal Herbrand interpretation is the Hebrand universe of a given program. The definition of temporal Herbrand base is given as follows.

Definition 3.5 *Let $\mathcal{P}$ be a temporal logic program. The temporal Herbrand base $B_{\mathcal{P}}$ of $\mathcal{P}$ is the set of all canonical temporal ground atoms which can be constructed out of predicates that appear in $\mathcal{P}$ with ground terms in $U_{\mathcal{P}}$ as arguments.*

Each canonical temporal ground atom in $B_{\mathcal{P}}$ refers to a particular moment in time and therefore its meaning is fixed over the elements of ω in a temporal interpretation. It remains to show that temporal Herbrand interpretations are identified with subsets of the temporal Herbrand base.

In a Herbrand interpretation I, the domain of the interpretation is the Herbrand universe $U_{\mathcal{P}}$ and for all $e \in U_{\mathcal{P}}$, $I(e) = e$; therefore we can identify I with a subset H of the temporal Herbrand base $B_{\mathcal{P}}$ by the following.

$$< e_0, \ldots, e_{n-1} > \in I(p)(t) \text{ iff } \texttt{first next}^t\, p(e_0, \ldots, e_{n-1}) \in H$$

For instance, `cpu(a,0)` is true at time 2 in I iff `first next next cpu(a,0)` $\in H$. Since there is a one-to-one correspondence between temporal Herbrand interpretations and subsets of the temporal Herbrand base of a given program, we will refer to any subset of $B_{\mathcal{P}}$ as a temporal Herbrand model. We say that H is a model of $\mathcal{P}$ iff $\models_I \mathcal{P}$ for the I corresponding to H.

We will now extend the notion of canonical temporal atoms to temporal program clauses: a canonical ground instance of a temporal program clause is made of only

canonical temporal ground atoms, and therefore its meaning is fixed over elements of ω.

Definition 3.6 *Let $C = A \leftarrow B_0, \ldots, B_{n-1}$ be a temporal program clause. We say that C is canonical if A and all B_i's are canonical temporal atoms. C is ground if there are no variables in it.*

Let `first p(X)` $\leftarrow$ `next q(X)` be a program clause in some program $\mathcal{P}$. To obtain the canonical ground instances of the clause, first, the temporal operator `first` must be applied to the clause, followed by a number (possibly zero) of `next` in all possible ways. This process results in an infinite set of temporal program clauses whose conjunction is equivalent to the original clause.

$$\{\, \texttt{first next}^t\texttt{(first p(X)} \leftarrow \texttt{next q(X))} \mid t \in \omega \,\}$$

Second, by using the axioms A3–A6, the temporal operators are distributed over all those atoms in the clauses and then all extra operators are eliminated by axioms A1 and A2. Having done that, we obtain the set

$$\{\, \texttt{first p(X)} \leftarrow \texttt{first next}^{t+1}\,\texttt{q(X)} \mid t \in \omega \}$$

Last, the canonical ground instances of the clause are all those that are ground instances of some clause in the set. For instance, `first p(a)` $\leftarrow$ `first next q(a)` is a such ground instance for the $\mathtt{a} \in B_{\mathcal{P}}$.

It is easy to check that only canonical temporal ground atoms can appear in any canonical ground instance of any temporal program clause. We will show that a clause C is true in I iff all of its canonical ground instances are true in I.

Lemma 3.1 *Let $\mathcal{P}$ be a temporal logic program, I be a temporal Herbrand interpretation of $\mathcal{P}$, and C a temporal program clause in $\mathcal{P}$. I is a model of C iff $\models_I C_i$ for all canonical ground instances C_i of C.*

Proof. We will give the outline: $\models_I C$
iff for all $t \in \omega$, $\models_I \texttt{first next}^t C$
iff for all $t \in \omega$, $\models_I \texttt{first next}^t C_\alpha$ for all ground instances C_α of C,
iff for all $t \in \omega$, $\models_I \texttt{first next}^t C_i$ for all canonical ground instances C_i of C. ⊣

By lemma 3.1, a clause C is true in I iff all of its canonical ground instances are true in I. Then if $(A \leftarrow B_0, \cdots, B_{n-1})$ is a canonical ground instance of C, we have that $\models_I (A \leftarrow B_0, \cdots, B_{n-1})$ iff $A \in I$ or for some $i \in n$, $B_i \notin I$. Since the value of a canonical temporal ground atom A is fixed anytime in a temporal Herbrand interpretation H, there is no difference between these two notations: $\models_H A$ and $\models_{H,t} A$ where $t \in \omega$.

The following lemmas justify that temporal Hebrand interpretations are sufficient for the theory of temporal logic programming.

Lemma 3.2 *Let S be a set of temporal Horn clauses and suppose S has a temporal model. Then S has a temporal Herbrand model.*

Proof. Let I be a temporal model of S, i.e., $\models_I S$, and define H by the following:

$$H = \{\texttt{first next}^t A \in B_S \mid \models_{I,t} A\}$$

If I is a model of S, then H is a temporal Herbrand model of S. ⊣

Lemma 3.3 *Let $\mathcal{P}$ be a temporal logic program and $A \in B_\mathcal{P}$. Then $\mathcal{P} \cup \{\neg A\}$ is unsatisfiable in any temporal model of $\mathcal{P}$ iff no temporal Herbrand model of $\mathcal{P}$ satisfies $\mathcal{P} \cup \{\neg A\}$.*

Proof. If $\mathcal{P} \cup \{\neg A\}$ is satisfiable in some intensional model I of $\mathcal{P}$, then for some $t \in \omega$, $\models_{I,t} \mathcal{P} \cup \{\neg A\}$. Thus, by lemma 3.2, a construction from I to the corresponding temporal Herbrand model can be given. ⊣

The Minimum Temporal Herbrand Model

In this section, we will show that every temporal logic program has the minimum model property by showing first that it has a model which corresponds to the top-most interpretation in the lattice of temporal Herbrand interpretations; then by establishing that the intersection of the family of temporal Herbrand interpretations of a given program $\mathcal{P}$ is the minimum temporal Herbrand model of $\mathcal{P}$.

The following lemma states that every temporal logic program has a model, that is, the whole temporal Herbrand base.

Lemma 3.4 *Let $\mathcal{P}$ be a temporal logic program; then $B_{\mathcal{P}}$ is a model of $\mathcal{P}$.*

Proof. Consider any canonical ground instance of any clause in $\mathcal{P}$ and let $A \leftarrow B_0, \cdots, B_{n-1}$ be such an instance. A and each B_i are canonical temporal ground atoms, hence, $A \in B_{\mathcal{P}}$ and for all $i \in n$, $B_i \in B_{\mathcal{P}}$. Then the instance must be true in $B_{\mathcal{P}}$ at all moments in time. By lemma 3.1, we have that $\models_{B_{\mathcal{P}}} \mathcal{P}$. $\dashv$

Now we show that the model-intersection property holds for temporal logic programs.

Lemma 3.5 *Let $\mathcal{P}$ be a temporal logic program and $M = \{I_\alpha\}_{\alpha \in S}$ be a non-empty set of temporal Herbrand models for $\mathcal{P}$. Then $\cap M =_{def} \cap_{\alpha \in S} I_\alpha$ is a temporal Herbrand model for $\mathcal{P}$.*

Proof. First of all, $\cap M$ is a temporal Herbrand interpretation of $\mathcal{P}$. Suppose it is not a model of $\mathcal{P}$. Then there is a canonical ground instance of a clause in $\mathcal{P}$ which is false in $\cap M$ (at some moment.) Let $A \leftarrow B_0, \cdots, B_{n-1}$ be such an instance, i.e., $A \notin \cap M$, but for all $i \in n$, $B_i \in \cap M$. This implies for some $\alpha \in S$, $A \notin I_\alpha$; but since all B_i's are true in $\cap M$, $B_i \in I_\alpha$ for all $i \in n$ as well. Thus we obtain a contradiction that I_α is not a model of $\mathcal{P}$ either. $\dashv$

Since every temporal logic program has at least one model and enjoys the model-intersection property, the intersection of all temporal Herbrand models of a given

temporal logic program exists and it is the minimum temporal Herbrand interpretation of the program.

Theorem 3.6 *Let $\mathcal{P}$ be a temporal logic program. Then $\mathcal{P}$ has a minimum temporal Herbrand model $M_{\mathcal{P}}$, which is the intersection of all temporal Herbrand models of $\mathcal{P}$.*

Proof. Let $M = \{I_\alpha | \models_{I_\alpha} \mathcal{P}\}_{\alpha \in S}$ be the family of all temporal Herbrand interpretations of $\mathcal{P}$. Then M is non-empty by lemma 3.4. Therefore $M_{\mathcal{P}} =_{def} \cap M$ is the minimum temporal Herbrand model of $\mathcal{P}$ by lemma 3.5. $\dashv$

The next theorem shows that the canonical temporal ground atoms in $M_{\mathcal{P}}$ for a given temporal logic program $\mathcal{P}$ are those that are logical consequences of the program. Recall that each canonical temporal ground atom is fixed to a particular moment in time.

Theorem 3.7 *Let $\mathcal{P}$ be a temporal logic program. Then $M_{\mathcal{P}} = \{A \in B_{\mathcal{P}} \mid \mathcal{P} \models A\}$.*

Proof. Let $A \in B_{\mathcal{P}}$. Then A is of the form $\texttt{first next}^t\, p(e_0, \cdots, p_{n-1})$ and is a logical consequence of $\mathcal{P}$
iff $\mathcal{P} \cup \{\neg A\}$ is unsatisfiable in any temporal model of $\mathcal{P}$
iff no temporal Herbrand model of $\mathcal{P}$ satisfies $\mathcal{P} \cup \{\neg A\}$ by lemma 3.2
iff $\mathcal{P} \not\models_{I,t} \neg A$ for all $t \in \omega$ and for any temporal Herbrand model I of $\mathcal{P}$
iff $\mathcal{P} \models_I A$ for all temporal Herbrand models I of $\mathcal{P}$
iff $\mathcal{P} \models_{M_{\mathcal{P}}} A$ by theorem 3.6. $\dashv$

3.5.3 The Fixpoint Semantics of Temporal Logic Programs

The fixpoint semantics of Horn logic programs can be modified for further extensions of the theory of TLP as well. The mapping $T_{\mathcal{P}}$, the one-step modus ponens function, originally given in [vEK76] provides the basis for fixpoint semantics and therefore establishes the connection between the model-theoretic and operational

semantics of ordinary logic programs. Now we generalise the definition of $T_{\mathcal{P}}$ for temporal logic programs. Recall that $\mathcal{F}(\mathcal{P})$ denotes the family of temporal Herbrand interpretations of $\mathcal{P}$.

Definition 3.7 *Let $\mathcal{P}$ be a temporal logic program. For any $H \in \mathcal{F}(\mathcal{P})$, $T_{\mathcal{P}}(H)$ is a temporal Herbrand interpretation of $\mathcal{P}$ given below:*

$$T_{\mathcal{P}}(H) = \{\ A \mid A \leftarrow B_0, \ldots, B_{n-1} \text{ is a canonical ground instance of a clause in } \mathcal{P} \text{ and } \{B_0, \ldots, B_{n-1}\} \subseteq H\ \}$$

The mapping $T_{\mathcal{P}}$ shares all the properties of original $T_{\mathcal{P}}$ for ordinary logic programs, and those of some other non-classical extensions to logic programming such as multiple-valued schemes of Fitting [Fit88], and Blair et al [B$^+$88]. We first show that $T_{\mathcal{P}}$ is continuous.

Lemma 3.8 *Let $\mathcal{P}$ be a temporal logic program. Then $T_{\mathcal{P}}$ is continuous, that is, for any chain $< C_n >_{n\in\omega}$ of temporal Herbrand interpretations of $\mathcal{P}$, $T_{\mathcal{P}}(\bigcup_{n\in\omega} C_n) = \bigcup_{n\in\omega} T_{\mathcal{P}}(C_n)$.*

Proof. The proof is a pointwise extension of that of the continuity of $T_{\mathcal{P}}$ from [Llo84]. Let $\texttt{first next}^i\, p(e_0, \ldots, e_{n-1})$ be any canonical temporal ground atom in $B_{\mathcal{P}}$. Then it suffices to show that $\texttt{first next}^i\, p(e_0, \ldots, e_{n-1}) \in T_{\mathcal{P}}(\bigcup_{n\in\omega} C_n)$ iff $\texttt{first next}^i\, p(e_0, \ldots, e_{n-1}) \in \bigcup_{n\in\omega} T_{\mathcal{P}}(C_n)$. We omit the details. $\dashv$

Temporal Herbrand models can be characterised in terms of $T_{\mathcal{P}}$.

Lemma 3.9 *Let $\mathcal{P}$ be a temporal logic program and I be a temporal Herbrand interpretation of $\mathcal{P}$. Then I is a model of $\mathcal{P}$ iff $T_{\mathcal{P}}(I) \subseteq I$.*

Proof. The proof is similar to that of [Llo84]. $\dashv$

We now give the fixpoint characterisation of the minimum temporal Herbrand model of a temporal logic program.

Theorem 3.10 *Let $\mathcal{P}$ be a temporal logic program. Then $M_{\mathcal{P}} = lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$.*

Proof. Since $T_{\mathcal{P}}$ is continuous by lemma 3.8, the upward closure ordinal of $T_{\mathcal{P}}$ is $\leq \omega$ by lemma 2.8. Thus the least fixpoint of $T_{\mathcal{P}}$, $lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$. We also have that $M_{\mathcal{P}} =_{def} \bigcap\{I \mid \models_I \mathcal{P}\} = \bigcap\{I \mid T_{\mathcal{P}}(I) \subseteq I\}$ by lemma 3.9 and theorem 3.6. By Knaster-Tarski fixpoint theorem (theorem 2.7), $lfp(T_{\mathcal{P}}) = \bigcap\{I \mid T_{\mathcal{P}}(I) \subseteq I\}$. Therefore $M_{\mathcal{P}} = lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$. $\dashv$

3.6 Rules of Inference for Temporal Logic

We have not addressed any proof procedures for any of the languages we described so far. In doing so, we need to know the properties of the underlying logic of each language. Baudinet [Bau88] [Bau89] showed the completeness of a proof procedure for Templog called TSLD-resolution [AM87]. However, Templog lacks the temporal operator `first` and therefore TSLD-resolution cannot be directly applied to Chronolog. In this section, we will lay down some rules of inference for temporal logic which can be used to devise a proof procedure for Chronolog.

We first have all the axioms for the temporal operators `first` and `next` given earlier. Besides substitution and modus ponens, we have the following temporal operator *introduction rules* where A is a formula:

$$\textbf{(Rule 1)}\ \frac{A}{\texttt{first}\ A} \qquad \textbf{(Rule 2)}\ \frac{A}{\texttt{next}\ A}$$

We read these rules as follows: if A is true at all moments in time, then infer `first` A and `next` A. Since both `first` and `next` are necessity operators, these rules may be regarded as instances of *the rule of necessitation* from modal logic [HC68] [Che80].

We will illustrate how these two rules can be used to prove the canonical ground atom `first next next next fib(2)` from the temporal logic program that defines the predicate `fib`. The program is given below.

```
first fib(0).
```

```
first next fib(1).
next next fib(N) <- next fib(X), fib(Y), N is X+Y.
```

First, we apply Rule 2 and Rule 1 to the third clause, and obtain:

```
first(next(next next fib(N) <- next fib(X), fib(Y), N is X+Y.))
```

Then by using the temporal axioms given in Chapter 3, the temporal operators **first** and **next** are distributed over the temporal units in the clause.

```
(C1) first next next next fib(N) <-
        first next next fib(X), first next fib(Y), first next N is X+Y.
```

Similarly, we apply Rule 1 to the third clause and obtain the following conditional clause:

```
(C2) first next next fib(N) <-
        first next fib(X), first fib(Y), first N is X+Y.
```

Now, since we have the first two unconditional clauses in the program, by substitution and modus ponens, we obtain from C2 the canonical ground atom `first next next fib(1)`. Using this result, again by substitution and modus ponens, we obtain from C1 the canonical ground atom `first next next next fib(2)`.

We also have the following temporal operator *elimination rule*:

$$\textbf{(Rule 3)}\ \frac{\texttt{first}\,A \quad \texttt{next}\,A}{A}$$

Rule 3 says "from **first** A and **next** A, infer A". The correctness of these rules can be easily established. A resolution-type proof procedure for Chronolog could be based on these rules of inference. Then we conjecture that such a proof procedure is sound and complete. However, we leave this problem open for further research.

3.7 Discussion

Abadi and Manna of Stanford University [AM87] have proposed a temporal logic programming language, called Templog. We will show that the model-theory of temporal logic programming can be applied to Templog as well. Templog does not have an explicit initial time operator such as `first`, but it is richer than Chronolog, and provides two temporal operators $\Diamond$ and $\Box$ as well as the next time operator $\bigcirc$. Read $\Diamond$ as "sometime in the future," and $\Box$ as "from now on." The semantics of $\Diamond$ and $\Box$ are defined as follows.

- $\models_{I,t} \Diamond A$ iff $\models_{I,z} A$ for some $z \in \omega$ where $t \leq z$
- $\models_{I,t} \Box A$ iff $\models_{I,z} A$ for all $z \in \omega$ where $t \leq z$

where I is a temporal interpretation and $t \in \omega$.

Baudinet [Bau88] [Bau89] has independently developed a model-theoretic semantics for Templog programs based on the notion of a next-atom, i.e., atoms with only next time operator $\bigcirc$ applied to them. There, a ground next-atom is considered as a logical consequence of a given Templog program iff it follows from the program at the initial moment in time, hence the temporal operator `first` is implicit in that approach. Furthermore, in [Bau88] and [Bau89], it is shown that a Templog program is true in a temporal interpretation I iff all strictly ground instances of every program clause $C \in \mathcal{P}$ are true in I. This property can be used to establish the connection between the two model-theoretical approaches.

In our terminology, a strictly ground instance of a program clause free of operators $\Box$ and $\Diamond$ is called a canonical ground instance based on canonical temporal ground atoms. Then all the results apply to Templog, once temporal Herbrand models of a Templog program $\mathcal{P}$ are characterised in terms of strictly ground instances of program clauses that occur in $\mathcal{P}$. Therefore, as conjectured in [OW88a] but not proved, the theory extends to Templog. Note that in this case, the model-theory of

Templog applies to Chronolog as well.

We are also considering an extension to Chronolog which employs the set of integers $\mathcal{Z}$ as the set of possible worlds and another temporal operator **pre** to refer to the previous moment in time. To define the semantics of formulas of the form **pre** A, let $R_p = \{< t, t-1 > | t \in \omega\}$ be the accessibility relation associated with the temporal operator **pre**. Then we have that

- $\models_{I,t}$ **pre** A iff for all $< t, s > \in R_p$, $\models_{I,t-1} A$

where I is a temporal interpretation and $t \in \mathcal{Z}$. Since R_p is a single-valued relation, namely the function $\lambda t.t-1$, the semantics of **pre** can be simplified further.

- $\models_{I,t}$ **pre** A iff $\models_{I,t-1} A$

Note that **pre** is the inverse of **next** and therefore we have the following two axioms:

A9. **pre next** $A \leftrightarrow A$.

A10. **next pre** $A \leftrightarrow A$.

Furthermore, **pre** shares all of the properties of **next**. For instance, **pre** is self-dual as well.

In order to give semantics to Chronolog programs with negative time, we need to modify the definitions of canonical ground atoms, temporal Herbrand base, and temporal Herbrand interpretations. In this case, canonical temporal atoms are formulas of the form **first next**$^t A$ and **first pre**$^t A$ where A is an atomic formula and $t \geq 0$. Then the theory readily extends to Chronolog with negative time. This is fine, but what about the model-theoretical semantics of the 2-dimensional and 3-dimensional languages introduced in the beginning of this chapter? Another possibility is of course Chronolog extended with $\Diamond$ and $\Box$, and so on.

Then the question is "does this semantics based on canonical temporal ground atoms apply to other temporal/intensional logic programming languages?" The

answer is, in general, a firm "no", because there is no guarantee that we can always find canonical intensional atoms whose meanings are fixed over, or independent of, the elements of the set of possible worlds and yet in terms of which intensional Herbrand models can be characterised. Thus we must explore general conditions and constraints on intensional logic programming languages without referring to a particular one such as Chronolog. However, the loss in generality of this approach is made up for its simplicity and a smoother theory.

Chapter 4

Intensional Semantics

This chapter lays down the groundwork for a generalized model-theoretical investigation of intensional logic programming. In the preceding chapter, we have defined the semantics of intensional operators by first stating the structure of the set of possible worlds, and then by providing a satisfaction relation $\models$ for each intensional language. We have also employed Kripke-style semantics for some of the intensional languages. In Kripke-style semantics for intensional logic, the semantics of intensional operators are usually defined in terms of accessibility relations [Gol87].

From here on in the dissertation, we will employ a more general approach attributed to Scott [Sco70] and Montague [Mon74], called *neighborhood semantics*, because we would like our theory to be general enough to apply to intensional logics for which a Kripke-style semantics is not possible. Moreover, Scott-Montague semantics will provide us with an abstract characterization of intensional operators, according to which intensional operators receive a denotation reflecting the mathematical properties of the intensional logic under discussion. In this chapter, we will study several important properties of intensional operators which will be used as semantic constraints on intensional logic programming languages. We will also show how a language can be extended with extra intensional operators defined in

terms of monotonic formulas of the language.

4.1 Semantics of Intensional Operators

Let *IL* denote the underlying intensional language of an intensional logic. The values of formulas of *IL* vary with the $w \in \mathcal{U}$ in a given intensional interpretation. Let $\|A\|^I$ denote the meaning of a formula A of the language given by the intensional interpretation I. As A may have different values in I at different possible worlds, $\|A\|^I$ is really a function, which, given a possible world in $\mathcal{U}$, returns the value of A at that world. In order to make the values visible, we will write 0 for *false* and 1 for *true*. More formally, we conclude that $\|A\|^I$ is an element of $[\mathcal{U} \rightarrow 2]$ where $2 = \{0,1\}$. Note that $\|A\|^I$ is also called an *intension* which gathers the values (extensions) of A at all possible worlds. The intension $\|A\|^I$ can also be viewed as a subset of $\mathcal{U}$, whose elements are all those possible worlds at which A is true in I, i.e., $\|A\|^I = \{w \in \mathcal{U} \mid \models_{I,w} A\}$. Of course, we do not claim that every subset of $\mathcal{U}$ is represented by some formula. However, the reverse claim is made, i.e., each formula represents a subset of $\mathcal{U}$.

Note that $[\mathcal{U} \rightarrow 2]$, or equivalently $P(\mathcal{U})$, together with the usual set operations and a complementation operation relative to $P(\mathcal{U})$, is a complete Boolean algebra (CBA), denoted by $< P(\mathcal{U}), \emptyset, \mathcal{U}, \neg, \cap, \cup >$. We also have that $P(\mathcal{U})$ is a complete lattice denoted by $< P(\mathcal{U}), \subseteq >$ whose elements are ordered by the subset relation $\subseteq$.

Intensional operators take formulas as their arguments, and the denotations of formulas are intensions, which suggest that the denotations of intensional operators can be thought of as mappings from intensions to intensions. If ∇ is a unary operator of *IL*, its denotation must be a function, an element of $[P(\mathcal{U}) \rightarrow P(\mathcal{U})]$ in the underlying algebra, which is independent of intensional interpretations of *IL*. Then

the definition of the satisfaction relation $\models$ can be extended to assign meanings to formulas of the form $\bigtriangledown A$ as follows.

$$\models_{I,w} \bigtriangledown A \text{ iff } w \in \|\bigtriangledown\|(\|A\|^I)$$

In orher words, $\|\bigtriangledown\|$ is a function in the underlying algebra associated with the symbol $\bigtriangledown$. One way of interpreting the condition $w \in \|\bigtriangledown\|(\|A\|^I)$ is that w can see all the worlds in the intension $\|A\|^I$ through $\|\bigtriangledown\|$. Moreover, in this approach, there is no need to refer to the accessibility relation associated with $\bigtriangledown$ or even to require that there is one.

In particular, the semantics of classical operators $\neg$, $\wedge$ and $\vee$ correspond to the complementation operation, intersection and union respectively. Let I be an intensional interpretation and $w \in \mathcal{U}$.

- $\models_{I,w} \neg A$ iff $w \in \neg\|A\|^I$
- $\models_{I,w} A \wedge B$ iff $w \in \|A\|^I \cap \|B\|^I$
- $\models_{I,w} A \vee B$ iff $w \in \|A\|^I \cup \|B\|^I$

We will also use the symbol $\|\neg\|$ to refer to the complementation operation. The usual definition of the implication operation is assumed, i.e., $A \rightarrow B =_{def} \neg A \vee B$.

In general, *IL* can be enriched with an extra n-ary operator, say $\bigtriangledown$, by letting $\| \bigtriangledown \| = \Theta$ for some element $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$, and by extending the definition of the satisfaction relation $\models$ in order to give meanings to formulas of the form $\bigtriangledown(A_0, \ldots, A_{n-1})$. More formally,

Definition 4.1 *Let $\bigtriangledown$ be an n-ary intensional operator where $\| \bigtriangledown \| \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. Then the semantics of the formulas of the form $\bigtriangledown(A_0, \ldots, A_{n-1})$ are given as*

$$\models_{I,w} \bigtriangledown(A_0, \ldots, A_{n-1}) \text{ iff } w \in \|\bigtriangledown\|(\|A_0\|^I, \ldots, \|A_{n-1}\|^I)$$

for any intensional interpretation I and any $w \in \mathcal{U}$.

The denotations of $\Box$ and $\Diamond$ have the same logical type as $\|\neg\|$; in other words, both $\|\Box\|$ and $\|\Diamond\|$ are elements of $[P(\mathcal{U}) \rightarrow P(\mathcal{U})]$. We would also like to know how these two functions operate, i.e., what kind of a transformation they do on a given intension. The definition of the satisfaction relation $\models$ can be utilized to obtain the functions $\|\Box\|$ and $\|\Diamond\|$ as follows. Consider the semantics of a formula of the form $\Diamond A$ in the intensional interpretation I, i.e., $\|\Diamond A\|^I = \|\Diamond\|(\|A\|^I)$. Let R be the accessibility relation associated with $\Box$ and $\Diamond$. Then we have that

$$\begin{aligned}
\|\Diamond\|(\|A\|^I) &= \{w \in \mathcal{U} \mid \models_{I,w} \Diamond A\} \\
&= \{w \in \mathcal{U} \mid \models_{I,v} A \text{ for some } v \in \mathcal{U} \text{ where } < w, v > \in R\} \\
&= \{w \in \mathcal{U} \mid v \in \|A\|^I \text{ for some } v \in \mathcal{U} \text{ where } < w, v > \in R\}
\end{aligned}$$

We have that $\|A\|^I \in P(\mathcal{U})$; then the definition of $\|\Diamond\|$ can be obtained from the above expression by lambda abstraction.

$$\begin{aligned}
\|\Diamond\| &= \lambda X.\{w \in \mathcal{U} \mid v \in \|A\|^I \text{ for some } v \in \mathcal{U} \text{ where } < w, v > \in R\} \\
&= \lambda X.\{w \in \mathcal{U} \mid X \cap \{v \in \mathcal{U} \mid < w, v > \in R\} \neq \emptyset\}
\end{aligned}$$

The function $\|\Box\|$ can be formed in a similar fashion. But we leave out the intermediate steps.

$$\|\Box\| = \lambda X.\{w \in \mathcal{U} \mid X \supseteq \{v \in \mathcal{U} \mid < w, v > \in R\}\}$$

As for *S5* modalities, since we take $R = \mathcal{U} \times \mathcal{U}$ in *S5*, the definitions of $\|\Box\|$ and $\|\Diamond\|$ can be simplified further:

$$\|\Diamond\| = \lambda X.\{w \in \mathcal{U} \mid X \neq \emptyset\}$$

$$\|\Box\| = \lambda X.\{w \in \mathcal{U} \mid X = \mathcal{U}\}$$

We will now construct the denotations of the temporal operators `first` and `next`, i.e., the functions $\|\texttt{first}\|$ and $\|\texttt{next}\| \in [P(\omega) \rightarrow P(\omega)]$. Consider the semantics of a formula of the form `first` A in a temporal interpretation I, i.e.,

$\|\texttt{first}\, A\|^I = \|\texttt{first}\|(\|A\|^I)$. Then, from the definition of the satisfaction relation $\models$, the function $\|\texttt{first}\| \in [P(\omega) \to P(\omega)]$ can be obtained as follows:

$$\begin{aligned} \|\texttt{first}\|(\|A\|^I) &= \{\, t \in \omega \mid \models_{I,t} \texttt{first}\, A \,\} \\ &= \{\, t \in \omega \mid \models_{I,0} A \,\} \\ &= \{\, t \in \omega \mid 0 \in \|A\|^I \,\} \end{aligned}$$

Since $\|A\|^I$ is an arbitrary element of $P(\omega)$, we can obtain the definition of the function $\|\texttt{first}\|$ from the above expression by lambda abstraction.

$$\|\texttt{first}\| = \lambda X.\{t \in \omega \mid 0 \in X\}$$

which conforms to the semantics of **first** in terms of the satisfaction relation $\models$ given earlier.

Similarly, the function $\|\texttt{next}\| \in [P(\omega) \to P(\omega)]$ can be formed as:

$$\begin{aligned} \|\texttt{next}\|(\|A\|^I) &= \{\, t \in \omega \mid \models_{I,t} \texttt{next}\, A \,\} \\ &= \{\, t \in \omega \mid \models_{I,t+1} A \,\} \\ &= \{\, t \in \omega \mid t+1 \in \|A\|^I \,\} \end{aligned}$$

Hence, by lambda abstraction, the definition of $\|\texttt{next}\|$ can be obtained as

$$\|\texttt{next}\| = \lambda X.\{t \in \omega \mid t+1 \in X\}$$

Keep in mind that in temporal logic, the set of possible worlds is the set of natural numbers ω with ordering $\leq$ so that we can refer to the elements of $\mathcal{U}$ explicitly.

4.1.1 Neighborhood Semantics

Scott [Sco70] gives the most general definition of semantics of unary intensional operators in terms of what he calls "neighborhoods". Here we will generalise his definition to include functions with arbitrary arities. Suppose ∇ is an n-ary intensional operator of *IL*. Then $\|\nabla\| = \Theta \in [P(\mathcal{U})^n \to P(\mathcal{U})]$. Associate with Θ an

indexed family of subsets of $P(\mathcal{U})^n$ by the following: $\Theta|_w = \{\vec{X} \in P(\mathcal{U})^n | w \in \Theta(\vec{X})\}$ for each $w \in \mathcal{U}$. In other words, $\Theta|_w$ consists of sets of neighborhoods of w with respect to Θ. Note that these neigborhoods of w do not necessarily contain w. Let $A_0, \cdots, A_{n-1}$ be formulas of IL, I be an intensional interpretation of IL, and $w \in \mathcal{U}$. Then the meaning of a formula of the form $\nabla(A_0, \cdots, A_{n-1})$ at w in I can be defined as follows.

$$\models_{I,w} \nabla(A_0, \ldots, A_{n-1}) \text{ iff } < \|A_0\|^I, \ldots, \|A_{n-1}\|^I > \in \Theta|_w$$

This approach is called *neighborhood semantics* and is, in fact, equivalent to the the semantics of intensional operators given above.

Given an element $\Theta \in [P(\mathcal{U})^n \to P(\mathcal{U})]$, we have described how to obtain the corresponding indexed family of neighborhoods; we can go in the opposite direction as well. Let $\{\mathcal{N}_w\}_{w \in \mathcal{U}}$ be an indexed family of neighborhoods where for all $w \in \mathcal{U}$, $\mathcal{N}_w \in P(P(\mathcal{U})^n)$. Then an element $\Theta \in [P(\mathcal{U})^n \to P(\mathcal{U})]$ that corresponds to the family can be obtained as follows:

$$\Theta = \lambda \vec{X}.\{w \in \mathcal{U} \mid \vec{X} \in \mathcal{N}_w\}$$

Therefore, both of the approaches lead to the same semantics.

Consider the temporal operator `first` again. Then the corresponding indexed family for $\|\texttt{first}\|$ can be defined as follows. For all $t \in \omega$,

$$\begin{aligned} (\|\texttt{first}\|)|_t &= \{ X \in P(\omega) \mid t \in \|\texttt{first}\|(X) \} \\ &= \{ X \in P(\omega) \mid 0 \in X \} \end{aligned}$$

Here it is easy to check that the condition $t \in \|\texttt{first}\|(X)$ is simply equivalent to the condition $0 \in X$. As for the temporal operator `next`, for all $t \in \omega$,

$$\begin{aligned} (\|\texttt{next}\|)|_t &= \{ X \in P(\omega) \mid t \in \|\texttt{next}\|(X) \} \\ &= \{ X \in P(\omega) \mid t+1 \in X \} \end{aligned}$$

In the above examples, we used accessibility relations and the satisfaction relation $\models$ to obtain the denotations of intensional operators in terms of algebraic and neighborhood semantics. This by no means suggests that Kripke-style semantics is equivalent to either of these two approaches. In fact, Kripke-style semantics based on accessibility relations may be cast as neighborhood semantics, but not conversely [vB84] [Che80]. Chellas [Che80] gives an example of a formula valid in any Kripke-style semantics, but not valid in general in neighborhood semantics. However, we will not go into details here. A thorough discussion about the relative strengths of different approaches to the semantics of intensional logic can be found in [Woj88].

In short, the semantics of intensional operators given in this chapter start from an abstract object, say an element Θ of $[P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$, and then extends the underlying language with a new symbol $\bigtriangledown$ by letting $\|\bigtriangledown\| = \Theta$. Therefore, in this approach, there is no need to refer to any accessibility relation associated with $\bigtriangledown$, even if there is one. Furthermore, it is far from obvious, if not impossible, how to formalize the semantics of intensional operators with arbitrary arities in terms of Kripke-style semantics.

An Abstract Characterisation of Intensional Logic

In either of the two approaches described above, an intensional logic can be characterised as follows: we first provide a set of possible worlds $\mathcal{U}$. Then we choose a collection of functions in $\cup_{n\in\omega}[P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$ including the complementation operation, intersection and union (or alternatively, a collection of families of neighborhoods.) In other words, we define an algebra, in notation $IA =< P(\mathcal{U}), \emptyset, \mathcal{U}, IOP >$ where IOP is a collection of functions over $P(\mathcal{U})$. Then we associate symbols (intensional operators) in the underlying language for the functions in IOP. The next step is to define a satisfaction relation $\models$ in terms of intensional interpretations in order to assign meanings to all elements of the language.

The underlying algebra together with the satisfaction relation $\models$ can be used to characterise intensional logics as follows: if IL is an intensional logic based on the algebra IA and the satisfaction relation $\models$, we will use the notation $IA^{\models}$ to refer to the logic. Sometimes we will also use the term IL to refer to the underlying intensional language where no confusion may arise. There are also more general algebraic approaches to the semantics of intensional logics [Woj88] [BS84]. However, possible worlds, in general, are not a feature of algebraic semantics.

4.2 Properties of Intensional Operators

The above formulation of the denotations of intensional operators will enable us to discuss general properties of intensional logics without referring to a particular one. The mathematical properties of the intensional logic under discussion will be reflected in the denotations of intensional operators of the logic. This section will discuss several important properties of intensional operators which will be used later to impose semantic constraints on ILP systems. Throughout, we assume that $IL = IA^{\models}$ is an arbitrary intensional logic based on the algebra $IA =< P(\mathcal{U}), \emptyset, \mathcal{U}, \ldots >$. However, we would like IL to have certain properties in order to assure that intensional logic programming languages based on IL share the properties of ordinary logic programming.

4.2.1 Monotonicity

Our first requirement of an intensional operator of IL is that its denotation be monotonic. This is hardly surprising, given that logic programming is based on monotonic logic. Monotonicity property has nice implications which cannot be dispensed with in intensional logic programming. It simply implies that if we know more information about the argument of a function, we shall know no less about

the result. Note that monotonicity has been studied in various subjects in computer science, including programming language semantics [Sto77].

We now define an ordering relation between the elements of $P(\mathcal{U})^n$ for any $n \in \omega$ by the following: Let $\vec{X}$ and $\vec{Y} \in P(\mathcal{U})^n$. We say that $\vec{X} \leq \vec{Y}$ iff $X_i \subseteq Y_i$ for all $i \in n$. In fact, $P(\mathcal{U})^n$ under $\leq$ is a complete lattice with $< \mathcal{U}, \mathcal{U}, \ldots, \mathcal{U} >$ as its top element and $< \emptyset, \emptyset, \ldots, \emptyset >$ as its bottom element.

Definition 4.2 *Let $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. We say that Θ is monotonic iff for all $\vec{X}$ and $\vec{Y} \in P(\mathcal{U})^n$, $\vec{X} \leq \vec{Y}$ implies $\Theta(\vec{X}) \subseteq \Theta(\vec{Y})$.*

The denotation of negation $\|\neg\|$ is not monotonic. Indeed, given an element X of $P(\mathcal{U})$, $\|\neg\|$ returns the complement of X with respect to $\mathcal{U}$, i.e., $\|\neg\| = \lambda X.(\mathcal{U} - X)$. It can be easily verified that the denotations of temporal operators `first` and `next` are both monotonic, and so are those of modal operators $\Box$ and $\Diamond$.

One immediate consequence of monotonicity in the neighborhood semantics is formulated by the following lemma. It states that, for a given n-ary monotonic function Θ, the neighborhoods of any $w \in \mathcal{U}$ with respect to Θ, is closed under $\leq$ relation (or supplemented in the terminology of Chellas [Che80]).

Lemma 4.1 *Let $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. Θ is monotonic iff for all $w \in \mathcal{U}$ and for all $\vec{X} \in P(\mathcal{U})^n$, $\vec{X} \in \Theta|_w$ implies for all $\vec{X} \leq \vec{Y} \in P(\mathcal{U})^n$, $\vec{Y} \in \Theta|_w$.*

Proof. Straightforward. ⊣

4.2.2 Universality

The monotonicity property rules out many functions in $\bigcup_{n \in \omega}[P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$ as potential intensional operators, but what is left still contains some undesirable functions. For instance, suppose `start` is a unary temporal operator of the temporal logic of Chronolog. The semantics of `start` is given as $\models_{I,t}$ `start` A iff $t = 0$ and

$\upsilon \in \|A\|^I$. It basically says that the time is 0 and A is true. We will form the function $\|\mathtt{start}\| \in [P(\omega) \to P(\omega)]$ as follows:

$$\begin{aligned} \|\mathtt{start}\|(\|A\|^I) &= \{\, t \in \omega \mid \models_{I,t} \mathtt{start}\, A \,\} \\ &= \{\, t \in \omega \mid t = 0 \text{ and } 0 \in \|A\|^I \} \\ &= \{\, 0 \mid 0 \in \|A\|^I \,\} \end{aligned}$$

Thus, by lambda abstraction, we obtain the definition of $\|\mathtt{start}\|$ as

$$\|\mathtt{start}\| = \lambda X.\{\, 0 \mid 0 \in X \,\}$$

which, given an element of $P(\mathcal{U})$, returns either $\emptyset$ or $\{0\}$. The problem with this operator is that any formula of the form $\mathtt{start}\, A$ does not have any intensional (temporal) models; therefore for any intensional (temporal) interpretation I, $\not\models_I \mathtt{start}\, A$.

We need a property which will state that if $\triangledown$ is an operator of IL whose denotation has the property, then any formula of the form $\triangledown A$ is guaranteed to have an intensional model. Monotonicity alone is not sufficient for that purpose, since we know that $\|\mathtt{start}\|$ is already monotonic, but still an undesirable function. Below is the formal definition of the property of *universality.*

Definition 4.3 *Let $\Theta \in [P(\mathcal{U})^n \to P(\mathcal{U})]$. We say that Θ is universal iff for some $\vec{X} \in P(\mathcal{U})^n$, $\Theta(\vec{X}) = \mathcal{U}$.*

Clearly $\|\mathtt{start}\|$ is not universal, because for any $X \in P(\omega)$, we have $\|\mathtt{start}\|(X) = \{0\}$ or $\emptyset$. Since $\|\mathtt{first}\|(\{0\}) = \omega$, $\|\mathtt{first}\|$ is an example of universal functions.

Universality condition also guarantees that the family of neighborhoods of Θ associated with any $w \in \mathcal{U}$ is not empty.

Lemma 4.2 *Let $\Theta \in [P(\mathcal{U})^n \to P(\mathcal{U})]$. If Θ is universal, then $\Theta|_w \neq \emptyset$ for all $w \in \mathcal{U}$.*

Proof. Trivial. ⊣

Consider negation again: $\|\neg\|$ is not monotonic, but universal, since $\|\neg\|(\emptyset) = \mathcal{U}$. On the other hand, $\|\neg\|(\mathcal{U}) = \emptyset$; therefore $\|\neg\|$ does not turn universal truth into universal truth. Let $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. When Θ is both universal and monotonic, we can obtain a stronger condition which says Θ turns universal truth into universal truth, for instance, the functions $\|\texttt{first}\|$ and $\|\texttt{next}\|$.

Lemma 4.3 *Let* $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$ *and* $\vec{U} =< \mathcal{U}, \mathcal{U}, \ldots, \mathcal{U} >$. *If* Θ *is universal and monotonic, then* $\Theta(\vec{U}) = \mathcal{U}$, *i.e.,* $\vec{U} \in \Theta|_w$ *for all* $w \in \mathcal{U}$.

Proof. Since Θ is universal, for some $\vec{X} \in P(\mathcal{U})^n$, $\Theta(\vec{X}) = \mathcal{U}$. Thus $\Theta(\vec{U}) = \mathcal{U}$ by the monotonicity of Θ. ⊣

4.2.3 Conjunctivity

We have given an axiom related to the temporal operator **first** that states that **first** can be distributed over conjunction and initial truths can be conjoined.

A3. $\texttt{first}(A \wedge B) \leftrightarrow (\texttt{first}\, A \wedge \texttt{first}\, B)$

We need to encapsulate this property, but at a more general and semantic level. Van Benthem [vB84] has introduced a similar notion of conjunctivity related to the necessity operator $\Box$ to study the conditions under which a neighborhood semantics may be replaced by a standard Kripke-style semantics; but this is beyond the scope of this dissertation. Therefore the focus will be the consequences of conjunctivity within intensional logic programming.

Let $\vec{X}$ and $\vec{Y} \in P(\mathcal{U})^n$. We define $\vec{X} \cap \vec{Y} =< X_0 \cap Y_0, \ldots, X_{n-1} \cap Y_{n-1} >$ and $\vec{X} \cup \vec{Y} =< X_0 \cup Y_0, \ldots, X_{n-1} \cup Y_{n-1} >$. These operations naturally extend to arbitrary elements of $P(P(\mathcal{U})^n))$. Below is the formal definition of the property of *conjunctivity.*

Definition 4.4 *Let* $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. *We say that* Θ *is conjunctive iff for all* $\{\vec{X}_\alpha \in P(\mathcal{U})^n\}_{\alpha \in S}$, $\Theta(\bigcap_{\alpha \in S} \vec{X}_\alpha) = \bigcap_{\alpha \in S} \Theta(\vec{X}_\alpha)$.

Conjunctivity relates to, or can be broken into, two properties: monotonicity, and another property with a striking consequence in the neighborhood semantics. Recall that for any $t \in \omega$, $(\|\texttt{first}\|)|_t = \{X \in P(\omega) \mid 0 \in X\}$, or equivalently $(\|\texttt{first}\|)|_t = \{X \in P(\omega) \mid \{0\} \subseteq X\}$. Then it can be shown that $\{0\}$ is the least element in $(\|\texttt{first}\|)|_t$, and, in fact, is the intersection of all elements of $(\|\texttt{first}\|)|_t$. This intersective property is what we are after:

Lemma 4.4 *Let* $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. Θ *is conjunctive iff* Θ *is monotonic and for all* $w \in \mathcal{U}$, $\Theta|_w \neq \emptyset$ *implies* $\Theta|_w$ *is closed under intersection.*

Proof. Suppose Θ is conjunctive. For any $\vec{X}$ and $\vec{Y} \in P(\mathcal{U})^n$, if $\vec{X} \leq \vec{Y}$ (or $\vec{X} \cap \vec{Y} = \vec{X}$), then $\Theta(\vec{X}) = \Theta(\vec{X}) \bigcap \Theta(\vec{Y})$ by the conjunctivity of Θ, which means $\Theta(\vec{X}) \subseteq \Theta(\vec{Y})$. Therefore Θ is monotonic. Now pick any $w \in \mathcal{U}$ with $\Theta|_w$ nonempty. To show $\Theta|_w$ is closed under intersection, first let $\{\vec{X}_\alpha\}_{\alpha \in S}$ be any subset of $\Theta|_w$. Then for all $\alpha \in S$, $w \in \Theta(\vec{X}_\alpha)$, which implies $w \in \bigcap_{\alpha \in S} \Theta(\vec{X}_\alpha)$ and hence $w \in \Theta(\bigcap_{\alpha \in S} \vec{X}_\alpha)$ by the conjunctivity of Θ. Therefore $\bigcap_{\alpha \in S} \vec{X}_\alpha \in \Theta|_w$.

Conversely, suppose Θ is monotonic and for all $w \in \mathcal{U}$, $\Theta|_w \neq \emptyset$ implies $\Theta|_w$ is closed under intersection. Consider any subset of $\Theta|_w$ $\{\vec{X}_\alpha \in \Theta|_w\}_{\alpha \in S}$; then for any $\alpha \in S$, $\Theta(\bigcap_{\alpha \in S} \vec{X}_\alpha) \subseteq \Theta(\vec{X}_\alpha)$ by the monotonicity of Θ. Thus $\Theta(\bigcap_{\alpha \in S} \vec{X}_\alpha) \subseteq \bigcap_{\alpha \in S} \Theta(\vec{X}_\alpha)$ which is the first half of the conjunctivity of Θ. Now for all $w \in \bigcap_{\alpha \in S} \Theta(\vec{X}_\alpha)$, for all $\alpha \in S$, $\vec{X}_\alpha$ is in $\Theta|_w$. Therefore $\bigcap_{\alpha \in S} \vec{X}_\alpha$ is in $\Theta|_w$ as well, because $\Theta|_w$ is closed under intersection, which in turn implies that $w \in \Theta(\bigcap_{\alpha \in S} \vec{X}_\alpha)$, i.e., $\Theta(\bigcap_{\alpha \in S} \vec{X}_\alpha) \supseteq \bigcap_{\alpha \in S} \Theta(\vec{X}_\alpha)$. Hence we have that Θ is conjunctive. $\dashv$

We will show that the denotation of the *S5* possibility operator $\|\Diamond\|$ is monotonic, but not conjunctive. Consider the indexed family of subsets of $P(\mathcal{U})$ associated with

$\|\Diamond\|$, i.e., for all $w \in \mathcal{U}$,

$$\begin{aligned}(\|\Diamond\|)|_w &= \{X \in P(\mathcal{U}) \mid w \in \|\Diamond\|(X)\} \\ &= \{X \in P(\mathcal{U}) \mid X \neq \emptyset\}\end{aligned}$$

Then for any $w \in \mathcal{U}$, $(\|\Diamond\|)|_w$ is not closed under intersection, since, for instance, the intersection of two different singletons in $P(\mathcal{U})$ is the emptyset and the emptyset is not an element of $(\|\Diamond\|)|_w$. The denotation of the *S5* necessity operator $\|\Box\|$ is conjunctive and, in fact, for any $w \in \mathcal{U}$, $(\|\Box\|)|_w = \{\mathcal{U}\}$.

4.2.4 Finitariness

Let $\bigtriangledown$ be a unary intensional operator of the language. Then the value of a formula of the form $\bigtriangledown A$ at a given world w may depend on the values of A at a set of worlds, possibly including w. If this set happens to be infinite, any machinery to prove $\bigtriangledown A$ at w may fail to terminate because of the need to prove A at all worlds in the set. Therefore, for computability reasons, another property is necessary to filter out such kind of intensional operators. Yaghi [Yag84] invesigated a similar property for the operators of dataflow language Lucid [AW76]; Fitting [Fit87] applied the same notion to modular logic programming.

We say that $\vec{X} \in P(\mathcal{U})^n$ is finite iff for all $i \in n$, X_i is finite. Below is the formal definition of *finitariness* (called compactness in Fitting's work.)

Definition 4.5 *Let $\Theta \in [P(\mathcal{U})^n \to P(\mathcal{U})]$. We say that Θ is finitary iff for all $\vec{X} \in P(\mathcal{U})^n$ and for all $w \in \mathcal{U}$, $w \in \Theta(\vec{X})$ implies $w \in \Theta(\vec{S})$ for some finite $\vec{S} \leq \vec{X}$.*

Intuitively, this definition means that we can verify that $w \in \Theta(X)$ by referring only to a finite subset of X; therefore a finite amount of information is sufficient. For instance, consider $\|\texttt{first}\|$ and $\|\texttt{next}\|$. Let $t \in \omega$ and $X \in P(\omega)$. If $t \in \|\texttt{first}\|(X)$, then $\{0\} \subseteq X$ and $t \in \|\texttt{first}\|(\{0\})$. It follows that $\|\texttt{first}\|$ is

finitary. Similarly, if $t \in \|\mathtt{next}\|(X)$, then $\{t+1\} \subseteq X$ and $t \in \|\mathtt{next}\|(\{t+1\})$, and therefore $\|\mathtt{next}\|$ is finitary as well.

As for $\|\Box\|$, if $w \in \|\Box\|(X)$ where $w \in \mathcal{U}$ and $X \in P(\mathcal{U})$, then X must be equal to $\mathcal{U}$, thus $\|\Box\|$ is not finitary. But $\|\Diamond\|$ is finitary, since $w \in \|\Diamond\|(X)$ implies that $X \neq \emptyset$ and therefore for any $z \in X$, $w \in \|\Diamond\|(\{z\})$. Finitariness has some implications for the neighborhood semantics

Lemma 4.5 *Let $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. Suppose Θ is finitary. Then for all $w \in \mathcal{U}$ and for all $\vec{X} \in P(\mathcal{U})$, $\vec{X} \in \Theta|_w$ implies there exists a finite $\vec{S} \leq \vec{X}$ such that $\vec{S} \in \Theta|_w$.*

Proof. Straightforward. ⊣

The lemma also implies that the minimal elements of any family of neighborhoods of a finitary Θ are finite.

When Θ is conjunctive, finitariness can also be expressed as follows: Θ is finitary iff $\Theta|_w \neq \emptyset$ implies $\cap\Theta|_w$ is finite for all $w \in \mathcal{U}$. Note that since $\Theta|_w$ is closed under intersection, $\cap\Theta|_w$ is the minimum element in $\Theta|_w$ for which $w \in \Theta(\cap\Theta|_w)$. In other words, $\cap\Theta|_w$ is the least neighborhood of w with respect to Θ, therefore we can verify that $w \in \Theta(\vec{X})$ if $\cap\Theta|_w \leq \vec{X}$.

4.2.5 Continuity

Finitariness combined with monotonicity leads to another desirable and strong property of functions which has been discussed in programming language semantics under the name of *continuity* [Sto77]. In intensional semantics, we will adopt the following definition of continuity.

Definition 4.6 *Let $\Theta \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. We say that Θ is continuous iff for all chains $< \vec{X}_n >_{n\in\omega}$ over $P(\mathcal{U})^n$, $\cup_{n\in\omega}\Theta(\vec{X}_n) = \Theta(\cup_{n\in\omega}\vec{X}_n)$.*

Given monotonicity, finitarines is associated with continuity. We state the following theorem whose proof is adapted from that of an analogous theorem given by Yaghi [Yag84] for temporal operators of dataflow language Lucid [WA85].

Theorem 4.6 *Let* $\Theta \in [P(\mathcal{U})^n \to P(\mathcal{U})]$. *Then* Θ *is continuous iff it is monotonic and finitary.*

Proof. We first show that if Θ is continuous, then it is monotonic and finitary. Consider any $\vec{X}$ and $\vec{Y} \in P(\mathcal{U})^n$ such that $\vec{X} \leq \vec{Y}$, i.e., $< \vec{X}, \vec{Y} >$ is a chain. Then $\Theta(\vec{X}) \cup \Theta(\vec{Y}) = \Theta(\vec{X} \cup \vec{Y})$ by continuity of Θ. Since we have that $\vec{X} \cup \vec{Y} = \vec{Y}$, $\Theta(\vec{X}) \cup \Theta(\vec{Y}) = \Theta(\vec{Y})$. Therefore $\Theta(\vec{X}) \subseteq \Theta(\vec{Y})$, that is, Θ is monotonic. Now suppose Θ is non-finitary. Let $\vec{X} =< \{w_0^0, w_1^0, w_2^0, \ldots\}, \ldots, \{w_0^{n-1}, w_1^{n-1}, w_2^{n-1}, \ldots\} >$ be an elelement of $\Theta|_w$ for any $w \in \mathcal{U}$ with no finite $\vec{S} \leq \vec{X}$ in $\Theta|_w$, and construct an ω-chain $< \vec{C}_i >_{i \in \omega}$ of finite elements of $P(\mathcal{U})^n$ such that $\cup_{i \in \omega} \vec{C}_i = \vec{X}$: $< \vec{C}_i >_{i \in \omega} =<< \emptyset, \ldots, \emptyset >, < \{w_0^0\}, \ldots, \emptyset >, < \{w_0^0\}, \{w_0^1\}, \ldots, \emptyset >, \ldots, < \{w_0^0\}, \{w_0^1\}, \ldots, \{w_0^{n-1}\} >, < \{w_0^0, w_1^0\}, \{w_0^1\}, \ldots, \{w_0^{n-1}\} >, \ldots >$. Since all $\vec{C}_i$'s in the chain are finite, we have that $w \notin \Theta(\vec{C}_i)$ for any $i \in \omega$. Then $w \notin \cup_{i \in \omega} \Theta(\vec{C}_i)$ but $w \in \Theta(\cup_{i \in \omega} \vec{C}_i)$, which contradicts that Θ is continuous. Thus Θ is finitary.

Conversely, suppose that Θ is monotonic and finitary. Let $< \vec{X}_\alpha >_{\alpha \in I}$ be an upwards chain of elements over $P(\mathcal{U})^n$. Let $lub(< \vec{X}_\alpha >_{\alpha \in I}) = \cup_{\alpha \in A} \vec{X}_\alpha$. Then for all $\alpha \in I$, $\vec{X}_\alpha \subseteq \cup_{\alpha \in I} \vec{X}_\alpha$, which implies that for all $\alpha \in I$, $\Theta(\vec{X}_\alpha) \subseteq \Theta(\cup_{\alpha \in I} \vec{X}_\alpha)$ by monotonicity of Θ. Thus $\cup_{\alpha \in I} \Theta(\vec{X}_\alpha) \subseteq \Theta(\cup_{\alpha \in I} \vec{X}_\alpha)$. Now suppose that $\cup_{\alpha \in I} \Theta(\vec{X}_\alpha) \subset \Theta(\cup_{\alpha \in I} \vec{X}_\alpha)$, that is, Θ is discontinuous. Then for some $w \in \mathcal{U}$, $w \notin \cup_{\alpha \in I} \Theta(\vec{X}_\alpha)$, but $w \in \Theta(\cup_{\alpha \in I} \vec{X}_\alpha)$. Therefore for all $\alpha \in I$, $w \notin \Theta(\vec{X}_\alpha)$. Since Θ is finitary and $w \in \Theta(\cup_{\alpha \in I} \vec{X}_\alpha)$, $w \in \Theta(\vec{S})$ for some finite $\vec{S} \leq (\cup_{\alpha \in I} \vec{X}_\alpha)$. Now it must be case that $\vec{S} \leq \vec{X}_\alpha$ for some $\alpha \in I$, because $\vec{S}$ is finite. Then by monotonicity of Θ, $\Theta(\vec{S}) \subseteq \Theta(\vec{X}_\alpha)$, which in turn implies that $w \in \Theta(\vec{X}_\alpha)$ as well, a clear contradiction. Hence $\cup_{\alpha \in I} \Theta(\vec{X}_\alpha) = \Theta(\cup_{\alpha \in I} \vec{X}_\alpha)$, that is, Θ is continuous. $\dashv$

Consider the *S5* necessity operator $\Box$. Recall that $\|\Box\| = \lambda X.\{w \in \mathcal{U} \mid X = \mathcal{U}\}$. Since $\|\Box\|$ is not finitary, it fails to be continuous. Indeed, $\|\Box\|$ does not satisfy the definition of continuity. Let $< C_n >_{n\in\omega} = < \emptyset, \{w_0\}, \{w_0, w_1\}, \{w_0, w_1, w_2\}, \ldots >$ be an ω-chain of elements of $P(\mathcal{U})$ where $\cup_{n\in\omega} C_n = \mathcal{U}$. For any $n \in \omega$, $\|\Box\|(C_n) = \emptyset$. Then $\|\Box\|(\cup_{n\in\omega} C_n) = \mathcal{U}$, but $\cup_{n\in\omega}\|\Box\|(C_n) = \emptyset$.

4.3 Monotonic Formulas

It is a common practice in logic that new operators are introduced in a language by considering formulas of the language as the definitions of operators. We can pick a formula of the language in n-distinct propositional variables, and treat it as the defining formula of an n-ary operator. Here propositional variables are just place-holders for formulas. For instance, disjunction ($\vee$) can be defined in terms of conjunction and negation, i.e., $A \vee B =_{def} \neg(\neg A \wedge \neg B)$. Similarly, the possibility operator $\Diamond$ can be defined as $\Diamond A =_{def} \neg\Box\neg A$. As an alternative, we could directly extend the underlying language with extra (intensional) operators. In the first case, the denotation of a defined intensional operator can be obtained from the denotations of those operators used in the definition. In the second case, we explicitly specify the denotations of extra intensional operators as elements of $\cup_{n\in\omega}[P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$. In this section, we will focus on the first approach.

Consider the temporal logic of Chronolog and a formula of the form $\texttt{first}\ A_0 \vee (A_1 \wedge \texttt{next next}\ A_1)$ where A_0 and A_1 are distinct propositional variables, i.e., place-holders for formulas. Then a new binary intensional operator, say $\bigtriangledown$, can be defined as follows.

$$\bigtriangledown(A_0, A_1) =_{def} \texttt{first}\ A_0 \vee (A_1 \wedge \texttt{next next}\ A_1)$$

Then any formula of the form $\bigtriangledown(B_0, B_1)$ would be regarded as a formula obtained from the definition of $\bigtriangledown$ by substituting B_0 for A_0 and B_1 for A_1. Here the deno-

tation of $\bigtriangledown$ can be obtained from the definition as

$$\|\bigtriangledown\|(X,Y) = \|\texttt{first}\|(X) \cup (Y \cap \|\texttt{next}\| \circ \|\texttt{next}\|(Y))$$

with the obvious interpretations of $\cup$ and $\cap$. The symbol $\circ$ represents the function composition operation. We will consider, as defining formulas for new operators, *monotonic formulas* in one or more propositional variable. A monotonic formula contains only those intensional operators whose denotations are monotonic, and atomic formulas. Monotonic formulas with n propositional variables correspond to n-ary monotonic functions.

The properties of functions involved in the definitions of new functions such as $\|\bigtriangledown\|$ are preserved under certain restrictions. For instance, we would like to know whether or not $\|\bigtriangledown\|$ is monotonic. The following lemma answers the question as to which properties are closed under function composition.

Lemma 4.7 *Monotonicity, conjunctivity and finitariness are closed under composition.*

Proof. We will prove only the second part of the lemma. Let Θ be a composed function over $P(\mathcal{U})$. We will show by induction on the structure of Θ that Θ is conjunctive

Basis case: Θ is conjunctive when it is of the form $\Theta(\vec{X}) =_{def} \Psi(\vec{Y})$ where Ψ is a conjunctive m-ary function and each Y_j is equal to some X_i.

Inductive case: Θ is of the form $\Theta(\vec{Z}) =_{def} \Psi(X_0,\ldots,\Phi(\vec{Y}),\ldots,X_{k-1})$ where Ψ and Φ are conjunctive functions of arities k and m respectively. Furthermore, each X_i or Y_j is equal to some Z_l. Pick any $\vec{Z}$ and $\vec{Z}' \in P(\mathcal{U})^n$,

$$\begin{aligned}
\Theta(\vec{Z} \cap \vec{Z}') &= \Psi(X_0 \cap X_0',\ldots,\Phi(\vec{Y} \cap \vec{Y}'),\ldots,X_{k-1} \cap X_{k-1}') \\
&= \Psi(X_0 \cap X_0',\ldots,\Phi(\vec{Y}) \cap \Phi(\vec{Y}'),\ldots,X_{k-1} \cap X_{k-1}') \\
&= \Psi(X_0,\ldots,\Phi(\vec{Y}),\ldots,X_{k-1}) \cap \Psi(X_0',\ldots,\Phi(\vec{Y}'),\ldots,X_{k-1}') \\
&= \Theta(\vec{Z}) \cap \Theta(\vec{Z}')
\end{aligned}$$

that is, Θ is conjunctive. $\dashv$

However, universality is not closed under composition. Consider the function $\|\neg\| \circ \Theta$ where $\Theta = \lambda X.\mathcal{U}$. We have that both $\|\neg\|$ and Θ are universal, because $\|\neg\|(\emptyset) = \mathcal{U}$ and $\Theta(X) = \mathcal{U}$ for any subset X of $\mathcal{U}$. But $\|\neg\| \circ \Theta$ is clearly not universal. In fact, for any subset X of $\mathcal{U}$, $\|\neg\| \circ \Theta(X) = \emptyset$.

The following lemma states that universality is preserved for functions composed out of universal and monotonic functions.

Lemma 4.8 *Given monotonicity, universality is closed under composition.*

Proof. Let Θ be a composed function over $P(\mathcal{U})$ and $\vec{U} = < \mathcal{U}, \mathcal{U}, \ldots, \mathcal{U} >$. Then it can be shown that, by lemma 4.3, $\Theta(\vec{U}) = \mathcal{U}$. Therefore Θ is universal. $\dashv$

In summary, at the semantic level, monotonic formulas with n distinct propositional variables correspond to n-ary monotonic functions in the underlying algebra. We can p[ut] a monotonic formula of *IL* and treat it as the definition of a new intensional operator $\bigtriangledown$, and still remain within the underlying logic. In the example given above, the semantics of formulas of the form $\bigtriangledown(\|B_0\|, \|B_1\|)$ can be defined as follows. We have that $\models_{I,w} \bigtriangledown(B_0, B_1)$

$$\begin{aligned} &\text{iff} \quad w \in \|\bigtriangledown\|(\|B_0\|, \|B_1\|) \\ &\text{iff} \quad w \in \|\texttt{first}\|(\|B_0\|^I) \cup (\|B_1\|^I \cap \|\texttt{next}\| \circ \|\texttt{next}\|(\|B_1\|^I)) \end{aligned}$$

where I be an intensional interpretation and $w \in \mathcal{U}$. This is to manifest that any formula of the form $\bigtriangledown(B_0, B_1)$ refers to the defining formula of $\bigtriangledown$ with A_0 and A_1 substituted by B_0 and B_1 respectively. Note that in this approach, we do not gain any increase in the expressive power of the underlying logic. Segerberg [Seg82] presents a thorough discussion on this topic.

Chapter 5

Models of Intensional Logic Programs

In this chapter, we will develop a generalised model theory for intensional logic programs in terms of minimum intensional Herbrand models, along the lines of [vEK76] [Llo84]. We shall not focus on any particular language such as those defined earlier. Instead, several essential constraints will be imposed on intensional operators available in the language so that the results will remain valid regardless of what the language offers. From here on, unless otherwise stated, we *assume* that all (intensional) operators that can be used in intensional logic programs have the properties formulated in the previous chapter. In other words, if $\|\nabla\|$ is an (intensional) operator of the logic, its denotation is assumed to be universal, monotonic, conjunctive and finitary.

In the following, we will first extend the formal definition of intensional logic programs, and then develop the model-theoretic semantics of intensional logic programs in terms of intensional Herbrand interpretations. The fixpoint characterisation of the minimum intensional Herbrand model of an intensional logic program will be provided as well. We will justify why the properties monotonicity, universality, fini-

tariness and conjunctivity are needed, and demonstrate how they can affect the underlying theory. We will also show that the theory can be applied to a number of intensional languages, including the ones described in the dissertation.

5.1 Intensional Logic Programs Revisited

The intensional logic under discussion will be denoted by $IL = IA^{\models}$ based on the algebra IA as defined in the previous chapter. We have defined intensional logic programs as a set of intensional program clauses. The basic building blocks in an intensional logic program are now monotonic formulas, which we will call *intensional units* in this context. In the previous chapter, we have shown that monotonic formulas with n distinct propositional variables can be considered as the definitions of new intensional operators. Then intensional units that appear in the programs may be regarded as the applications of such defined operators to some other intensional units, i.e., if A is an intensional unit with n distinct atomic formulas, then A is of the form $\bigtriangledown(A_0, \ldots, A_{n-1})$ and we have that $\|\bigtriangledown\| \in [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$.

The following discussion will be based on this approach, because, under the light of lemmas 4.7 and 4.8 (stating the properties of intensional operators are closed under composition), it makes no difference from a model-theoretical point of view, and furthermore we will have a much smoother theory. Recall that all variables in intensional Horn clauses are assumed to be universally quantified. For instance, the following is an intensional (temporal) program clause:

$$\texttt{first p(X,Y)} \leftarrow \texttt{next (q(X)} \vee \texttt{first q(Y))}$$

or equivalently,

$$\texttt{first p(X,Y)} \leftarrow \bigtriangledown\texttt{(q(X),q(Y))}$$

where $\bigtriangledown(A, B) =_{def} \texttt{next}(A \vee \texttt{first}\, B)$ for distinct propositional variables A and B. We have that $\|\bigtriangledown\|$ is monotonic, universal and finitary by lemmas 4.7 and 4.8.

We are interested in those interpretations of an intensional logic program which makes the program true at all worlds in $\mathcal{U}$. We understand an intensional logic program $\mathcal{P}$ in terms of intensional interpretations as follows: $\mathcal{P}$ is true in an intensional interpretation I iff all clauses in $\mathcal{P}$ are true in I. A clause is true in I iff it is true at all worlds in $\mathcal{U}$. We can now elaborate what a model of $\mathcal{P}$ means.

Definition 5.1 *Let I be an intensional interpretation of IL. Then I is a model of $\mathcal{P}$ iff I is a model for each clause $C \in \mathcal{P}$, that is, $\models_I \mathcal{P}$ iff for all $C \in \mathcal{P}$, $\models_I C$.*

5.2 Intensional Herbrand Interpretations

We will now study intensional Herbrand interpretations. We call ground atomic formulas *intensional ground atoms.* The meaning of an intensional ground atom in an intensional interpretation I is an intension, that is, a function in $[\mathcal{U} \rightarrow 2]$ or equivalently an element of $P(\mathcal{U})$, whereas the meaning of a ground atom in classical Horn logic is simply 0 or 1 in an interpretation. Here the definitions of intensional Herbrand universe and base of an intensional logic program $\mathcal{P}$ coincide with those of logic programming given earlier (definitions 2.13 and 2.14.) Now the intensional Herbrand base $B_{\mathcal{P}}$ is the set of all intensional ground atoms which can be constructed out of predicate symbols that appear in $\mathcal{P}$ with ground terms in $U_{\mathcal{P}}$ as arguments.

Intensional Herbrand interpretations of $\mathcal{P}$ have $U_{\mathcal{P}}$ as their domain. In general, a set-theoretic representation of an intensional Herbrand interpretation of $\mathcal{P}$ is not possible, because we cannot depend on the existence of canonical intensional ground atoms as we did for temporal logic programming, and furthermore there is no guarantee that they even exist for an arbitrary language. However, as the domain of an intensional Herbrand interpretation I is the Herbrand universe $U_{\mathcal{P}}$ and for all $e \in U_{\mathcal{P}}$, $I(e) = e$, I can be identified with a function H which assigns to each

intensional ground atom $p(e_0, \ldots, e_{n-1}) \in B_{\mathcal{P}}$ an element of $P(\mathcal{U})$ by the following.

$$< e_0, \ldots, e_{n-1} > \in I(p)(w) \text{ iff } w \in \|p(e_0, \ldots, e_{n-1})\|^H$$

In fact, H is just an element of $[B_{\mathcal{P}} \rightarrow P(\mathcal{U})]$.

We say that H is a model of $\mathcal{P}$, in notation $\models_H \mathcal{P}$, iff $\models_I \mathcal{P}$ for any I corresponding to H. From here on, we will use these two dual notions of an intensional Herbrand interpretation interchangeably. Given the semantics of the formulas of IL, the notion of intension naturally extends to all formulas of IL. We also assume that the semantics of intensional operators are naturally embedded in H, i.e., the function that corresponds to I. We also say that a clause is true in an intensional Herbrand interpretation I iff all of its ground instances are true in I at all possible worlds in $\mathcal{U}$. This is implied by the semantics of the universal quantifier $\forall$ and the fact that $I(e) = e$ for all $e \in B_{\mathcal{P}}$. Suppose $C = A \leftarrow B_0, B_1, \ldots, B_{n-1}$ is such a ground instance. Then it is clear that C is true in I, i.e., I is a model of C, iff $\|A\|^I \supseteq \cap_{i \in n} \|B_i\|^I$.

Since intensional Herbrand interpretations can no longer be treated as subsets of $B_{\mathcal{P}}$, we shall define an ordering relation and two operations on intensional Herbrand interpretations which are analogous to set inclusion, intersection and union respectively. Let $\mathcal{F}(\mathcal{P})$ denote the set of intensional Herbrand interpretations of a given intensional logic program $\mathcal{P}$.

Definition 5.2 *Let $\mathcal{P}$ be an intensional logic program. We define $I \sqsubseteq J$ iff $\|A\|^I \subseteq \|A\|^J$ for all $A \in B_{\mathcal{P}}$ for any I and $J \in \mathcal{F}(\mathcal{P})$.*

For a given intensional logic program $\mathcal{P}$, the least upper bound (lub) of any family M of intensional Herbrand interpretations is $\sqcup M$, and the greatest lower bound (glb) $\sqcap M$, both of which are defined by the following:

Definition 5.3 *Let $\mathcal{P}$ be an intensional logic program and $M = \{I_\alpha\}_{\alpha \in S}$ a family of intensional Herbrand interpretations of $\mathcal{P}$.*

(a) $\sqcap$-*intersection* : $\sqcap M =_{def} \sqcap_{\alpha \in S} I_\alpha$ *is an intensional Herbrand interpretation of* $\mathcal{P}$ *where* $\|A\|^{\sqcap M} = \cap_{\alpha \in S} \|A\|^{I_\alpha}$ *for all* $A \in B_{\mathcal{P}}$.

(b) $\sqcup$-*union* : $\sqcup M =_{def} \sqcup_{\alpha \in S} I_\alpha$ *is an intensional Herbrand interpretation of* $\mathcal{P}$ *where* $\|A\|^{\sqcup M} = \cup_{\alpha \in S} \|A\|^{I_\alpha}$ *for all* $A \in B_{\mathcal{P}}$.

In summary, the ordering relation $\sqsubseteq$ defines a partial order on the intensional Herbrand interpretations of a given intensional (Horn) logic program $\mathcal{P}$, as does the set inclusion on Herbrand interpretations of a given Horn logic program. In fact, the family of all intensional Herbrand interpretations of $\mathcal{P}$ is a complete lattice under the partial order of $\sqsubseteq$. The topmost element of the lattice is the intensional Herbrand interpretation $H_{\mathcal{P}} = \sqcup \mathcal{F}(\mathcal{P})$ defined by the following: for all $A \in B_{\mathcal{P}}$, $\|A\|^{H_{\mathcal{P}}} = \mathcal{U}$. The bottommost element $H_\emptyset = \sqcap \mathcal{F}(\mathcal{P})$ of the lattice can be defined in a similar way : for all $A \in B_{\mathcal{P}}$, $\|A\|^{H_\emptyset} = \emptyset$. In fact, $\mathcal{F}(\mathcal{P})$ is a complete Boolean algebra induced by the complete Boolean algebra of $P(\mathcal{U})$.

The following two lemmas justify the claim that intensional Herbrand interpretations are sufficient for proving the unsatisfiability of a set of intensional Horn clauses.

Lemma 5.1 *Let* $\mathcal{S}$ *be a set of intensional Horn clauses and suppose* $\mathcal{S}$ *has an intensional model. Then* $\mathcal{S}$ *has an intensional Herbrand model.*

Proof. Let I be an intensional interpretation of $\mathcal{S}$. Then the corresponding intensional Herbrand interpretation I_H can be defined as follows : The domain $\mathbf{D}$ of I_H is $U_{\mathcal{S}}$, and for all $p(e_0, \ldots, e_{n-1}) \in B_{\mathcal{S}}$ and for all $w \in \mathcal{U}$, $w \in \|p(e_0, \ldots, e_{n-1})\|^{I_H}$ whenever $< I(e_0), ..., I(e_{n-1}) > \in I(p)(w)$. Clearly if I is a model of $\mathcal{S}$, so is I_H. $\dashv$

Lemma 5.2 *Let* $\mathcal{P}$ *be an intensional logic program and* A *be an intensional unit where all the atomic formulas that appear in* A *are in* $B_{\mathcal{P}}$. *Then* $\mathcal{P} \cup \{\neg A\}$ *is unsatisfiable in any intensional model of* $\mathcal{P}$ *iff no intensional Herbrand model of* $\mathcal{P}$ *satisfies* $\mathcal{P} \cup \{\neg A\}$.

Proof. If $\mathcal{P} \cup \{\neg A\}$ is satisfiable in some intensional model I of $\mathcal{P}$, then for some $w \in \mathcal{U}$, $\models_{I,w} \mathcal{P} \cup \{\neg A\}$. By lemma 5.1, a construction from I to the corresponding intensional Herbrand model H can be given. $\dashv$

5.3 Model-Theoretical Semantics

In this section, a minimum model semantics for intensional logic programs will be developed. We will first show that every intensional logic program has a model which is the topmost element in the lattice of the family of intensional Herbrand interpretations, therefore the set of models of any program is non-empty. In other words, intensional logic programs are consistent. Then we will show that the model $\sqcap$-intersection of the family of intensional Herbrand models of a given program $\mathcal{P}$ characterises the declarative semantics of $\mathcal{P}$.

We now prove the existence of models of intensional logic programs. The following lemma can not be proved if we drop the restriction that the denotations of operators be universal and monotonic. Recall that, for any X and $Y \subseteq \mathcal{U}$, $X \subseteq Y]$ implies $\|\neg\|(X) \supseteq \|\neg\|(Y)$, where $\|\neg\| = \lambda X.(\mathcal{U} - X)$. But $\|\neg\|$ is universal, since $\|\neg\|(\emptyset) = \mathcal{U}$. If any such operator is used in the head of any clause of an intensional logic program $\mathcal{P}$, the program, in general, may not even be consistent.

Lemma 5.3 *Let $H_{\mathcal{P}}$ be an intensional Herbrand interpretation of an intensional logic program $\mathcal{P}$ where for all $A \in B_{\mathcal{P}}$, $\|A\|^{H_{\mathcal{P}}} = \mathcal{U}$. Then $H_{\mathcal{P}}$ is a model of $\mathcal{P}$, i.e., $\models_{H_{\mathcal{P}}} \mathcal{P}$.*

Proof. Consider any ground instance $A \leftarrow B_0, B_1, \ldots, B_{m-1}$ of any clause in $\mathcal{P}$. Here A is an intensional unit of the form $\nabla(A_0, \ldots, A_{n-1})$ where $\|\nabla\| \in [P(\mathcal{U})^n \to P(\mathcal{U})]$. We have that $\|\nabla\|$ is universal and monotonic by lemmas 4.7 and 4.8. We also know that for any $F \in B_{\mathcal{P}}$, $\|F\|^{H_{\mathcal{P}}} = \mathcal{U}$ by construction, therefore $\|\nabla(A_0, \ldots, A_{n-1})\|^{H_{\mathcal{P}}} = \|\nabla\|(\|A_0\|^{H_{\mathcal{P}}}, \ldots, \|A_{n-1}\|^{H_{\mathcal{P}}}) = \|\nabla\|(\vec{U})$. Then by

lemma 4.3, $\|\bigtriangledown\|(\vec{U}) = \mathcal{U}$, therefore $H_{\mathcal{P}}$ must be a model of the ground instance. Thus $\models_{H_{\mathcal{P}}} A \leftarrow B_0, B_1, \ldots, B_{m-1}$ for any ground instance of any clause in $\mathcal{P}$, which implies that $\models_{H_{\mathcal{P}}} \mathcal{P}$. $\dashv$

The lemma given above is not valid for the following single line program as we know that $\|\texttt{start}\|$ is not universal: $\mathcal{P} = \{\ \texttt{start p} \leftarrow\ \}$. In fact, $\mathcal{P}$ has no intensional Herbrand models. The restrictions imposed upon intensional operators will prevent these kind of anomalies.

The Model $\sqcap$-intersection Property

We will show that conjunctivity is related to the model intersection property. The following lemma establishes that the $\sqcap$-intersection of a family of intensional Herbrand models of any intensional unit A interpreted as the defining formula of an intensional operator $\bigtriangledown$ is also a model, providing $\|\bigtriangledown\| = \Theta$ is conjunctive. Recall from the neighborhood semantics that $\Theta|_w = \{\vec{X} \in P(\mathcal{U})^n \mid w \in \Theta(\vec{X})\ \}$. Let $\mathcal{P}$ be an intensional logic program.

Lemma 5.4 *Let $A_i \in B_{\mathcal{P}}$ for all $i \in n$ and $\bigtriangledown$ be an n-ary intensional operator with $\|\bigtriangledown\| = \Theta$ universal and conjunctive. Let $M = \{I_\alpha\}_{\alpha \in S}$ be a family of intensional Herbrand models of $\bigtriangledown(A_0, \ldots, A_{n-1})$, that is, for all $I_\alpha \in M$, $\models_{I_\alpha} \bigtriangledown(A_0, \ldots, A_{n-1})$. Then $\models_{\sqcap M} \bigtriangledown(A_0, \ldots, A_{n-1})$.*

Proof. We will outline the proof. Since Θ is universal and conjunctive, for all $w \in \mathcal{U}$, $\Theta|_w \neq \emptyset$ by lemma 4.3 and $\cap\Theta|_w \in \Theta|_w$ by lemma 4.4. Given for any $w \in \mathcal{U}$ and for any $I_\alpha \in M$, $< \|A_0\|^{I_\alpha}, \ldots, \|A_{n-1}\|^{I_\alpha} > \in \Theta|_w$, we have that $< \|A_0\|^{\sqcap M}, \ldots, \|A_{n-1}\|^{\sqcap M} > = < \cap_{\alpha \in S}\|A_0\|^{I_\alpha}, \ldots, \cap_{\alpha \in S}\|A_{n-1}\|^{I_\alpha} > \in \Theta|_w$, because $\Theta|_w$ is closed under intersection by lemma 4.4. But this means that $\sqcap M$ is a model as well. $\dashv$

The family of models of a formula of the form $\Diamond A$ is not closed under the $\sqcap$-intersection where $\Diamond$ is the *S5* possibility operator. Indeed, take all models which

assign a different singleton set to A, then the denotation of A in the $\sqcap$-intersection is the emptyset therefore $\Diamond A$ is false. We state the following corollary to lemma 5.4.

Corollary 5.5 *Let $A_i \in B_{\mathcal{P}}$ for all $i \in n$ and $\bigtriangledown$ be an n-ary intensional operator with $\|\bigtriangledown\| = \Theta$ universal and conjunctive. Let $M = \{I_\alpha \mid \models_{I_\alpha} \bigtriangledown(A_0, \ldots, A_{n-1})\}_{\alpha \in S}$ be the family of intensional Herbrand models of $\bigtriangledown(A_0, \ldots, A_{n-1})$. Then $\sqcap M =_{def} \sqcap_{\alpha \in S} I_\alpha \in M$. Moreover $< \|A_0\|^{\sqcap M}, \ldots, \|A_{n-1}\|^{\sqcap M} >$ is equal to $\cup_{w \in \mathcal{U}} \cap \Theta|_w$.*

Proof. The first part of the corollary follows from lemma 5.4. As regards the second part of the lemma, we have that

$$\cup_{w \in \mathcal{U}} \cap \Theta|_w \leq < \cap_{\alpha \in S} \|A_0\|^{I_\alpha}, \ldots, \cap_{\alpha \in S} \|A_{n-1}\|^{I_\alpha} >$$

since $\cup_{w \in \mathcal{U}} \cap \Theta|_w$ is the least element in $\Theta|_w$. Moreover, one of the models of $\bigtriangledown(A_0, \ldots, A_{n-1})$ must assign to each A_i X_i from $< X_0, \ldots, X_{n-1} >= \cup_{w \in \mathcal{U}} \cap \Theta|_w$. Thus we have the inclusion relation $\leq$ in the opposite direction as well. $\dashv$

We next show that the model intersection property smoothly extends to a family of intensional Herbrand models of an intensional logic program in which no operator whose denotation is non-universal or non-conjunctive appears in the head of any clause.

Lemma 5.6 *Let $\mathcal{P}$ be an intensional logic program and $M = \{I_\alpha\}_{\alpha \in S}$ be a non-empty family of intensional Herbrand models of $\mathcal{P}$. Then $\sqcap M$ is an intensional Herbrand model of $\mathcal{P}$, i.e., $\models_{\sqcap M} \mathcal{P}$.*

Proof. Suppose $\sqcap M$ is not a model of $\mathcal{P}$. Then there is a ground instance of an intensional clause in $\mathcal{P}$ of the form $A \leftarrow B_0, \ldots, B_{n-1}$ which is false in $\sqcap M$ at some $w \in \mathcal{U}$. That means all B_i's are true, but A is false in $\sqcap M$ at w. Since all B_i's are true at w in $\sqcap M$ and all the intensional operators in B_i's have the monotonicity property, it must be the case that all B_i's are true at w in all $I_\alpha \in M$. This implies that A must be true at w in all $I_\alpha \in M$. Therefore A is true in $\sqcap M$ at w by lemma

5.4, which is a contradiction to the assumption that $\sqcap M$ is not a model of $\mathcal{P}$. Thus $\sqcap M$ is a model of $\mathcal{P}$. $\dashv$

Let $\mathcal{P} = \{\Diamond \mathrm{p} \leftarrow\}$ be an intensional logic program where $\Diamond$ is the possibility operator. It can be verified that the model $\sqcap$-intersection property does not hold for $\mathcal{P}$, since lemma 5.4 no longer applies.

The Minimum Intensional Herbrand Model

The following theorem states that there is a model of an intensional logic program called the minimum intensional Herbrand model, which, as far as declarative semantics is concerned, is all we need to know about the program.

Theorem 5.7 *Every intensional logic program $\mathcal{P}$ has a $\sqsubseteq$-minimum intensional Herbrand model $M_{\mathcal{P}}$, which is the model $\sqcap$-intersection of all intensional Herbrand models of $\mathcal{P}$.*

Proof. Let $\mathcal{F}(\mathcal{P}) = \{I_\alpha \mid \models_{I_\alpha} \mathcal{P}\}_{\alpha \in S}$ be the family of all intensional Herbrand models of $\mathcal{P}$. Then $\mathcal{F}(\mathcal{P})$ is non-empty by lemma 5.3. Therefore $M_{\mathcal{P}} =_{def} \sqcap \mathcal{F}(\mathcal{P})$ is the minimum model of $\mathcal{P}$ by lemma 5.6. $\dashv$

The theorem given below states that an intensional unit $\bigtriangledown(A_0, \ldots, A_{n-1})$ is a logical consequence of an intensional logic program $\mathcal{P}$ at a given world $w \in \mathcal{U}$ iff $w \in \|\bigtriangledown(A_0, \ldots, A_{n-1})\|^{M_{\mathcal{P}}}$ and $\models_{M_{\mathcal{P}},w} \mathcal{P}$ at any world $w \in \mathcal{U}$. As far as the following theorem is concerned, we do not have to impose any restrictions on $\bigtriangledown$, but again for computability reasons, $\|\bigtriangledown\|$ should be monotonic and finitary, but does not have to be universal or conjunctive.

Theorem 5.8 *Let $\mathcal{P}$ be an intensional logic program and $A_i \in B_{\mathcal{P}}$ for all $i \in n$. Then $\bigtriangledown(A_0, \ldots, A_{n-1})$ is a logical consequence of $\mathcal{P}$ at any $w \in \mathcal{U}$ iff $\mathcal{P} \models_{M_{\mathcal{P}},w} \bigtriangledown(A_0, \ldots, A_{n-1})$.*

Proof. $\bigtriangledown(A_0, \ldots, A_{n-1})$ is logical consequence of $\mathcal{P}$ at the $w \in \mathcal{U}$
iff $\mathcal{P} \cup \{\neg\bigtriangledown(A_0, \ldots, A_{n-1})\}$ is unsatisfiable at w in any intensional model of $\mathcal{P}$
iff no intensional Herbrand model of $\mathcal{P}$ satisfies $\mathcal{P} \cup \{\neg\bigtriangledown(A_0, \ldots, A_{n-1})\}$ at w by lemma 5.2
iff $\mathcal{P} \not\models_{I,w} \neg\bigtriangledown(A_0, \ldots, A_{n-1})$ for all intensional Herbrand models I of $\mathcal{P}$
iff $\mathcal{P} \models_{I,w} \bigtriangledown(A_0, \ldots, A_{n-1})$ for all intensional Herbrand models I of $\mathcal{P}$
iff $\mathcal{P} \models_{M_\mathcal{P},w} \bigtriangledown(A_0, \ldots, A_{n-1})$ by theorem 5.7. $\dashv$

5.4 The Fixpoint Semantics

The fixpoint theory of Horn logic programs can be modified for further extensions of the theory of intensional logic programs as well. The continuous mapping $T_\mathcal{P}$ originally given in [vEK76] provides the basis for fixpoint semantics and therefore establishes the connection between the model-theoretic and operational semantics of Horn logic programs. The major result of the fixpoint theory of logic programs is that $lfp(T_\mathcal{P}) = T_\mathcal{P} \uparrow \omega = M_\mathcal{P}$. Baudinet [Bau88] [Bau89] shows that the model-theoretic and fixpoint semantics of Templog [AM87] programs coincide. Fitting [Fit88] and Blair [B$^+$88] provide fixpoint semantics for multiple-valued logic programming schemes. We will provide a general result for intensional logic programming. Recall that only those (intensional) operators whose denotations are universal, monotonic, conjunctive and finitary are allowed in intensional logic programs.

The Mapping $T_\mathcal{P}$

Let $\mathcal{P}$ be an intensional logic program. Recall that $\mathcal{F}(\mathcal{P})$ denotes the family of intensional Herbrand interpretations of $\mathcal{P}$. Then $T_\mathcal{P}$, which is the one step modus ponens function for intensional logic programs, is defined as follows. Let H be an intensional Herbrand interpretation of $\mathcal{P}$. Let $T_\mathcal{P} \in [\mathcal{F}(\mathcal{P}) \rightarrow \mathcal{F}(\mathcal{P})]$ where we want

$T_{\mathcal{P}}(H)$ to be an intensional Herbrand interpretation satisfying the model-theoretical condition given below:

$$w \in \|A\|^{T_{\mathcal{P}}(H)} \text{ iff } w \in \|B_i\|^H \text{ for all } i \in n$$

where $A \leftarrow B_0, \ldots, B_{n-1}$ is a ground instance of some intensional program clause in $\mathcal{P}$. However, this condition does not explicitly specify what the intensions assigned to the ground intensional atoms in $B_{\mathcal{P}}$ should be.

Let us reflect. The major problem in defining $T_{\mathcal{P}}$ is that intensional operators may appear in the heads of program clauses in $\mathcal{P}$. Then A is possibly an intensional unit of the form $\bigtriangledown(A_0, \ldots, A_{m-1})$ where each $A_i \in B_{\mathcal{P}}$. Then $T_{\mathcal{P}}(H)$ actually reads as follows :

$$w \in \|\bigtriangledown(A_0, \ldots, A_{m-1})\|^{T_{\mathcal{P}}(H)} \text{ iff } w \in \|B_i\|^H \text{ for all } i \in n$$

We must elaborate what $T_{\mathcal{P}}(H)$ assigns to each A_i. In case $\bigtriangledown$ is the empty sequence and $m = 1$, we have that $\|\bigtriangledown\| = \lambda X.X$. Let $\Theta = \|\bigtriangledown\|$ and $S = \{w \in \mathcal{U} \mid w \in \|B_i\|^H \text{ for all } i \in n\}$. Under the assumption that Θ is universal and conjunctive and hence monotonic, we have that $\|A_i\|^{T_{\mathcal{P}}(H)} = \cup_{w \in S}(\cap\Theta|_w)_i$ for all $i \in m$, so that $w \in \|A\|^{T_{\mathcal{P}}(H)}$ for all $w \in S$. If A_i's appear in the head of any other ground instances, the intension assigned to each A_i is the union of all the intensions induced by each ground instance. Without monotonicity, the condition cannot be satisfied in general.

For any given conjunctive Θ, for all $w \in \mathcal{U}$, we call $\cap\Theta|_w$ the *cluster* of $\Theta|_w$ which is the least element in $\Theta|_w$. In other words, any element of $\Theta|_w$ other than its cluster may contain unnecessary worlds which are not really required to assure that w is included in the output of the function. Then the following is the formal definition of the mapping $T_{\mathcal{P}}$ respecting clusters.

Definition 5.4 *Let $\mathcal{P}$ be an intensional logic program and H be an intensional Herbrand interpretation of $\mathcal{P}$. Then $T_{\mathcal{P}}(H)$ is an intensional Herbrand interpretation*

defined as follows: for all $A \in B_{\mathcal{P}}$,

$$\|A\|^{T_{\mathcal{P}}(H)} = \bigcup \{ X_i \mid \bigtriangledown(A_0, \ldots A_{i-1}, A, A_{i+1}, \ldots, A_{m-1}) \leftarrow B_0, \ldots, B_{n-1}$$
$$\textit{is a ground instance of a clause in } \mathcal{P} \textit{ and for some } w \in \mathcal{U},$$
$$w \in \cap_{k \in n} \|B_k\|^H \textit{ and } < X_0, \ldots, X_{m-1} > = \cap(\|\bigtriangledown\||_w) \quad \}$$

If $\|\bigtriangledown\| = \Theta$ is not conjunctive, then the cluster of $\Theta|_w$ for some $w \in \mathcal{U}$ is not an element of $\Theta|_w$. Consider the following intensional logic program: $\mathcal{P} = \{\Diamond \text{ p} \leftarrow\}$. Let $H_\emptyset$ denote $\sqcap\mathcal{F}(\mathcal{P})$. We have that for all $A \in B_{\mathcal{P}}$, $\|A\|^{H_\emptyset} = \emptyset$ and $\cap(\|\Diamond\||_w) = \emptyset$ for all $w \in \mathcal{U}$. Then $T_{\mathcal{P}}(H_\emptyset) = H_\emptyset$, which means that $H_\emptyset$ is a fixpoint of $T_{\mathcal{P}}$, indeed, the least one, and yet it is not a model of $\mathcal{P}$! Similar anomalies occur when $\|\bigtriangledown\|$ is not universal or not monotonic.

Properties of $T_{\mathcal{P}}$

The mapping $T_{\mathcal{P}}$ shares the properties of that of ordinary logic programs defined in [vEK76] and [Llo84], and those of multiple-valued schemes of Fitting [Fit88] and Blair [B+88], provided that all the constraints are met. We will first prove that $T_{\mathcal{P}}$ is continuous. Let $\mathcal{P}$ be an intensional logic program in which all intensional operators have the desired properties.

Lemma 5.9 *Let* $\mathcal{P}$ *be an intensional logic program. Then* $T_{\mathcal{P}}$ *is continuous, that is, for any chain* $C =< C_n >_{n \in \omega}$ *of intensional Herbrand interpretations of* $\mathcal{P}$, $T_{\mathcal{P}}(\sqcup_{n \in \omega} C_n) = \sqcup_{n \in \omega} T_{\mathcal{P}}(C_n)$.

Proof. The proof is a pointwise extension of that of the continuity of $T_{\mathcal{P}}$ from [Llo84]. It suffices to show that for any ground intensional atom A in $B_{\mathcal{P}}$ and or any $w \in \mathcal{U}$,

$$w \in \|A\|^{T_{\mathcal{P}}(\sqcup_{n \in \omega} C_n)} \text{ iff } w \in \|A\|^{\sqcup_{n \in \omega} T_{\mathcal{P}}(C_n)}$$

We proceed as follows: $w \in \|A\|^{T_{\mathcal{P}}(\sqcup C)}$
iff for some ground instance $\bigtriangledown(A_0, \ldots, A_{m-1}) \leftarrow B_0 \ldots, B_{n-1}$ of some clause in $\mathcal{P}$

with $A = A_i$ for some $i \in m$ and $w \in X_i$ where $< X_0, \ldots, X_{m-1} >= \cap(\|\bigtriangledown\|_v)$ for some $v \in \|B_k\|^{\sqcup C}$ for all $k \in n$ by the definition of $T_{\mathcal{P}}$

iff for some $C_\alpha \in C$, $v \in \|B_k\|^{C_\alpha}$ for all $k \in n$, since the denotations of all intensional operators that appear in $\mathcal{P}$ are continuous

iff $w \in \|A\|^{T_{\mathcal{P}}(C_\alpha)}$

iff $w \in \bigsqcup_{\alpha\in\omega} \|A\|^{T_{\mathcal{P}}(C_\alpha)}$

iff $w \in \|A\|^{\sqcup_{\alpha\in\omega} T_{\mathcal{P}}(C_\alpha)}$. $\dashv$

Here lemma 5.9 fails if an intensional operator with a non-finitary denotation is used in the body of some clause in $\mathcal{P}$. For instance, consider the following temporal logic program:

$$\mathcal{P} = \{\, \mathtt{p(a)} \leftarrow \Box \mathtt{q(a)},\ \mathtt{q(X)} \leftarrow \,\}$$

where $\Box$ is the *S5* necessity operator. It will be shown that $T_{\mathcal{P}}$ is not continuous. Pick an ω-chain $< C_n >_{n\in\omega} =< C_0, C_1, \ldots >$ of intensional Herbrand interpretations of $\mathcal{P}$ where each C_n is defined as follows : $\|\mathtt{p(a)}\|^{C_n} = \emptyset$ and $\|\mathtt{q(a)}\|^{C_n} = \{w_0, w_1, \ldots, w_{n-1}\} \subseteq \mathcal{U}$. Then we have that $\|\mathtt{q(a)}\|^{\sqcup_{n\in\omega} C_n} = \mathcal{U}$, which implies that $\|\mathtt{p(a)}\|^{T_{\mathcal{P}}(\sqcup_{n\in\omega} C_n)} = \mathcal{U}$, but $\|\mathtt{p(a)}\|^{\sqcup_{n\in\omega} T_{\mathcal{P}}(C_n)} = \emptyset$. Therefore $T_{\mathcal{P}}$ is not continuous, that is, $T_{\mathcal{P}}(\sqcup_{n\in\omega} C_n) \neq \sqcup_{n\in\omega} T_{\mathcal{P}}(C_n)$.

Similarly, the continuity of $T_{\mathcal{P}}$ fails if any intensional operator with a non-monotonic denotation is used in $\mathcal{P}$. This can be verified from this single line program: $\mathcal{P} = \{\mathtt{p} \leftarrow \neg\mathtt{q}\}$. Suppose that $\mathcal{U} = \{0\}$ and the language has no intensional operators. Then, by ignoring the generality of the definition of intensional Herbrand interpretations, subsets of the Herbrand base of $\mathcal{P}$ can be regarded as (intensional) Herbrand interpretations. We have that $\emptyset \subset \{\mathtt{q}\}$, but $T_{\mathcal{P}}(\emptyset) = \{\mathtt{p}\}$ and $T_{\mathcal{P}}(\{\mathtt{q}\}) = \emptyset$, which implies that $T_{\mathcal{P}}(\emptyset) \supset T_{\mathcal{P}}(\{\mathtt{q}\})$; therefore $T_{\mathcal{P}}$ is not even monotonic. This fact is also noted in [Llo84] for ordinary logic programs with negation.

The following lemma is also needed in order to characterise intensional Herbrand models of $\mathcal{P}$ in terms of $T_{\mathcal{P}}$.

Lemma 5.10 *Let $\mathcal{P}$ be an intensional logic program. Let I be an intensional Herbrand interpretation of $\mathcal{P}$. Then I is a model of $\mathcal{P}$ iff $T_{\mathcal{P}}(I) \sqsubseteq I$.*

Proof. The proof is again a pointwise extension of the characterisation of Herbrand models in terms of $T_{\mathcal{P}}$ [Llo84] and therefore is omitted. ⊣

Recall that $H_{\emptyset}$ is the bottommost interpretation in the complete lattice $\mathcal{F}(\mathcal{P})$ of intensional Herbrand interpretations of $\mathcal{P}$, ordered by $\sqsubseteq$. Then the following theorem gives the fixpoint characterisation of the minimum model of an intensional logic program $\mathcal{P}$. Its proof is adapted from that of ordinary logic programming [Llo84]. By definition 2.18, recall that $T_{\mathcal{P}} \uparrow \omega =_{def} \sqcup\{T_{\mathcal{P}} \uparrow n \mid n \in \omega\}$.

Theorem 5.11 *Let $\mathcal{P}$ be an intensional logic program. Then $lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega = M_{\mathcal{P}}$.*

Proof. Since $T_{\mathcal{P}}$ is continuous by lemma 5.9, it follows that the closure ordinal of $T_{\mathcal{P}}$ is $\leq \omega$ by lemma 2.8. Thus the least fixpoint of $T_{\mathcal{P}}$, $lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$. We also have that $M_{\mathcal{P}} = \sqcap\{I \mid T_{\mathcal{P}}(I) \sqsubseteq I\}$ by lemma 5.10 and theorem 5.7. By Knaster-Tarski fixpoint theorem (theorem 2.7), $lfp(T_{\mathcal{P}}) = \sqcap\{I \mid T_{\mathcal{P}}(I) \sqsubseteq I\}$. Therefore $lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega = M_{\mathcal{P}}$. ⊣

5.5 Applying The Theory

We have shown that each constraint on intensional operators has model-theoretical implications on the semantics of ILP. However, it is possible to relax some of these constraints, because we do not need all of them to prove each theorem, except monotonicity. Finitariness is not needed for intensional operators used in the heads of intensional program clauses. Conjunctivity and universality are not needed for intensional operators used in the bodies of intensional program clauses. We can now formulate the following meta-theorem of the theory of intensional logic programming:

Theorem 5.12 *Let $\mathcal{P}$ be an intensional logic program. Then $\mathcal{P}$ has the minimum model property which can also be characterised by the least fixpoint of the mapping $T_{\mathcal{P}}$ provided that the denotations of all (intensional) operators that appear in the heads of the clauses in $\mathcal{P}$ are universal, monotonic and conjunctive, and the denotations of all (intensional) operators that appear in the bodies of the clauses in $\mathcal{P}$ are monotonic and finitary.*

Proof. We will give the outline of the proof. If all the intensional operators appearing in the heads have monotonicity and universality properties, the model existence lemma (lemma 5.3) holds for $\mathcal{P}$. To prove that the model-intersection property (lemma 5.6) holds for $\mathcal{P}$, we need to use monotonicity for all intensional operators in $\mathcal{P}$, and conjunctivity for the intensional operators appearing in the heads. Then the minimum model semantics for $\mathcal{P}$ follows. As for the fixpoint semantics, we need to use monotonicity, universality, and conjunctivity for the intensional operators appearing in the heads, and monotonicity and finitariness, that is, continuity, for the intensional operators appearing in the bodies. Then the theorem 5.11 holds. ⊣

In short, the precondition of the theorem is sufficient to prove individual theorems about intensional logic programs, as can be seen from the proofs of the results presented so far. The intensional operators of the languages described in the previous chapters can be used anywhere in programs, because it can be shown that their denotations have all of the desired properties. This entails that all of these languages are contained in our theory by theorem 5.12. Therefore there is no need to provide separate model-theoretic and fixpoint semantics for each language.

Consider the extension of Chronolog with negative moments in time where $\mathcal{U}$ is the set of integers, $\mathcal{Z}$, and the underlying language is extended with the previous moment operator **pre**. The semantics of **pre** is given in terms of $\models$ relation: $\models_{I,t}$ **pre** A iff $\models_{I,t-1} A$. The denotation of **pre** can be defined as follows:

$$\|\mathbf{pre}\| = \lambda X.\{t \in \mathcal{Z} \mid t-1 \in X\}$$

It can be shown that ||pre|| has all the desired properties, therefore temporal logic programs in Chronolog with negative time enjoy the theorem 5.12 as well.

Chronolog can also be extended with $\Box$ and $\Diamond$ plus monotonic formulas. We just disallow $\Box$ in the bodies and $\Diamond$ in the heads of intensional program clauses. Then theorem 5.12 is still valid for extended Chronolog programs. However, later in the dissertation, we will show some of these extensions can be accommodated without any extra temporal operators.

As for extensional operators like $\wedge$ and $\vee$, $\| \wedge \| = \cap$ is monotonic, universal, finitary and conjunctive; $\| \vee \| = \cup$ is monotonic, universal and finitary but not conjunctive, which means that $\vee$ can not be used in the heads of intensional Horn clauses. For instance, both $\wedge$ and $\vee$ can be used in ordinary logic programs with $\vee$ disallowed in the heads of program clauses.

A number of other ILP languages enjoy our results as well:

5.5.1 Temporal Logic Programming

Templog originally proposed by Abadi and Manna [AM87] is based on a temporal logic with temporal modalities $\Diamond$ and $\Box$ as well as the next time operator $\bigcirc$, same as **next** from Chronolog, and allows monotonic formulas in the bodies of program clauses. Let $B_0, \ldots, B_{n-1}$ be monotonic formulas and C an atomic formula. In Templog there are two kinds of program clauses:

- Permanent clauses of the form $\Box(\bigcirc^k C \leftarrow B_0, \ldots, B_{n-1})$.
- Initial clauses of the form $\bigcirc^k C \leftarrow B_0, \ldots, B_{n-1}$ or $\Box \bigcirc^k C \leftarrow B_0, \ldots, B_{n-1}$.

Initial clauses are interpreted as assertions true at the initial moment in time, whereas permanent clauses are interpreted as assertions true at all moments in time. Therefore the temporal operator `first` is implicitly available in Templog.

In our semantic approach, the necessity operator $\Box$ is implicit; we regard all program clauses as assertions true at all moments in time. Then Templog with an explicit initial time operator such as **first** is an instance of intensional logic programming where initial clauses are turned into equivalent permanent clauses by applying **first** to the whole clause by the following. Let $A \leftarrow B_0, \ldots, B_{n-1}$ be an initial clause where $B_0, \ldots, B_{n-1}$ are monotonic formulas. If A is of the form $\Box \bigcirc^k C$ for an atom C, then the corresponding permanent clause is

$$\bigcirc^k C \leftarrow \texttt{first}\, B_0, \ldots, \texttt{first}\, B_{n-1}$$

otherwise it is

$$\texttt{first} \bigcirc^k C \leftarrow \texttt{first}\, B_0, \ldots, \texttt{first}\, B_{n-1}$$

It can be shown that this transformation is correct and preserves model-hood.

Then the next question is if Templog with **first** enjoys the minimum model semantics described in this chapter. Recall that the denotations of temporal $\Diamond$ and $\Box$ are given as:

$$\|\Box\| = \lambda X.\{t \in \omega \mid X \supseteq \{z \in \omega \mid z \geq t\}\}$$
$$\|\Diamond\| = \lambda X.\{t \in \omega \mid X \cap \{z \in \omega \mid z \geq t\} \neq \emptyset\}$$

It can be shown that $\|\Box\|$ and $\|\Diamond\|$ share the properties of the denotations of *S5* modalities $\Box$ and $\Diamond$. Moreover, Abadi and Manna [AM87] restrict the use of $\Diamond$ to the bodies of temporal clauses, and the use of $\Box$ to the heads of temporal claues. Therefore the pre-conditions of theorem 5.12 are satisfied by Templog programs.

TL1 is a fragment of the Templog language [AM87] and later mentioned in [Bau88] and [Bau89]. TL1 with an implicit initial time operator is the same as Chronolog, and by a similar argument given above, it is covered in the theory. Baudinet [Bau88] [Bau89] has independently developed the model-theoretical semantics of Templog programs by showing that any given Templog program is true in a temporal Herbrand interpretation iff all strictly ground instances of each clause

in the program are true. In fact, the set of all strictly ground instances of a clause can be regarded as an infinite set of TL1 clauses; and the theory can be extended to include such infinitary programs in a rather straightforward manner. Then for any Templog program there exists an equivalent infinite TL1 program to which the theory applies.

Temporal Prolog introduced by Gabbay [Gab87] allows in the heads of the program clauses modal operators such as possible in the future and possible in the past, denoted by F and P, and modal/temporal programs may include clauses whose heads contain other clauses. Even if we consider clausal subsets of this language, we have a negative result : Gabbay's system is not contained in our theory; one of the reasons being that $\|F\|$ and $\|P\|$ are non-conjunctive, which implies that Temporal Prolog does not have the minimum model and fixpoint semantics as we have developed.

5.5.2 Modal Logic Programming

Molog proposed by del Cerro [dC86] is rather a framework for modal logic programming. In [dC86], the semantics of modal operators are defined in terms of $\models$ relation, and modal operators are grouped in two categories, universal and existential, and furthermore, no constraints are imposed on the use of modal operators. In [BdCH88], the declarative semantics of an instance of Molog is developed in terms of trees, and certain transformations on trees. We will summarize some features of this language and show that it is contained in the theory.

In [BdCH88], first, a language with only two operators, namely $\Box$ and $\Diamond$, is described; then a Kripke-style semantics for it is provided. With $\Diamond$ and $\Box$ disallowed in the heads and in the bodies of modal clauses respectively, the pre-condition of theorem 5.12 is satisfied. Then the language is enriched with a countable set of modal operators, each of which refers to some possible world in a way the temporal

operators **first** and **next** does, e.g., if $<\pi>$ is such an operator, then $\models_{I,w} <\pi> A$ iff $\models_{I,f_\pi(w)} A$, where $f_\pi \in [\mathcal{U} \to \mathcal{U}]$. For the temporal operator **first**, the corresponding function would map any given time to 0. The denotations of those operators can be obtained as follows:

$$\|<\pi>\| = \lambda X.\{w \in \mathcal{U} \mid f_\pi(w) \in X\}$$

It can be verified that each $\|<\pi>\|$ has all the desired properties, therefore, with constraints on the use of $\Box$ and $\Diamond$, we conclude that theorem 5.12 applies to this particular language.

InTense proposed by Mitchell and Faustini [MF89] is a multi-dimensional intensional logic programming language and it supports any finite number of temporal and spatial dimensions. Thus a possible world in InTense is simply a point in a time-space hyperfield. For instance, if we have m temporal and n spatial dimensions, the set of possible worlds $\mathcal{U}$ is given as $\mathcal{Z}^{m+n}$. In general, we may take $\mathcal{U}$ to be $\mathcal{Z}^{\omega}$. InTense provides users with unary intensional operators for each dimension by the following: $\texttt{prior}_k$, $\texttt{initial}_k$ and $\texttt{rest}_k$ are associated with the k^{th} spatial dimension; and similarly, $\texttt{prev}_k$, $\texttt{first}_k$ and $\texttt{next}_k$ are associated with the k^{th} temporal dimension.

The semantics of temporal operators $\texttt{prev}_k$, $\texttt{first}_k$ and $\texttt{next}_k$ are similar to those of Chronolog with negative time, the only difference being that they operate on the k^{th} temporal dimension. The semantics of spatial operators $\texttt{prior}_k$, $\texttt{initial}_k$ and $\texttt{rest}_k$ are counterparts of temporal operators over the k^{th} spatial dimension. Therefore all of the intensional operators of InTense share the desired properties, which in turn implies that *pure* InTense programs enjoy the minimum model semantics.

5.5.3 Interval Logic Programming

Interval type temporal logic programming languages such as Tokio [AFMo86] and Tempura [Mos86] have features which go beyond pure logic programming. In fact, Tempura is not even based on the Horn-clause subset of logic. As for Tokio, Aoyagi et al [ÅFMo86] do not clearly specify the syntax of Tokio, but intensional operators of Tokio are in fact those of Tempura. In Tokio, the variables are intensional, i.e., the meanings of variables depend on moments in time, and there are even temporal operators that can be applied to the terms of the language. However, the fact that the minimum model semantics does not apply to Tempura and especially Tokio does not mean that it cannot be applied to interval logic programming at all. As a matter of fact, it is known that an interval logic can be embedded into a 2-dimensional logic by transformation [vB88]. This result suggests that, if all the constraints are satisfied, interval logic programming enjoys the minimum model semantics.

An interval is a pair of natural numbers $[x, y]$ where $x \leq y$. Let $Sub([x, y]) =_{def} \{[m, n] \mid x \leq m \leq n \leq y\}$, i.e., $Sub([x, y])$ is the set of all sub-intervals of $[x, y]$. In interval logic, the satisfaction relation can be defined over intervals and temporal interpretations. The semantics of interval operators of Tempura are defined as follows. Let I be a temporal interpretation and $[m, n]$ an interval.

- $\models_{I,[m,n]} \Box A$ iff $\models_{I,[x,n]} A$ for all $[x, n] \in Sub([m, n])$
- $\models_{I,[m,n]} \Diamond A$ iff $\models_{I,[x,n]} A$ for some $[x, n] \in Sub([m, n])$
- $\models_{I,[x,y]} \mathtt{cbop}(A, B)$ iff $\models_{I,[x,n]} A$ and $\models_{I,[n,y]} B$
  where $x < n \leq y$ or $x \leq n < y$
- $\models_{I,[x,y]} \mathbf{next}\, A$ iff $x < y$ and $\models_{I,[x+1,y]} A$
- $\models_{I,[x,y]} \mathbf{weaknext}\, A$ iff $x = y$ or $\models_{I,[x,y]} \mathbf{next}\, A$
- $\models_{I,[x,y]} \mathbf{keep}\, A$ iff $x < y$ and $\models_{I,[x,y-1]} A$

- $\models_{I,[x,y]} \mathtt{fin}\, A$ iff $\models_{I,[y,y]} A$

The semantics of atomic formulas can be defined as follows.

- $\models_{I,[x,y]} p(e_0,\ldots,e_{n-1})$ iff $x \in \|p(e_0,\ldots,e_{n-1})\|^I$

Note that there are surely other ways of defining the semantics of atomic formulas.

Consider the intensional logic $IL = IA^{\models}$ where $\mathcal{U} = \{< x,y > \in \omega \times \omega \mid x \leq y\}$, and $IA = \{\| \odot \|, \|\Box\|, \|\Diamond\|, \ldots\}$. The semantics of intensional operators are defined as in interval logic, but in terms of pairs in $\mathcal{U}$. The extra operator $\odot$ projects any given world onto the main diagonal in $\mathcal{U}$, i.e., $\models_{I,<x,y>} \odot A$ iff $\models_{I,<x,x>} A$. Here the world $< x,x >$ can be interpreted as a moment in time, that is, x. We need the extra operator $\odot$, because the semantics of atomic formulas of IL are defined for all pairs in $\mathcal{U}$. Then the transformation procedure from interval logic to this 2-dimensional logic adds the new operator $\odot$ in front of every atomic formula A. This way, the semantics of A in terms of intervals coincides with the semantics of $\odot A$ in terms of pairs in $\mathcal{U}$. Moreover, IL inherits all the properties of operators in the interval logic as well as the function *Sub* with intervals interpreted as possible worlds.

The next step is to transform a given interval logic program $\mathcal{P}$ into this 2-dimensional logic. Then we just check whether the transformed program satisfies the pre-conditions of the theorem 5.12. If so, the minimum intensional Herbrand model of the interval logic program may be constructed from that of the transformed program in IL, by carrying everything from the worlds along the main diagonal in $\mathcal{U}$ over to a model of $\mathcal{P}$, with worlds of the form $< x,x >$ interpreted as moments in time.

If we restrict Tokio to extensional variables, and strip Tokio off all of its structures which go beyond our framework, it can be shown that the resulting interval language enjoys the minimum model semantics. We have that the denotations of all interval operators of Tokio are universal, monotonic, and finitary (all intervals are of finite

length). On the other hand, $\|\Diamond\|$ and the chop operator $\|\texttt{chop}\|$ are not conjunctive, which in turn implies that $\Diamond$ and chop can not be used in the heads of program clauses. In [AFMo86], no example to the contrary is given, therefore, under this assumption, theorem 5.12 is valid by transformation for restricted interval logic programs.

In summary, the objective of a language-independent unified theory for intensional logic programming is two-fold:

- We investigate whether some intensional logic programming language is contained in the theory, and enjoys the properties outlined in this dissertation.
- We use the theory as a template to *design* a new intensional logic programming language with the desired properties.

We do not claim that the theory developed so far is the ultimate way to go about intensional/temporal logic programming, but it clarifies how important the role of each constraint is, and the conditions under which a desired property of intensional logic programs can be guaranteed.

Chapter 6

Choice Predicates Non-Determinism

Intensional logic programs can model dataflow style of stream-oriented computations; a dataflow stream can be represented by a predicate over a set of possible worlds, such as a time-varying predicate over a collection of moments in time. But, due to inherent non-determinism in logic programming, each predicate defined in an intensional logic program does not necessarily specify a unique stream of ground terms. A problem then emerges in connection with stream-oriented communication such as that of the dataflow language Lucid [WA85]. Wadge [Wad85] proposed a logical extension to temporal logic programming, called "choice predicates", through which data-flow style of computations can be expressed. A choice predicate is associated with each predicate used in an intensional logic program, and, in principle, acts like a non-deterministic filter with multiple input lines which randomly selects one of its inputs as its output.

In the following, after introducing choice predicates, we will define the notion of choice formulas and extended programs. An extended program includes non-Horn axioms, as well as program clauses, to establish the connection between predicates

and the corresponding choice predicates. We will then investigate general properties of intensional Herbrand models of such programs. Since the minimum model semantics does not apply to extended programs, the declarative semantics of extended programs will be developed in terms of *minimal* intensional Herbrand models [OW89a], in which case answers to queries are no longer logical consequences of the program, but true in some minimal model. This is how the correctness is defined. We will show that some of these minimal models are not constructible from the program. We will also relate our work to committed choice languages such as Concurrent Prolog [Sha87] and Parlog [CG86].

6.1 Choice Predicates

Logic programming is non-deterministic since there may be more than one possible solution for a given query. When we want to model dataflow style of stream-oriented computations, it is desirable to have exactly one solution for the query chosen arbitrarily amongst all possible solutions (if any). Committed-choice languages support stream-oriented communication and offer dataflow modularity to some extent, but cannot guarantee single-valued solutions in model-theoretical terms. For instance, suppose we want to produce an "arbitrary" stream of natural numbers starting from 0 in strictly increasing order. The following infinitary logic program appears to work.

```
stream([0|L]) <- rest-of-stream([0|L]).
rest-of-stream([X,Y|L]) <- X < Y, rest-of-stream([Y|L]).
```

Given a query like `<- stream(L)`, an implementation of infinitary logic programming produces partial answers to the query, i.e., initial segments of an arbitrary stream of increasing numbers starting with 0, and never terminates.

It is possible to regard `stream(L)` as a dataflow node producing an infinite stream of numbers, one at a time. However, the set of logical consequences of this program is empty. Then the "intended" meaning of the program can be modeled by the greatest fixpoint semantics such as that of [vENA84], based on Herbrand models over a Herbrand universe extended with infinite terms. With respect to the greatest fixpoint semantics, the `stream` predicate represents all increasing sequences of natural numbers. Hence we cannot force the `stream` predicate to represent a single arbitrary stream with respect to any semantic approach.

Let us try to write a Chronolog program to solve the same problem. We would like the time-varying `stream` predicate to represent an arbitrary increasing stream of values over the collection of moments in time. In other words, at any given time $t+1$, the `stream` predicate will be true of a value (number) which is strictly greater than the value generated at time t. The following program appears to work, but does not.

```
first stream(0).
next  stream(Y) <- stream(X), X < Y.
```

What the program really says that if `X` might be the (say) 10th output, and `X` is less than `Y`, then `Y` might be the 11th output. The `stream` predicate is not single valued. In fact, at time $t > 0$, it is true of all $x \geq t$. It can be thought of as a "fuzzy" stream which at each point in time has many possible values; the rule is that every value possible at time $t+1$ is greater than "some" value possible at time t. But there is no way in which we can regard the `stream` predicate as representing an increasing stream or even a family of increasing streams. What we need to say is that the time 11 "possible" outputs are those numbers which are greater than the time 10 "actual" output. This is not possible in Chronolog.

Wadge [Wad85] proposed an extension to temporal logic programming, called "choice predicates". Choice predicates are associated with each predicate that ap-

pear in a given temporal logic program, and they represent arbitrary but definite single-valued relations, extracted from the relation the corresponding predicate represents. Therefore it is possible to regard a choice predicate as representing a stream over the collection of moments in time. However, since there may be more than one value possible (if any) at each moment in time, a choice predicate acts like a non-deterministic filter with multiple input lines which arbitrarily selects one of its inputs as its output. In short, choice predicates offer a dataflow style of non-deterministic stream-oriented communication within temporal logic programming.

We will now show how the above problem can be solved by using choice predicates. The solution to the above problem is to provide an extra predicate `#stream` which succeeds only for the ground term that was actually produced. Then the temporal logic program should be

```
first stream(0).
next  stream(Y) <- #stream(X), X < Y.
```

Here the `#stream` predicate is called a "choice predicate". Then the `#stream` predicate represents a stream of output values chosen non-deterministically from those terms the `stream` predicate represents over the collection of moments in time. Note that the `#stream` predicate represents a single-valued relation at any moment in time. Choices made for the `#stream` predicate at any moment in time will also affect the values which the `stream` predicate is true of.

In a dataflow-structured program some parts are producers and others are consumers; thus every predicate is potentially an "output stream". This means that we must supply choice predicates for every predicate symbol in a temporal logic program. Of course, the programmer is free to use any number of them, or none at all. Note that programmers do not define choice predicates; they are supplied by the implementation. However, we must employ a temporal logic with equality in order to establish the relation between predicates and the corresponding choice

predicates.

The relation between `stream` and `#stream` can be axiomatised in a first-order logic with equality by the following formulas.

F1. $(\forall x)(\texttt{\#stream}(x) \rightarrow \texttt{stream}(x))$.

F2. $(\forall x)(\texttt{\#stream}(x) \rightarrow (\forall y)(\texttt{\#stream}(y) \rightarrow x \doteq y)))$.

F3. $(\exists x)(\texttt{stream}(x)) \leftrightarrow (\exists y)(\texttt{\#stream}(y))$.

We suppose all of these formulas are statements true at all moments in time. The symbol $\doteq$ is the syntactical equality symbol in the underlying language. We interpret these formulas as follows: at any given moment in time, (1) F1 says that if the `#stream` predicate is true of some term, the `stream` predicate is true of the same term as well; (2) F2 asserts the uniqueness of the term which the `#stream` predicate is true of, and (3) F3 says that whenever `stream` is true of some term, the `#stream` predicate is true of some term as well (not necessarily the same term) and vice versa.

Chronolog programs now implicitly include these formulas for each predicate defined in the programs. However, this results in non-Horn temporal logic programs. Therefore the minimum model semantics cannot be applied to temporal logic programs with choice predicates. Answers to queries are no longer logical consequences of the program, because choice predicates represent streams which depend on arbitrary choices made by the implementation. We must be careful in defining the correctness of the results obtained from a given implementation. Essentially, the implementation is constructing a "minimal" model of the program together with the implicit non-Horn axioms.

6.2 Choice Formulas, Extended Programs

Representing streams in a temporal language such as Chronolog may seem quite natural. However, since the very same idea can be naturally exploited in any given intensional language, we will develop a language-independent theory for intensional logic programs with choice predicates. Let *IL* be the underlying language of some intensional logic. We differentiate the choice predicates from the others by the following. Let *Pred* denote the set of all predicate symbols of *IL*, other than the choice predicates. Then the choice predicate related to any $p \in Pred$ is denoted by $\#p$ and can be used only in the bodies of program clauses in intensional logic programs, and in queries.

We now define non-Horn axioms for establishing the relation between predicates and choice predicates. Any intensional logic program in which some choice predicates occur in the bodies of clauses implicitly includes these axioms for each predicate used.

Definition 6.1 (Choice formulas.) *Let $p \in Pred$ be any n-ary predicate symbol. We define Ψ_p as the set of choice formulas associated with p as follows.*

$$\Psi_p = \left\{ \begin{array}{l} (\forall \vec{X})(\#p(\vec{X}) \rightarrow p(\vec{X})), \\ (\forall \vec{X})(\#p(\vec{X}) \rightarrow (\forall \vec{Y})(\#p(\vec{Y}) \rightarrow \vec{X} \doteq \vec{Y})), \\ (\exists \vec{X})(p(\vec{X})) \leftrightarrow (\exists \vec{Y})(\#p(\vec{X})) \end{array} \right\}$$

where $\vec{X} =_{def} < X_0, \ldots, X_{n-1} >$ and $\vec{Y} =_{def} < Y_0, \ldots, Y_{n-1} >$.

Here $\vec{X} \doteq \vec{Y}$ represents the formula $\wedge_{i \in n} X_i \doteq Y_i$. We use $\doteq$ for the equality symbol in the object language in order to distinguish it from the equality symbol $=$ in the meta-language. We will also refer to these three formulas, in the given order, as F1, F2 and F3.

Read choice formulas as assertions true at all possible worlds in $\mathcal{U}$. Intuitively, the choice formulas for any $p \in Pred$ altogether say that at any world w, whenever a

predicate is true of one or more terms, the choice of that predicate is true of a unique term chosen arbitrarily from those terms. Then an intensional logic program $\mathcal{P}$ of *IL* with equality, extended with choice formulas given above, is called an extended intensional logic program. The formal definition is given below.

Definition 6.2 (Extended programs.) *Let $\mathcal{P}$ be an intensional logic program in which choice predicates occur. Then $\mathcal{P} \cup \{\Psi_p \mid p \in Pred\}$ is called an extended intensional logic program.*

The intensional Herbrand base $B_{\mathcal{P}}$ of an intensional logic program $\mathcal{P}$ with choice predicates now includes all those ground atomic formulas obtained from choice predicates and ground terms in the Herbrand universe $U_{\mathcal{P}}$ of $\mathcal{P}$ as well as atomic formulas obtained from non-choice predicates. However, as choice predicates are not defined in $\mathcal{P}$, the correct interpretation of them is only possible in the extended version of $\mathcal{P}$. Recall that an intensional Herbrand interpretation I of $\mathcal{P}$ assigns a subset of the set of possible worlds $\mathcal{U}$ to each ground atom A in $B_{\mathcal{P}}$, i.e., $\|A\|^I \in P(\mathcal{U})$. Since choice formulas do not introduce any extra predicates other than $\doteq$, I is also an interpretation for extended $\mathcal{P}$.

We will treat $\doteq$ in a different way and assume that the denotation of $\doteq$ in *any* intensional Herbrand interpretation is a reflexive binary relation (an identity relation) over the Herbrand universe of a given program, i.e., $\|\doteq\| = \{< e, e > \mid e \in U_{\mathcal{P}}\}$. Therefore, technically, there is no need to include those ground atoms related to the equality in the intensional Herbrand base of any extended program. Since the value of a term e does not depend on the elements of $\mathcal{U}$, neither does $\|\doteq\|$.

We will now formulate some model-theoretical conditions for intensional Herbrand interpretations of an extended program to be a model. This way, there is no need to refer to the syntactic properties and structure of extended programs. Naturally, these model-theoretical conditions will be the semantic counterparts of the choice formulas given above.

Definition 6.3 *Let $\mathcal{P}$ be an extended intensional logic program and I is an intensional Herbrand interpretation of $\mathcal{P}$. For all n-ary predicate symbols $p \in Pred$,*

C1. $\|\#p(\vec{e})\|^I \subseteq \|p(\vec{e})\|^I$ *for all* $\vec{e} \in (U_\mathcal{P})^n$,

C2. *for all* $\vec{e}$ *and* $\vec{t} \in (U_\mathcal{P})^n$ *where* $\vec{e} \neq \vec{t}$, $\|\#p(\vec{e})\|^I \cap \|\#p(\vec{t})\|^I = \emptyset$,

C3. $\bigcup_{\vec{e} \in (U_\mathcal{P})^n} \|\#p(\vec{e})\|^I = \bigcup_{\vec{e} \in (U_\mathcal{P})^n} \|p(\vec{e})\|^I$.

The connection between choice formulas and models of choice formulas can be established through these conditions. The following lemma shows that C1, C2 and C3 are the necessary and sufficient conditions for an intensional Herbrand interpretation to be a model of the choice formulas.

Lemma 6.1 *Let $\mathcal{P}$ be an extended intensional logic program, Ψ the set of choice formulas for $\mathcal{P}$, and I an intensional Herbrand interpretation of $\mathcal{P}$. $\models_I \Psi$ iff I satisfies C1, C2 and C3 for all $p \in Pred$.*

Proof. Since model-theoretical conditions are the counterparts of choice formulas, the lemma can be proved for each one of the three condition-formula pairs. However, it is straightforward to show that I satisfies C1 and C3 iff $\models_I$ F1 and $\models_I$ F3 for all $p \in Pred$. We will give only the detailed proof of the F2–C2 pair.

Suppose I satisfies C2, but not a model of F2 for some $p \in Pred$. Then for some $\vec{e}$ and $\vec{t} \in (U_\mathcal{P})^n$ where $\vec{e} \neq \vec{t}$, F2 is false at some $w \in \mathcal{U}$. This implies that $w \in \|\#p(\vec{e})\|$ and $w \in \|\#p(\vec{t})\|$, which results in the contradiction $w \in \|\#p(\vec{e})\| \cap \|\#p(\vec{t})\|$. Thus I must be a model of F2.

Conversely, suppose I is a model of F2, but does not satisfy C2. Pick any $\vec{e}$ and $\vec{t} \in (U_\mathcal{P})^n$ where $\vec{e} \neq \vec{t}$ and $\|\#p(\vec{e})\| \cap \|\#p(\vec{t})\| \neq \emptyset$. At any w chosen from the intersection, the following instance of F2 is false: $\#p(\vec{e}) \rightarrow (\#p(\vec{t}) \rightarrow \vec{e} \doteq \vec{t})$. Thus I must satisfy C2. $\dashv$

The intensional Herbrand interpretation that corresponds to the entire Herbrand base of an ordinary logic program – namely the one that assigns $\mathcal{U}$ to every atom

in $B_{\mathcal{P}}$ – is in general not a model of $\mathcal{P}$, because it may fail to satisfy some of the choice formulas whenever a predicate is true of more than one ground term at a world. In turn, the model-existence lemma 5.3 is lost. Then the question is whether an extended logic program $\mathcal{P}$ is consistent or not. The following lemma gives an affirmative answer.

Lemma 6.2 *Let $\mathcal{P}$ be an extended intensional logic program. Then $\mathcal{P}$ has a model, that is, $\models_I \mathcal{P}$ for some intensional Herbrand interpretation I.*

Proof. We will construct an intensional Herbrand model I of $\mathcal{P}$ as follows: all the ordinary predicates will be true of every term whereas choice predicates will be true of an arbitrarily chosen term. Let $B_{\mathcal{P}}$ be the intensional Herbrand base of $\mathcal{P}$ and t be an arbitrary term in the Herbrand universe of $\mathcal{P}$. We construct I as follows: if $p \in Pred$ is an n-ary predicate symbol, then $\|p(\vec{e})\|^I = \mathcal{U}$ for all $p(\vec{e}) \in B_{\mathcal{P}}$, and $\|\#p(\vec{t})\|^I = \mathcal{U}$ where $\vec{t}$ denotes the n-tuple of terms, each of whose elements is t. It can be verified that I is a model of the intensional Horn clauses in $\mathcal{P}$, because choice predicates do not appear in the heads of program clauses.

In order to show that all choice formulas in $\mathcal{P}$ are also true in I, it suffices to check whether I satisfies C1, C2, and C3. C1 is true, because, for all $\vec{e} \in (U_{\mathcal{P}})^n$ either $\|\#p(\vec{e})\|^I = \|p(\vec{e})\|^I = \mathcal{U}$ (when $\vec{e} = \vec{t}$), or $\|\#p(\vec{e})\|^I = \emptyset$ and $\|p(\vec{e})\|^I = \mathcal{U}$ (when $\vec{e} \neq \vec{t}$). C2 is true, because, for all $\vec{e} \in (U_{\mathcal{P}})^n$, $\vec{e} \neq \vec{t}$ implies $\|\#p(\vec{e})\|^I = \emptyset$. C3 is true, because $\bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^I = \bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \|p(\vec{e})\|^I = \mathcal{U}$. Therefore I is a model of $\mathcal{P}$ by lemma 6.1. $\dashv$

6.3 Minimal Models of Extended Programs

Extended programs have models; but the family of intensional Herbrand models of an extended program does not share the properties of ordinary intensional logic programs. For instance, consider the program given above that defines the `stream`

predicate. It is possible that the **#stream** predicate represents the stream of odd numbers $< 0, 1, 3, 5, \ldots >$ in some intensional Herbrand model, and the stream of even numbers $< 0, 2, 4, 6, \ldots >$ in another. In the model $\sqcap$-intersection of such two models, the **#stream** predicate is true of 0 at the initial moment t_0, and is not true of any term at all the other moments in time. But at time t_1, the **stream** predicate represents the relation $\{n \mid n > 0\}$. This implies that choice formulas are not true at time t_1. Therefore the model $\sqcap$-intersection does not preserve model-hood for arbitrary sets of models.

Since the family of intensional Herbrand models of an extended program $\mathcal{P}$ is not closed under the model $\sqcap$-intersection, extended programs do not enjoy the minimum model semantics. However, the family is not a totally unstructured collection. Indeed, we will prove the following important result which states that the family is closed under the model $\sqcap$-intersection of nonempty downwards chains. Then we can appeal to a fundamental result from set-theory known as Zorn's lemma to show that the family contains minimal intensional Herbrand models characterising the declarative semantics of extended programs.

If we have a nonempty downwards chain of intensional Herbrand models of an extended program $\mathcal{P}$, it suffices to show that the model $\sqcap$-intersection operation over such a chain preserves the necessary and sufficient conditions C1, C2 and C3 for an intensional Herbrand interpretation of $\mathcal{P}$ to be a model.

Lemma 6.3 *Let $\mathcal{P}$ be an extended intensional logic program and $M =< I_\alpha >_{\alpha \in S}$ be a downwards nonempty chain of intensional Herbrand models of $\mathcal{P}$, ordered by $\sqsubseteq$. Then $\sqcap M =_{def} \sqcap_{\alpha \in S} I_\alpha$ is also a model of $\mathcal{P}$.*

Proof. We know that the model $\sqcap$-intersection property holds for intensional program clauses. Then it suffices to show that over a downwards chain of models of $\mathcal{P}$, the model $\sqcap$-intersection preserves all of C1, C2, and C3. Let p be an n-ary predicate symbol in *Pred*.

C1: All of C1, C2 and C3 hold in all models in M. Then we have that,

$$\begin{aligned}(\forall \vec{e} \in (U_{\mathcal{P}})^n)\ (\forall \alpha \in S)\ \|\#p(\vec{e})\|^{I_\alpha} &\subseteq \|p(\vec{e})\|^{I_\alpha} \implies \\ (\forall \vec{e} \in (U_{\mathcal{P}})^n)\ \bigcap_{\alpha\in S} \|\#p(\vec{e})\|^{I_\alpha} &\subseteq \bigcap_{\alpha\in S} \|p(\vec{e})\|^{I_\alpha} \implies \\ (\forall \vec{e} \in (U_{\mathcal{P}})^n)\ \|\#p(\vec{e})\|^{\sqcap_{\alpha\in S} I_\alpha} &\subseteq \|p(\vec{e})\|^{\sqcap_{\alpha\in S} I_\alpha}\end{aligned}$$

which is C1.

C2: We have that for all $\vec{e}$ and $\vec{t} \in (U_{\mathcal{P}})^n$, if $\vec{e} \neq \vec{t}$,

$$\begin{aligned}(\forall \alpha \in S)\ \|\#p(\vec{e})\|^{I_\alpha} \cap \|\#p(\vec{t})\|^{I_\alpha} = \emptyset &\implies \\ (\bigcap_{\alpha\in S} \|\#p(\vec{e})\|^{I_\alpha}) \cap (\bigcap_{\alpha\in S} \|\#p(\vec{t})\|^{I_\alpha}) = \emptyset &\implies \\ \|\#p(\vec{e})\|^{\sqcap_{\alpha\in S} I_\alpha} \cap \|\#p(\vec{t})\|^{\sqcap_{\alpha\in S} I_\alpha} = \emptyset &\end{aligned}$$

which is C2.

C3: We proceed as follows.

$$\begin{aligned}(\forall \alpha \in S)\ \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^{I_\alpha} &= \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \|p(\vec{e})\|^{I_\alpha} \implies \\ \bigcap_{\alpha\in S} \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^{I_\alpha} &= \bigcap_{\alpha\in S} \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \|p(\vec{e})\|^{I_\alpha}\end{aligned}$$

The next step is crucial. We want to switch in the above equation the places of the intersections and unions in order to obtain C3. This can not be done in general. In other words, it suffices to show that

$$\bigcap_{\alpha\in S} \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^{I_\alpha} = \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \bigcap_{\alpha\in S} \|\#p(\vec{e})\|^{I_\alpha}$$

and a similar equation for non-choice predicates. For any $w \in \mathcal{U}$,

$$w \in \bigcap_{\alpha\in S} \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^{I_\alpha} \implies (\forall \alpha \in S)\ w \in \bigcup_{\vec{e}\in(U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^{I_\alpha} \implies$$

$$(\forall \alpha \in S)\ (\exists \vec{e} \in (U_{\mathcal{P}})^n)\ w \in \|\#p(\vec{e})\|^{I_\alpha}$$

Since for all $\vec{e} \in (U_{\mathcal{P}})^n$, $< \|\#p(\vec{e})\|^{I_\alpha} >_{\alpha\in S}$ is a chain, w must appear in every member of exactly one of these chains. Then it is possible to switch the places of

the universal and existential quantifiers in the above formula.

$$\begin{aligned}
(\exists \vec{e} \in (U_{\mathcal{P}})^n)\ (\forall \alpha \in S) \quad & w \in \|\#p(\vec{e})\|^{I_\alpha} && \Longrightarrow \\
(\exists \vec{e} \in (U_{\mathcal{P}})^n) \quad & w \in \bigcap_{\alpha \in S} \|\#p(\vec{e})\|^{I_\alpha} && \Longrightarrow \\
& w \in \bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \bigcap_{\alpha \in S} \|\#p(\vec{e})\|^{I_\alpha} && \Longrightarrow
\end{aligned}$$

$$\bigcap_{\alpha \in S} \bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^{I_\alpha} \subseteq \bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \bigcap_{\alpha \in S} \|\#p(\vec{e})\|^{I_\alpha}$$

The reverse direction of this inclusion can be found in set theory books, e.g., see [KM76]. An analogous equation for non-choice predicates can be given. Therefore we have the desired result:

$$\begin{aligned}
\bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \bigcap_{\alpha \in S} \|\#p(\vec{e})\|^{I_\alpha} &= \bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \bigcap_{\alpha \in S} \|p(\vec{e})\|^{I_\alpha} \Longrightarrow \\
\bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \|\#p(\vec{e})\|^{\sqcap_{\alpha \in S} I_\alpha} &= \bigcup_{\vec{e} \in (U_{\mathcal{P}})^n} \|p(\vec{e})\|^{\sqcap_{\alpha \in S} I_\alpha}
\end{aligned}$$

which is merely C3. Then $\sqcap M$ is a model of choice formulas of $\mathcal{P}$ by lemma 6.1. $\dashv$

Zorn's lemma states the following: if a nonempty family of sets is closed under unions of nonempty chains, then it contains a maximal element, which is not necessarily unique. But we are interested in intersections of nonempty downwards chains and minimal elements. Therefore the following lemma is needed.

Lemma 6.4 (The dual of Zorn's lemma.) [Abi65] *Let X be a nonempty family of sets which is closed under intersections of nonempty downwards chains; then X has a minimal element.*

Lemma 6.3 together with the dual of Zorn's lemma imply the following major result of the model-theory of extended programs:

Theorem 6.5 *Let $\mathcal{P}$ be an extended intensional logic program and $\mathcal{M}(\mathcal{P})$ be the family of all intensional Herbrand models of $\mathcal{P}$. Then $\mathcal{M}(\mathcal{P})$ has a minimal element.*

Proof. We know that $\mathcal{M}(\mathcal{P})$ is non-empty by lemma 6.2. Then the theorem follows from lemmas 6.3 and 6.4. $\dashv$

In short, minimal models constitute the model-theoretical semantics of extended programs. As mentioned earlier, any implementation of intensional logic programming that supports choice predicates must construct one such minimal model of an extended program. Again, this is how the correctness is defined. However, minimal model semantics do not provide us with any information as to which minimal models an implementation can really construct.

6.4 Constructible Minimal Models

The logical structure of an extended program dictates which minimal intensional Herbrand models an implementation can construct. We will illustrate this by an example. Consider the program $\mathcal{P}$ = { `p(a) <-, p(b) <- #p(X)` } and the query `<-#p(b)`. In order to prove `#p(b)`, we use the second clause and obtain another query `<-#p(X)`. Now the first clause cannot be used, because choice predicates represent single-valued relations and hence it contradicts the original query. Then the second clause can be used again, and so on and so forth. There is no proof of the query at any world in $\mathcal{U}$, and yet it is true in some minimal model, e.g., the model that assigns $\mathcal{U}$ to `p(a)`, `p(b)` and `#p(b)`, and $\emptyset$ to `#p(a)`.

Consider another query: `<-#p(X)`. We have just shown that there is no proof of the query `<-#p(b)`. Then the answer to the query at any world is the ground atom `#p(a)`. Therefore it is possible that any implementation can only construct exactly *one* and the same minimal model, i.e., the one that assigns $\mathcal{U}$ to all of `p(a)`, `p(b)`, and `#p(a)`; and $\emptyset$ to `#p(b)`. There are surely many minimal models of the program, but only one of them is *constructible* in this sense. In the following, we will formalise the notion of a constructible minimal model by modifying and extending the fixpoint semantics of intensional logic programs.

6.4.1 The Mappings $NT_{\mathcal{P}}$ and $C_{\mathcal{P}}$

We now define the mapping $NT_{\mathcal{P}}$ as the one step modus ponens operation for program clauses in an extended program. It is an extension of $T_{\mathcal{P}}$ for intensional logic programs given in Chapter 5. New $T_{\mathcal{P}}$ operation does not affect the denotations of choice predicates. Let $\mathcal{F}(\mathcal{P})$ denote the family of intensional Herbrand interpretations of $\mathcal{P}$ and $T_{\mathcal{P}}$ be the mapping given in the definition 5.4.

Definition 6.4 *Let $\mathcal{P}$ be an extended intensional logic program and H an intensional Herbrand interpretation of $\mathcal{P}$. Then the mapping $NT_{\mathcal{P}} \in [\mathcal{F}(\mathcal{P}) \rightarrow \mathcal{F}(\mathcal{P})]$ where $NT_{\mathcal{P}}(H)$ satisfies the conditions*

$$\|\#p(\vec{e})\|^{NT_{\mathcal{P}}(H)} = \|\#p(\vec{e})\|^{H} \quad \textit{for all} \quad \#p(\vec{e}) \in B_{\mathcal{P}} \textit{ and}$$
$$\|p(\vec{e})\|^{NT_{\mathcal{P}}(H)} = \|p(\vec{e})\|^{T_{\mathcal{P}}(H)} \quad \textit{for all} \quad p(\vec{e}) \in B_{\mathcal{P}}$$

By definition, $NT_{\mathcal{P}}$ leaves the denotations of choice predicates intact. $NT_{\mathcal{P}}$ shares most of the properties of $T_{\mathcal{P}}$ for ordinary intensional logic programs; for example, it can be shown that $NT_{\mathcal{P}}$ is continuous. But lemma 5.10 reads differently: I is a model of intensional program clauses (not necessarily a model of $\mathcal{P}$) iff $NT_{\mathcal{P}}(I) \sqsubseteq I$.

Since $NT_{\mathcal{P}}$ does not cover the choice formulas of $\mathcal{P}$, we will define another mapping $C_{\mathcal{P}}$ as follows. Let H be an intensional Herbrand interpretation of $\mathcal{P}$. For all $p \in Pred$ and for all $w \in \mathcal{U}$, define $E_{p,w}(H)$ and $E_{\#_{p,w}}(H)$ as:

- $E_{p,w}(H) = \{\vec{e} \mid w \in \|p(\vec{e})\|^{H}\}$
- $E_{\#_{p,w}}(H) = \{\vec{e} \mid w \in \|\#p(\vec{e})\|^{H}\}$

In other words, $E_{p,w}(H)$ is the set of ground terms which p is true of at world w; and similarly $E_{\#_{p,w}}(H)$ is the set of ground terms which $\#p$ is true of at world w. Let $S = \{< p, w > \mid E_{p,w}(H) \neq \emptyset \textit{ and } E_{\#_{p,w}}(H) = \emptyset\}$, i.e., for any $< p, w > \in S$, no choice has been made for p at world w. Let $Cset(H) = \prod_{<p,w> \in S} E_{p,w}(H)$. Any

member of $Cset(H)$ represents a choice function, which, given any $< p, w > \in S$, returns an arbitrary term chosen from $E_{p,w}$. We now give the formal definition of $C_{\mathcal{P}}$ operation.

Definition 6.5 *Let $\mathcal{P}$ be an extended intensional logic program and H an intensional Herbrand interpretation of $\mathcal{P}$. Then the mapping $C_{\mathcal{P}} \in [\mathcal{F}(\mathcal{P}) \rightarrow P(\mathcal{F}(\mathcal{P}))]$ where, for all $\alpha \in Cset(H)$, $H_\alpha \in C_{\mathcal{P}}(H)$ iff for all $p \in Pred$, and for all $\vec{e} \in (U_{\mathcal{P}})^n$,*

$$\|p(\vec{e})\|^{H_\alpha} = \|p(\vec{e})\|^{H} \quad and$$
$$\|\#p(\vec{e})\|^{H_\alpha} = \|\#p(\vec{e})\|^{H} \cup \{w \mid \vec{e} = \alpha(p, w)\}.$$

The $C_{\mathcal{P}}$ operation returns all possible immediate extensions of a given intensional Herbrand interpretation of $\mathcal{P}$, determined by arbitrary choice functions. The interpretations in $C_{\mathcal{P}}(H)$ for any given H are indexed by α's in $Cset(H)$.

The ordinal powers of $NT_{\mathcal{P}}$ with respect to any intensional Herbrand interpretation I of an extended program $\mathcal{P}$ is defined as follows: $NT_{\mathcal{P}} \uparrow \omega\ (I) =_{def} \sqcup_{n \in \omega} NT_{\mathcal{P}} \uparrow n\,(I)$. Recall that $\mathcal{F}(\mathcal{P})$ denotes the family of intensional Herbrand interpretations of $\mathcal{P}$. Let $I_\emptyset$ denote $\sqcap\mathcal{F}(\mathcal{P})$, which is the greatest lower bound in the family. Therefore $I_\emptyset$ assigns $\emptyset$ to all ground intensional atoms in $B_{\mathcal{P}}$. In other words, for all predicate symbols $p \in Pred$ and for all $\vec{e} \in (U_{\mathcal{P}})^n$,

$$\|p(\vec{e})\|^{I_\emptyset} = \|\#p(\vec{e})\|^{I_\emptyset} = \emptyset \text{ and } \|\doteq\|^{I_\emptyset} = \{< e, e > \mid e \in U_{\mathcal{P}}\}$$

We will show that $I_\emptyset$ is a model of choice formulas of $\mathcal{P}$.

Lemma 6.6 *Let $\mathcal{P}$ be an extended intensional logic program. Then $I_\emptyset$ is a model of choice formulas $\{\Psi_p \mid p \in Pred\}$.*

Proof. $I_\emptyset$ trivially satisfies all of C1, C2 and C3 for all predicate symbols in *Pred* and hence is a model of choice formulas of $\mathcal{P}$ by lemma 6.1. $\dashv$

6.4.2 Alternating Chains of Models

The mappings $NT_{\mathcal{P}}$ and $C_{\mathcal{P}}$ interact nicely: if we start form $I_\emptyset$ and alternatively apply $NT_{\mathcal{P}} \uparrow \omega$ and $C_{\mathcal{P}}$, we obtain an "alternating chain" of interpretations of $\mathcal{P}$. In such an alternating chain, we go, in a zig-zag pattern, from a model of program clauses to a model of choice formulas and vice versa. There are many such alternating chains because of the arbitrary choices made by $C_{\mathcal{P}}$ operations. The method of alternating chains is well-known in the theory of models, e.g., see [Add65] and [CK73].

In the following, we will show that the limits (upper bounds) of these alternating chains are in fact the constructible minimal models of $\mathcal{P}$. In so doing, we will use an important result from the theory of models known as Chang-Łoś-Suszko theorem. The theorem applies to a class of formulas, called universal-existential formulas ($\forall\exists$-formulas) described below.

Definition 6.6 *A formula is universal-existential if it has the form*

$$(\forall X_0)\ldots(\forall X_{m-1})(\exists Y_0)\ldots(\exists Y_{n-1})A$$

where A is quantifier-free.

Every program clause in an intensional logic program is universal and thus it is also universal-existential. It can be shown that the set of choice formulas of any given extended program is equivalent to a set of $\forall\exists$-formulas as well. Therefore, the following theorem can be applied to extended programs.

Theorem 6.7 (Chang-Łoś-Suszko.) [CK73] *Let Γ be a set of formulas. Then Γ is equivalent to a set of $\forall\exists$-formulas iff the least upper bound of any upwards chain of models of Γ is a model of Γ.*

We will now formalise the notion of alternating chains of interpretations of an extended program $\mathcal{P}$ originating from $I_\emptyset$ by the following.

Definition 6.7 *Let $\mathcal{P}$ be an extended intensional logic program. Then $\mathcal{R}_\omega$ is the binary relation between intensional Herbrand interpretations of $\mathcal{P}$ defined below inductively.*

- $\mathcal{R}_0 = \{< I_\emptyset, NT_\mathcal{P} \uparrow \omega\,(I_\emptyset) >\}$
- $\mathcal{R}_{2n} = \{< I, NT_\mathcal{P} \uparrow \omega\,(I) > \mid \textit{for some } M, < M, I > \in \mathcal{R}_{2n-1}\},\ n > 0$
- $\mathcal{R}_{2n+1} = \{< I, I_\alpha \in C_\mathcal{P}(I) > \mid \textit{for some } M, < M, I > \in \mathcal{R}_{2n}\},\ n \geq 0$
- $\mathcal{R}_\omega = \bigcup_{n\in\omega} \mathcal{R}_n$

In fact, all intensional Herbrand interpretations of $\mathcal{P}$ in the domain of $\mathcal{R}_\omega$, denoted by $|\mathcal{R}_\omega|$, constitute a partially ordered set denoted by $(|\mathcal{R}_\omega|, \sqsubseteq)$ whose smallest element is $I_\emptyset$. There are alternating (upwards) ω-chains in $(|\mathcal{R}_\omega|, \sqsubseteq)$, all of which start from $I_\emptyset$. Let $\mathcal{C} =< I_\alpha >_{\alpha\in S}$ be an ω-chain as such. Observe that for any adjacent pair of interpretations (I_α, I_β) in $\mathcal{C}$, either $I_\beta = NT_\mathcal{P} \uparrow \omega\,(I_\alpha)$ or $I_\beta \in C_\mathcal{P}(I_\alpha)$.

The following lemma shows that, when applied in an alternating ω-chain of interpretations starting from $I_\emptyset$, $C_\mathcal{P}$ operation always returns a family of models of the choice formulas.

Lemma 6.8 *Let $\mathcal{P}$ be an extended intensional logic program and $< I_\alpha, I_\beta > \in \mathcal{R}_\omega$ where $I_\beta \in C_\mathcal{P}(I_\alpha)$. Let Ψ be the set of choice formulas for $\mathcal{P}$. Then $\models_{I_\beta} \Psi$.*

Proof. We will outline the proof. Since $< I_\alpha, I_\beta > \in \mathcal{R}_\omega$ and there both I_α and I_β are in a chain of interpretations of $\mathcal{P}$ starting from $I_\emptyset$, only $C_\mathcal{P}$ operations can introduce any changes into the denotations of choice predicates. We also know by lemma 6.6 that $I_\emptyset$ is a model of choice formulas of $\mathcal{P}$; thus the definition of $C_\mathcal{P}$ guarantees that I_β satisfies all of C1, C2 and C3, because no arbitrary terms are introduced in the denotations of choice predicates. Then $\models_{I_\beta} \Psi$ by lemma 6.1. $\dashv$

The following lemma states that, when applied in an alternating ω-chain of interpretations starting from $I_\emptyset$, the upward closure of $NT_\mathcal{P}$ operation is always a model of the program clauses in $\mathcal{P}$.

Lemma 6.9 *Let $\mathcal{P}$ be an extended intensional logic program, and $\mathcal{P}_H$ the set of intensional Horn clauses in $\mathcal{P}$. Let $< I_\alpha, NT_\mathcal{P} \uparrow \omega(I_\alpha) > \in \mathcal{R}_\omega$. Then $\models_{NT_\mathcal{P}\uparrow\omega(I_\alpha)} \mathcal{P}_H$.*

Proof. We will outline the proof. For all $w \in \mathcal{U}$ and $p(\vec{e}) \in B_\mathcal{P}$, that $w \in \|p(\vec{e})\|^{I_\alpha}$ is implied by a ground instance of some clause in $\mathcal{P}$, because $< I_\alpha, NT_\mathcal{P} \uparrow \omega(I_\alpha) >$ is in an ω-chain $\mathcal{C}$ starting from $I_\emptyset$ and each element in the chain is obtained from the previous one by either a $NT_\mathcal{P} \uparrow \omega$ operation or a $C_\mathcal{P}$ operation. As for choice predicates, they do not appear in the heads of intensional Horn clauses, therefore their denotations are solely determined by $C_\mathcal{P}$ operations. Then it can be shown that $< NT_\mathcal{P} \uparrow n(I_\alpha) >_{n\in\omega}$ is also a chain whose least upper bound is $NT_\mathcal{P} \uparrow \omega(I_\alpha)$. Therefore $NT_\mathcal{P} \uparrow \omega(I_\alpha)$ is a model of $\mathcal{P}_H$ by the continuity of $NT_\mathcal{P}$. $\dashv$

Lemmas 6.8 and 6.9 fail when we consider arbitrary chains of intensional Herbrand interpretations of $\mathcal{P}$. Any ω-chain $\mathcal{C}$ in $|\mathcal{R}_\omega|$ starts from $I_\emptyset$ and each element in the chain is obtained from the previous one by either $NT_\mathcal{P} \uparrow \omega$ or $C_\mathcal{P}$ operation. Therefore Lemmas 6.8 and 6.9 ensure that $\mathcal{C}$ is an alternating chain of interpretations of $\mathcal{P}$ whose elements are either models of intensional program clauses (those obtained by $NT_\mathcal{P} \uparrow \omega$ operations) or models of choice formulas (those obtained by $C_\mathcal{P}$ operations).

We will now show that the least upper bounds of ω-chains in the domain of $\mathcal{R}_\omega$ are models of extended programs. In other words, if we start from $I_\emptyset$, consecutive applications of $NT_\mathcal{P} \uparrow \omega$ and $C_\mathcal{P}$ will eventually produce a model of $\mathcal{P}$, in fact, a minimal one. These minimal models are the ones constructible from the program.

Theorem 6.10 *Let $\mathcal{P}$ be an extended intensional logic program; $\mathcal{C}$ be an ω-chain in $(|\mathcal{R}_\omega|, \sqsubseteq)$ and $\sqcup\mathcal{C} = \sqcup_{\alpha\in S} I_\alpha$ be the least upper bound of $\mathcal{C}$. Then $\models_{\sqcup\mathcal{C}} \mathcal{P}$.*

Proof. By construction, $\sqcup\mathcal{C}$ is the common least upper bound of two interleaving chains in $\mathcal{C}$, namely, $< C_{2n} \in \mathcal{C} | n \in \omega >$ and $< C_{2n+1} \in \mathcal{C} | n \in \omega >$. In other words, we have that $\sqcup\mathcal{C} = \sqcup_{n\in\omega} C_{2n} = \sqcup_{n\in\omega} C_{2n+1}$. This implies that, by Chang-Łoś-Suszko theorem (theorem 6.7), $\sqcup\mathcal{C}$ is a model of both program clauses and choice formulas. Hence we conclude that $\models_{\sqcup\mathcal{C}} \mathcal{P}$. ⊣

The following corollary strengthens theorem 6.10 by showing that, when we start from $I_\emptyset$, alternative applications of $NT_\mathcal{P} \uparrow \omega$ and $C_\mathcal{P}$ lead to a constructible minimal intensional Herbrand model.

Corollary 6.11 *Let $\mathcal{P}$ be an extended intensional logic program, $\mathcal{C}$ an ω-chain in $(|\mathcal{R}_\omega|, \sqsubseteq)$ and $\sqcup\mathcal{C} = \sqcup_{\alpha\in S} I_\alpha$ be the least upper bound of $\mathcal{C}$. Then $\sqcup\mathcal{C}$ is minimal.*

Proof. By theorem 6.10, $\sqcup\mathcal{C}$ is a model of $\mathcal{P}$. Suppose it is not minimal. Since the minimality condition concerns ony non-choice predicates, we will consider only those ground atoms related to non-choice predicates. In other words, for any $w \in \mathcal{U}$ and any $p(\vec{e}) \in B_\mathcal{P}$, if $w \in \|\#p(\vec{e})\|^{\sqcup\mathcal{C}}$ then $w \in \|p(\vec{e})\|^{\sqcup\mathcal{C}}$ for $\sqcup\mathcal{C}$ is a model of $\mathcal{P}$. Suppose that $w \in \|p(\vec{e})\|^{\sqcup\mathcal{C}}$, but this is not implied by any ground instance of any clause in $\mathcal{P}$. Then it must be the case that for some $I_\alpha \in \mathcal{C}$, $w \in \|p(\vec{e})\|^{I_\alpha}$. But for some I_β, we must have that $I_\alpha \sqsubseteq I_\beta \sqsubseteq NT_\mathcal{P} \uparrow \omega(I_\beta)$. This entails that $w \in \|p(\vec{e})\|^{I_\beta}$ but $w \notin \|p(\vec{e})\|^{NT_\mathcal{P}\uparrow\omega(I_\beta)}$, because $NT_\mathcal{P}$ operation does not keep w in the intension of $p(\vec{e})$ if it is not implied by any program clause. This leads to the contradiction that $\mathcal{C}$ cannot be a chain. Thus $\sqcup\mathcal{C}$ must be minimal. ⊣

Some extended programs, such as the one given in the beginning of this section, are deterministic in the sense that they operationally single out one minimal model. Then any implementation can construct only that minimal model.

Definition 6.8 *Let $\mathcal{P}$ be an extended intensional logic program. We say that $\mathcal{P}$ is* operationally deterministic *iff* $\mathtt{card}(M) = 1$ *where $M = \{ \sqcup\mathcal{C} \mid \mathcal{C}$ is an ω-chain in $(|\mathcal{R}_\omega|, \sqsubseteq)\}$.*

The properties of operationally deterministic programs are worth investigating in the future. But we still can not recover the notion of logical consequence, since it is basically model-theoretical. But we may consider that minimal model as the intended (canonical) meaning of an extended intensional logic program.

6.5 A Comparison With Committed-Choices

At the first glance, the idea of choice predicates seems to be a variant of committed-choices introduced in concurrent (infinitary) logic programming languages such as Parlog [CG86], Concurrent Prolog [Sha87], and GHC [Ued86], but it is not so. Up to now, proposed approaches to declarative semantics of concurrent logic programming languages, such as the greatest fixpoint semantics [vENA84] [Llo84] [NA85] [Mur88] etc., employ infinite data structures such as streams; these approaches fail to capture the idea of committed-choices, and furthermore do not characterise the set of logical consequences of a given logic program. Besides, we claim that the pure declarative reading of a concurrent logic program is not necessarily what the programmer intends to specify.

For instance, suppose we have only two alternative clauses for predicate `use` in the following segment of a concurrent logic program, which says that some non-shared resource `r` is used by either process `a` or process `b`. This is a mutual exclusion problem.

```
...
use(a,r) <- ... | ...
use(b,r) <- ... | ...
```

This is fine, since the programmer *knows* that given a query like "`<- use(P,r)`" any implementation will only commit to at most one of these two clauses and produce the corresponding ground instance of the query. On the other hand, with respect to the

greatest fixpoint semantics of the program, both of the ground instances `use(a,r)` and `use(b,r)` may be true. Furthermore, since committed-choice non-determinism can not guarantee the uniqueness of the ground term which the **use** predicate is true of, a query like "`<- use(P1,r),use(P2,r)`" may result in ground instances like " `use(a,r),use(b,r)`" which is certainly not what the programmer has in mind.

Intensional logic programming offers a logical alternative to infinitary versions of concurrent logic programming and has well-defined minimum model semantics. Intensional logic programming with choice predicates actually solves the mutual exclusion problem faced in the above program. We just use, in the queries, choice predicates provided for each predicate symbol. Then at any given world, the answer to the query "`<- #use(P,r)`" is exclusively either `#use(a,r)` or `#use(b,r)`. Similarly, the answer to the query "`<- #use(P1,r),#use(P2,r)`" in any implementation is either the ground instance " `#use(a,r),#use(a,r)`" or the ground instance " `#use(b,r),#use(b,r)`". This is also reflected in the minimal model semantics. In summary, choice predicates represent what is actual whereas non-choice predicates specify what is possible.

Hewitt and Agha [HA88] argue that proof theory does not provide a good model for the semantics of committed-choice languages due to the arrival-order assumptions of the messages, not implied by the logical reading of concurrent logic programs. Intensional logic programs with choice predicates can alleviate this problem, because semantics of such programs are defined in terms of minimal models where each choice predicate represents a non-deterministic stream in any minimal model of the program. Then it remains to show that proof theoretical semantics of intensional logic programs is equivalent to constructing one such minimal model, in fact, a constructible one. But the proof theory of logic programming does not directly apply to ILP languages with choice predicates and thus more work has to be done.

Chapter 7

Defining Intensional Operators

This chapter explores how the expresiveness of intensional logic programming can be improved from within. Suppose we have an intensional logic and we would like to extend it with an extra intensional operator. This can be done by using the machinery provided by intensional semantics, i.e., by extending the definition of the satisfaction relation $\models$ accordingly. However, within the context of intensional logic programming, this approach may not be always preferable. Sometimes, we would like to have additional intensional operators without resorting to enriching the underlying intensional logic, especially when they can be made available in terms of (monotonic) formulas interpreted as the definitions of intensional operators.

We will show that intensional Horn clauses can be used to define additional intensional operators and an intensional logic program with operator definitions can be transformed into an "equivalent" program in which only intensional operators of the underlying logic appear. Then the results from the previous chapters apply to the transformed program, and the correctness of the transformation procedure can be established. When recursion is allowed in defining intensional operators, we add more expressive power to intensional logic programming. The fixpoint techniques can be employed to solve recursive definitions, but we can no longer stay within first-

order intensional logic. We must employ an infinitary logic such as $L_{\omega_1\omega}$ [Kei71].

7.1 Non-Recursive Definitions

Suppose that we would like to extend the temporal logic with a temporal operator with the following reading: "now and during the next n-1 moments in time". One can notice that the same temporal operator can be expressed as $\bigwedge_{i\in n} \mathbf{next}^i A$ for a given formula A. If we still want to remain in the underlying temporal logic, this formula can be used as the definition of the new temporal operator, say $[n]$, as follows: $[n]A =_{def} \bigwedge_{i\in n} \mathbf{next}^i A$. Another alternative is to enrich the underlying logic directly with a new symbol $[n]$ by extending the definition of the satisfaction relation $\models$ accordingly, in which case we move to a *definitional extension* of the logic [Seg82]. Both approaches are fine, but, in temporal logic programming, we can define $[n]$ on the fly by a temporal program clause, and invoke this clause whenever a formula of the form $[n]B$ needs to be proved.

Since the temporal operator $[n]$ is not directly available in the underlying logic, the defining formula of $[n]$ given above can be used to express the temporal operator in the form of a temporal program clause as

$$[n]A \leftarrow \bigwedge_{i\in n} \mathbf{next}^i A$$

where A is just a propositional variable. Let $\mathcal{P}$ be a temporal logic program in which $[n]$ is applied to some temporal atom B in the body of some clause. To prove $[n]B$, the definition can be used after substituting A by B.

$$([n]A \leftarrow \bigwedge_{i\in n} \mathbf{next}^i A)\{A/B\} = [n]B \leftarrow \bigwedge_{i\in n} \mathbf{next}^i B$$

Here the occurrence of $[n]$ is not a direct application of $[n]$ to B; it is there for definitional purposes — establishing the connection between the use and the definition of $[n]$. In other words, the defining formula of $[n]$ works as a meta-rule.

Now we can use the right-hand side of the clause to prove $[n]B$. This is how an implementation should proceed.

The next step is to define the semantics of intensional operator definitions. The theorem 5.12 does not extend to intensional logic programs with meta-rules, although the meta-rule invocation rule seems to be correct. The problem is that we cannot directly incorporate the definition of $[n]$ in an intensional logic program, nor can we use $[n]$ in the program, because, in the object language, the meaning of the symbol $[n]$ is unknown. Furthermore, it is not clear whether the temporal clause is part of the definition of $[n]$ or of A. Therefore we will treat program clauses and meta-rules differently, and stipulate that meta-rules define additional operators which can be used only in the bodies of program clauses.

Since we regard $[n]$ as an intensional operator, its denotation $\|[n]\|$ must be an element of $[P(\omega) \to P(\omega)]$. We can formulate a straightforward model-theoretical condition for $\|[n]\|$ by the following: for all $X \in P(\omega)$,

$$\|[n]\|(X) \supseteq \bigcap_{i \in n} \|\mathbf{next}\|^i(X)$$

All the functions in $[P(\omega) \to P(\omega)]$ which satisfy this model-theoretical condition are candidates for $\|[n]\|$. However, we must choose the least function among all the candidates as the canonical meaning of the definition of $[n]$. Here $\lambda X. \bigcap_{i \in n} \|\mathbf{next}\|^i(X)$ is the least function that corresponds to the monotonic formula at the right-hand side of the definition of $[n]$. In the following, we will formalize this notion.

In general, a new intensional operator can be defined by a set of intensional program clauses whose heads are identical. Below is the formal definition of intensional program clauses as the definitions of new intensional operators.

Definition 7.1 *A new n-ary intensional operator $\bigtriangledown$ can be defined by a set of intensional program clauses as*

$$\{ \bigtriangledown(A_0, \ldots, A_{n-1}) \leftarrow \bigwedge_{k \in m_\alpha} B_{\alpha,k} \mid \alpha \in S \}$$

where $S \in \omega$, each $m_\alpha \in \omega$, all A_i's are propositional variables, and each $B_{\alpha,k}$ is an intensional unit constructed out of A_i's and already available intensional operators.

There is one essential restriction: the symbol ∇ can not appear in the body of any clause in the definition of ∇ or in the definitions of other intensional operators. Since defined intensional operators can be used only in the bodies of intensional program clauses, we require that the denotations of all intensional operators which may be used in defining new operators be monotonic and finitary. This will ensure that the denotations of defined operators are monotonic and finitary as well.

From here on, we assume new intensional operators are defined by a single clause, since the above set of clauses is equivalent to the clause given below by rules of propositional calculus.

$$\nabla(A_0, \ldots, A_{n-1}) \leftarrow \bigvee_{\alpha \in S} \bigwedge_{k \in m_\alpha} B_{\alpha,k}$$

This definition is interpreted as follows: $\nabla(A_0, \ldots, A_{n-1})$ is true at any given world $w \in \mathcal{U}$ when for some $\alpha \in S$, all $B_{\alpha,k}$'s are true at w.

7.1.1 Semantics of Operator Definitions

If we define a new intensional operator such as $[n]$ by using a monotonic formula of the underlying logic, its definition would be a two-way street. Moreover, the denotation of a defined intensional operator can be obtained from the denotations of those used in the definition. But, in intensional Horn logic, we have a weaker definition, with just an implication. In the following, we will elaborate what the definition really means. First, we formulate a model-theoretical condition for intensional operator definitions.

Definition 7.2 *Let ∇ be an n-ary intensional operator defined by an intensional program clause as $\nabla(A_0, \ldots, A_{n-1}) \leftarrow \bigvee_{\alpha \in S} \bigwedge_{k \in m_\alpha} B_{\alpha,k}$. We define Fun_∇ to be*

the family of functions

$$\{\, \Psi \in [P(\mathcal{U})^n \to P(\mathcal{U})] \mid (\forall \vec{X} \in P(\mathcal{U})^n)\ \Psi(\vec{X}) \supseteq \bigcup_{\alpha \in S} \bigcap_{k \in m_\alpha} \|\nabla_{\alpha,k}\|(\vec{X}) \,\}$$

where each intensional unit $B_{\alpha,k}$ is interpreted as an application of some n-ary intensional operator $\nabla_{\alpha,k}$ to A_i's.

We will now investigate the properties of Fun_∇. For any given n-ary functions Ψ and Θ, we define $\Psi \sqcap \Theta$ to be the function $\lambda \vec{X}.\Psi(\vec{X}) \cap \Theta(\vec{X})$ and $\Psi \sqcup \Theta$ to be the function $\lambda \vec{X}.\Psi(\vec{X}) \cup \Theta(\vec{X})$. These notions naturally extend to a family of functions. In fact, $\sqcap$ and $\sqcup$ are the greatest lower bound and the least upper bound operations on a given family of functions with the same arity.

We will now show that Fun_∇ is closed under $\sqcap$. Then $\sqcap Fun_\nabla$ should be taken as the canonical function implied by the definition of ∇, because it is the least function satisfying the model-theoretical condition imposed on Fun_∇. This is in keeping with the spirit of the minimum model semantics.

Lemma 7.1 *Let ∇ be an n-ary intensional operator defined by an intensional program clause as: $\nabla(A_0, \ldots, A_{n-1}) \leftarrow \bigvee_{\alpha \in S} \bigwedge_{k \in m_\alpha} B_{\alpha,k}$; then Fun_∇ is closed under $\sqcap$.*

Proof. Let *Sfun* be any subset of Fun_∇. Then we have that for all $\vec{X} \in P(\mathcal{U})^n$,

$$\begin{aligned}
(\forall \Psi \in Sfun)\ \Psi(\vec{X}) &\supseteq \bigcup_{\alpha \in S} \bigcap_{k \in m_\alpha} \|\nabla_{\alpha,k}\|(\vec{X}) \Longrightarrow \\
\bigcap_{\Psi \in Sfun} \Psi(\vec{X}) &\supseteq \bigcup_{\alpha \in S} \bigcap_{k \in m_\alpha} \|\nabla_{\alpha,k}\|(\vec{X}) \Longrightarrow \\
(\sqcap_{\Psi \in Sfun} \Psi)(\vec{X}) &\supseteq \bigcup_{\alpha \in S} \bigcap_{k \in m_\alpha} \|\nabla_{\alpha,k}\|(\vec{X})
\end{aligned}$$

which implies $(\sqcap_{\Psi \in Sfun} \Psi) \in Fun_\nabla$. $\dashv$

The following lemma shows that we have the stronger definition after all. In other words, the right-hand side of a given intensional operator definition can be regarded as the defining formula of the intensional operator in question. Therefore the meta-rule invocation rule is correct.

Lemma 7.2 *Let ∇ be an n-ary intensional operator defined by an intensional program clause as: $\nabla(A_0, \ldots, A_{n-1}) \leftarrow \bigvee_{\alpha \in S} \bigwedge_{k \in m_\alpha} B_{\alpha,k}$; then*

$$\sqcap Fun_\nabla = \lambda \vec{X}. \cup_{\alpha \in S} \cap_{k \in m_\alpha} \| \nabla_{\alpha,k} \| (\vec{X}).$$

Proof. Since $\sqcap Fun_\nabla$ is the least function in Fun_∇ by lemma 7.1, for any $\Psi \in Fun_\nabla$ and for any $\vec{X} \in P(\mathcal{U})^n$, $\sqcap Fun_\nabla(\vec{X}) \subseteq \Psi(\vec{X})$. But we also have that $\lambda \vec{X}. \cup_{\alpha \in S} \cap_{k \in m_\alpha} \| \nabla_{\alpha,k} \| (\vec{X}) \in Fun_\nabla$, because it satisfies the model-theoretical condition. Then for any $\vec{Z} \in P(\mathcal{U})^n$, $\sqcap Fun_\nabla(\vec{Z}) \subseteq \cup_{\alpha \in S} \cap_{k \in m_\alpha} \| \nabla_{\alpha,k} \| (\vec{Z})$. We also have that $\sqcap Fun_\nabla$ satisfies the model-theoretical condition, which implies that for any $\vec{Z} \in P(\mathcal{U})^n$, $\sqcap Fun_\nabla(\vec{Z}) \supseteq \cup_{\alpha \in S} \cap_{k \in m_\alpha} \| \nabla_{\alpha,k} \| (\vec{Z})$. Therefore $\sqcap Fun_\nabla = \lambda \vec{X}. \cup_{\alpha \in S} \cap_{k \in m_\alpha} \| \nabla_{\alpha,k} \| (\vec{X})$. $\dashv$

7.1.2 From Meta-Theories to Intensional Logic Programs

An intensional logic program $\mathcal{P}$ with intensional operator definitions is a meta-theory of the underlying intensional Horn logic. The results we have just given are encouraging, but yet fall short of extending the theory of intensional logic programming to meta-theories. The problem is that there are still uninterpreted constructs in $\mathcal{P}$ which are foreign to the underlying first-order language, even though, intuitively, we know what they mean. In this section, we will devise a transformation procedure which starts from a meta-theory and produces an intensional logic program in the object-language that captures the intended meaning of each defined intensional operator, that is, the least function implied by the definition of the operator.

As a meta-theory, $\mathcal{P}$ consists of two parts: meta-rules and ordinary program clauses in which new defined operators are used in the bodies. In order to obtain a program of the underlying logic, all applications of defined operators can be replaced by their corresponding defining formulas after proper substitutions. We first define a syntactic translation function τ as follows.

Definition 7.3 *Let τ be a syntactic translation function from formulas of a meta-theory to formulas of IL defined as follows:*

- $\tau(\bigtriangledown(B_0,\ldots,B_{n-1})) = \tau(\bigvee_{\alpha\in S}\bigwedge_{k\in m_\alpha} B_{\alpha,k}\{A_0/B_0,\ldots,A_{n-1}/B_{n-1}\})$ *where $\bigtriangledown$ is an intensional operator defined by the program clause (meta-rule) as* $(\bigtriangledown(A_0,\ldots,A_{n-1}) \leftarrow \bigvee_{\alpha\in S}\bigwedge_{k\in m_\alpha} B_{\alpha,k}) \in \mathcal{P}$.
- $\tau(\bigtriangledown(B_0,\ldots,B_{n-1})) = \bigtriangledown(\tau(B_0),\ldots,\tau(B_{n-1}))$ *where $\bigtriangledown$'s an intensional operator of the underlying logic.*
- $\tau(\bigwedge_{\alpha\in S} B_\alpha) = \bigwedge_{\alpha\in S}\tau(B_\alpha)$ *(similarly for $\bigvee$).*
- $\tau(B) = B$ *where B is an atomic formula.*

Let $\mathcal{P}_c$ denote the set of intensional program clauses in $\mathcal{P}$. Then the transformed program $\mathcal{P}^\tau$ is defined as follows:

$$\mathcal{P}^\tau = \{(A \leftarrow \tau(B_0),\ldots,\tau(B_{n-1})) \mid (A \leftarrow B_0,\ldots,B_{n-1}) \in \mathcal{P}_c\}$$

All the model-theoretical results from the previous sections apply to $\mathcal{P}^\tau$, since the pre-conditions of the theorem 5.12 are met. Given $\mathcal{P}$, we regard $\mathcal{P}^\tau$ as the canonical program associated with $\mathcal{P}$, which is, in a sense, equivalent to $\mathcal{P}$. As an alternative, we can push the definitions of intensional operators to the underlying logic, and leave the applications of those operators in intensional programs intact.

7.2 Recursively Defined Intensional Operators

Consider the temporal logic of Chronolog and call the underlying temporal language *TL*. We will now enrich *TL* with two new temporal operators, $\Box$ and $\Diamond$ as follows. Read $\Box$ as "always" and $\Diamond$ as "sometime". A formula of the form $\Box A$ is true at time t in a temporal interpretation I just in case A is true at all moments in time; a

formula of the form $\Diamond A$ is true at time t in I just in case A is true at some moment in time. In other words, $\Box$ and $\Diamond$ are just like (temporal) necessity and possibility operators. Then we find that the following theorems hold in the extended temporal logic.

- $\models \Box A \leftrightarrow \texttt{first}\ A \wedge \Box(\texttt{next}\ A)$
- $\models \Diamond A \leftrightarrow \texttt{first}\ A \vee \Diamond(\texttt{next}\ A)$

If we choose to remain in TL and insist on having $\Box$ and $\Diamond$, the theorems given above can be used to obtain recursive definitions of $\Box$ and $\Diamond$. However, it is not hard to see that such definitions actually lead to infinitary formulas in the object language (formulas with countable conjunctions and disjunctions.) For instance, by repeatedly unfolding the right-hand side of the definition of $\Box$, we obtain the following non-recursive definition of $\Box$ by an infinitary formula.

$$\Box A =_{def} \texttt{first}\ A \wedge \texttt{first next}\ A \wedge \texttt{first next next}\ A \wedge \ldots$$

Or in a more succint notation:

$$\Box A =_{def} \bigwedge_{n \in \omega} \texttt{first next}^n\ A.$$

Similarly, we obtain the non-recursive definition of $\Diamond$ by an infinitary formula as

$$\Diamond A =_{def} \bigvee_{n \in \omega} \texttt{first next}^n\ A.$$

Now the problem is that TL does not allow for infinitary formulas. Therefore we must adopt an infinitary logic such as $L_{\omega_1\omega}$ [Kei71]. The logic of $L_{\omega_1\omega}$ allows countable conjunctions and disjunctions, but only a finite set of quantifiers in front of any formula.

If recursion is allowed in defining new intensional operators, both $\Box$ and $\Diamond$ can be defined from the above theorems as meta-rules in temporal Horn logic as

- $\Box A \leftarrow \texttt{first}\ A \wedge \Box(\texttt{next}\ A)$.
- $\Diamond A \leftarrow \texttt{first}\ A \vee \Diamond(\texttt{next}\ A)$.

where A is a propositional variable. To prove any formula of the form $\Diamond B$, the definition of $\Diamond$ can be invoked after substituting A by B, which in turn may require further invocations, should $\texttt{first}\ B$ be not proved, and so on.

As mentioned earlier, we consider the use of these new intensional operators only in the bodies of intensional program clauses. The methods described in the previous section are not directly applicable to meta-theories of intensional Horn logic. In fact, if the above definitions appear in any intensional logic program $\mathcal{P}$, the transformation procedure described above may never eliminate recursive applications of $\Box$ and $\Diamond$ in their definitions. We will employ the fixpoint techniques to overcome this problem.

7.2.1 Fixpoints of Recursive Definitions

We will now outline a fixpoint approach to recursive definitions in intensional logic programming based on $L_{\omega_1\omega}$. Note that the model-theory developed so far can be extended to infinitary intensional logic programs (programs with countably many program clauses) in a straightforward manner. Baudinet [Bau89] shows the connections between the fixpoint semantics of temporal logic programming and the temporal logic μTL of Vardi [Var88]. Barringer [Bar87] employs a temporal fixpoint calculus (a temporal logic with fixpoint constructors) for recursively defined temporal operators in the context of program specification. Guessarian [Gue88] applies a similar approach to solving recursive data-base queries.

A functional Υ is an element of $\bigcup_{n\in\omega}[[P(\mathcal{U})^n \rightarrow P(\mathcal{U})] \rightarrow [P(\mathcal{U})^n \rightarrow P(\mathcal{U})]]$. Continuity and monotonicity of functionals can be defined as usual, e.g., see [Man74]. We state the following theorem without proof.

Theorem 7.3 *Any functional* Υ*, when defined by compositions of monotonic and finitary functions and the function variable* Θ*, is continuous.*

Every continuous functional Υ has a least fixpoint denoted by $lfp(\Upsilon)$. In fact, by lemma 2.8, we have that $lfp(\Upsilon) = \Upsilon \uparrow \omega =_{def} \sqcup\{\Upsilon^n(\lambda\vec{X}.\emptyset) \mid n \in \omega\}$. Clearly $\lambda\vec{X}.\emptyset$ is the bottommost function in the complete lattice of functions in $[P(\mathcal{U})^n \rightarrow P(\mathcal{U})]$ for any given n. Similarly, $\lambda\vec{X}.\mathcal{U}$ is the topmost one. Associated with every recursive definition of an intensional operator is a continuous functional whose least fixpoint is the meaning of the definition.

The functions implied by recursive definitions given for $\Box$ and $\Diamond$ of *TL* are the least fixpoints of the following functionals $\Upsilon_\Box$ and $\Upsilon_\Diamond$, obtained from the definitions.

$$\Upsilon_\Box(\Theta) = \lambda X.\|\texttt{first}\|(X) \cap \Theta \circ \|\texttt{next}\|(X)$$

$$\Upsilon_\Diamond(\Theta) = \lambda X.\|\texttt{first}\|(X) \cup \Theta \circ \|\texttt{next}\|(X)$$

Since $\|\Box\|$ is not continuous to start with, we do not expect to obtain a discontinuous function from the least fixpoint of $\Upsilon_\Box$; indeed, it can be verified that $lfp(\Upsilon_\Box) = \lambda X.\emptyset$. Then, with respect to the least fixpoint approach, the recursive defining formula of $\Box$ does not capture the intended meaning of $\Box$ at all. (We will discuss the meaning of the definition of $\Box$ a little later.) As $\|\Diamond\|$ is continuous, the least fixpoint of $\Upsilon_\Diamond$ is exactly what we wanted, i.e., the least function implied by the definition of $\Diamond$.

$$lfp(\Upsilon_\Diamond) = \lambda X. \bigcup_{n\in\omega} \|\texttt{first}\| \circ \|\texttt{next}\|^n(X)$$

Here $\|\texttt{next}\|^n$ is n-fold composition of $\|\texttt{next}\|$.

We have started from the recursive definition of $\Diamond$ and described a method to obtain the least function implied by the definition. Now the definition of $\Diamond$ can be obtained from that function by just using the syntactic counterparts of continuous functions, compositions, intersections, and unions. For set variables, we introduce

propositional variables. Below is the non-recursive definition of $\Diamond$ obtained from $lfp(\Upsilon_\Diamond)$,

$$\Diamond A \leftarrow \bigvee_{n\in\omega} \mathtt{first}\,\mathtt{next}^n\, A$$

where A is a propositional variable. In short, the least fixpoint techniques allow us to define, through recursion, intensional operators whose denotations are continuous.

The greatest fixpoint of a continuous functional Υ also exists and is defined as $gfp(\Upsilon) = \Upsilon \downarrow \omega =_{def} \sqcap\{\Upsilon^n(\lambda\vec{X}.\mathcal{U}) \mid n \in \omega\}$. In the terminology of Barringer [Bar87], unary conjunctive operators such as $\Box$ are called $\wedge$-*continuous*. The recursive definition of a given discontinuous (non-finitary) but $\wedge$-continuous operator is amenable to the greatest fixpoint construction. Then it can be verified that the original definition of $\|\Box\|$ is actually the greatest fixpoint of the functional $\Upsilon_\Box$.

$$gfp(\Upsilon_\Box) = \lambda X. \bigcap_{n\in\omega} \|\mathtt{first}\| \circ \|\mathtt{next}\|^n(X)$$

If we adopt the greatest fixpoint techniques in intensional logic programming, the non-recursive definition of $\Box$ can be formed from the greatest fixpoint of $\Upsilon_\Box$ by the following.

$$\Box A \leftarrow \bigwedge_{n\in\omega} \mathtt{first}\,\mathtt{next}^n\, A$$

Note that the greatest fixpoint of $\Upsilon_\Diamond$ is $\lambda X.\mathcal{U}$. The meta-rule invocation rule in fact interprets the recursive definition correctly, and it approximates to the non-recursive definition of $\Box$ given above.

However, there are several fundamental problems with this approach. Since the definition of $\Box$ has an infinite conjunction and lemma 4.7 does not hold for logics based on $L_{\omega_1\omega}$, the mapping $T_\mathcal{P}$ (for the transformed program) is no longer continuous. In fact, to prove any formula of the form $\Box B$, we need to prove $\bigwedge_{n\in\omega} \mathtt{first}\,\mathtt{next}^n\, B$. But there may not be a finite proof for this conjunction. Therefore the definition of $\Box$ cannot be regarded as the definition of an intensional operator, even though it is a monotonic formula. This means that the greatest fixpoint construction can not be adapted to intensional logic programming, because

we consider the use of recursively defined intensional operators only in the bodies of intensional program clauses, and continuity is a requirement for such operators.

In case of mutually recursive definitions, the fixpoint theory of functionals can be extended to cover a set of recursive definitions in a usual way, e.g., see [Man74]. We will give an example of mutual recursion from temporal logic. Let <*even*> and <*odd*> be temporal operators with the following readings: "at some even moment in time" and "at some odd moment in time". In temporal Horn logic, we can define <*even*> and <*odd*> in terms of each other as follows.

- $\texttt{<}even\texttt{>}A \leftarrow \texttt{first}\, A \vee \texttt{<}odd\texttt{>}(\texttt{next}\, A)$.
- $\texttt{<}odd\texttt{>}A \leftarrow \texttt{<}even\texttt{>}(\texttt{next}\, A)$.

Other definitions of <*even*> and <*odd*> are of course possible.

The functional associated with these definitions, say Υ, will take a binary tuple of functions, one for <*even*> and one for <*odd*>, and return a binary tuple of functions. The least fixpoint of the functional Υ will be a binary tuple of the least functions for <*even*> and <*odd*>. In other words, $lfp(\Upsilon)$ is the binary tuple

$$< \lambda X.\, \cup_{n\in\omega} \|\texttt{first}\| \circ \|\texttt{next}\|^{2n}(X)\,,\ \lambda X.\, \cup_{n\in\omega} \|\texttt{first}\| \circ \|\texttt{next}\|^{2n+1}(X) >\,.$$

Now the non-recursive definitions of <*even*> and <*odd*> can be obtained from the least functions in the tuple. Then we can use the syntactic transformation procedure in the manner described above. It should be noted that the least fixpoint techniques subsume the semantics of non-recursive operator definitions described earlier. However, we do not explore this subject any further.

Let $\mathcal{P}$ be a meta-theory which contains recursively defined intensional operators. We first modify $\mathcal{P}$ by replacing the bodies of the definitions of recursively defined intensional operators by the non-recursive definitions obtained from the least fixpoints of the corresponding functionals, and retain all the other program clauses.

Let us refer to the modified $\mathcal{P}$ as $\mathcal{P}_f$ which is surely a meta-theory but it can now be subjected to the transformation procedure induced by the translation function τ. From $\mathcal{P}_f$, we can obtain an intensional logic program $(\mathcal{P}_f)^\tau$, free of symbols foreign to the underlying first-order intensional logic based on $L_{\omega_1\omega}$. Then the theorem 5.12 tells us that $(\mathcal{P}_f)^\tau$ has the minimum model property.

In summary, intensional operator definitions add more expressive power to intensional logic programming. The minimum model theory and the fixpoint theory of intensional logic programs can still be applied to programs with operator definitions – more precisely, to transformed programs in $L_{\omega_1\omega}$. This means that any implementation based on the notion of meta-rule invocation correctly interprets all the recursive/non-recursive intensional operator definitions in a given intensional logic program.

Chapter 8

Conclusions

This chapter summarizes the main results and contributions of this dissertation, pointing out some of the open problems and other issues where further research is profitable, and making some remarks on relevant work reported elsewhere.

8.1 A Summary of the Dissertation

In Chapter 3, we showed that temporal logic programs can model non-terminating computations and the notion of dynamic change naturally and without employing infinitary objects such as streams. In temporal logic programming, an infinite computation is initiated by a query which is not fixed to any particular moment in time. Such a query fires an infinite series of queries, each of which is fixed to a particular moment in time. Therefore an infinite computation may be regarded as an infinite series of independent finite computations for each moment in time; and the answers obtained from each independent computation are logical consequences of the program, which is not the case for logic programming with infinitary terms.

We established that temporal logic programs of Chronolog enjoy the minimum model semantics and its fixpoint characterisation based on the mapping $T_{\mathcal{P}}$. The

semantics of temporal logic programming is a natural extension of van Emden-Kowalski semantics [vEK76]. We also pointed out the similarities between the temporal languages Chronolog [Wad85] and Templog [AM87]. We outlined some rules of inference for temporal logic which could be used to devise a sound and complete proof procedure for Chronolog. Baudinet [Bau88] [Bau89] showed the completeness of Templog's proof procedure, called $TSLD$-resolution. With some modifications, $TSLD$-resolution can be used for Chronolog as well. However, the model-theory of temporal logic programming lacks the generality to extend to other intensional logic programming languages.

In Chapter 4, we outlined an intensional semantics for studying the mathematical properties of intensional logics. We departed from the traditional Kripke-style semantics for intensional logic, and adopted the more general algebraic semantics and neighborhood semantics of Scott [Sco70] and Montague [Mon74]. In intensional semantics, the denotation of an intensional operator ∇ is abstracted as an element of

$$\bigcup_{n\in\omega} [P(\mathcal{U})^n \to P(\mathcal{U})]$$

where $P(\mathcal{U})$ is the set of subsets of the universe of possible worlds. Scott-Montague semantics helped us to isolate important concepts and properties of intensional logic, and paved the way to a language-independent investigation of ILP. We defined several semantic constraints on intensional operators, including monotonicity, universality, conjunctivity and finitariness. Monotonic and finitary intensional operators are also called continuous. We also showed that each property has certain implications for the neighborhood semantics, and some properties are interrelated.

In Chapter 5, we showed that an intensional logic program of an arbitrary intensional logic has the minimum model property provided that the intensional operators appearing in the program satisfy monotonicity, universality, conjunctivity and finitariness requirements. In particular, we established that each property of intensional

operators is related to some property of intensional logic programs. For instance, monotonicity is related to the model existence and model-intersection properties; universality is related to the model existence property as well; conjunctivity is directly related to the model-intersection property and thus guarantees the existence of the minimum intensional Herbrand model. Finitariness (continuity) is the computability requirement for intensional logic programs and it is related to the fixpoint characterization of the minimum intensional Herbrand model.

We demonstrated the generality of our approach by applying it to diverse intensional logic programming languages including Chronolog [Wad85], Templog [AM87], Temporal Prolog [Gab87], Molog [dC86], InTense [MF89] and Tokio [AFMo86]. In particular, we showed that Chronolog, Templog, InTense and Molog are instances of the general framework and enjoy the minimum model semantics. We gave a negative result for a clausal subset of Temporal Prolog, because it allowed non-conjunctive temporal operators in the heads of program clauses. Interval logic programming languages such as Tokio are not instances of the general framework presented in the dissertation. However, we provided a transformation from a given interval logic into a 2-dimensional logic and established that our theory extended to interval logic programs such as those of pure Tokio through transformation. Baudinet [Bau88] [Bau89] independently provided a declarative semantics for Templog programs. Balbiani et al [BdCH88] developed a declarative semantics for an instance of Molog.

In Chapter 6, we laid down the model-theory of intensional logic programs with choice predicates, a powerful logical tool for modeling non-determinism. We extended such intensional logic programs with non-Horn choice formulas to establish the relation between predicates and the corresponding choice predicates. Since the minimum model semantics no longer applied to extended programs due to non-determinism involved, we developed the declarative semantics of extended programs in terms of minimal intensional Herbrand models. We observed that not all mini-

mal models were constructible because of the logical structure of intensional logic programs. We therefore developed the notion of a constructible minimal model as the limit of an alternating chain of interpretations based on the mappings, extended $T_{\mathcal{P}}$ and $C_{\mathcal{P}}$.

In Chapter 7, we showed that intensional logic programming could be enriched with extra intensional operators without resorting to extending the underlying intensional logic. We regarded a certain kind of intensional program clauses as the definitions of intensional operators, and intensional logic programs with intensional operator definitions as meta-theories. Since the model theory of intensional logic programs did not extend to meta-theories, we provided a transformation procedure to compile out intensional operator definitions and their applications in order to obtain an "equivalent" program in the underlying logic.

We also considered recursive definitions and employed the fixpoint techniques [Man74] to solve a set of recursive definitions. Recursive definitions definitely added more expressive power to ILP; but we had to employ an infinitary intensional logic such as intensional $L_{\omega_1\omega}$ [Kei71]. With respect to the least fixpoint theory, intensional operator definitions corresponded to continuous functions in the underlying algebra. A temporal fixpoint calculus such as that of Barringer [Bar87] offers both the least and the greatest fixpoint constructors. However, we pointed out that the greatest fixpoint constructors did not yield continuous functions. Baudinet [Bau89] obtained a similar result from a different perspective, using the fixpoint theory of temporal logic programs. Baudinet also compared the expresiveness of (propositional) temporal logic programming with that of the positive fragment of μTL of Vardi [Var88] based on a temporal fixpoint calculus.

8.2 Further Research Directions

The model-theoretic semantics of intensional logic programming is not complete. It lacks rules of inference and therefore we do not have the connections between the model-theoretical and proof-theoretical semantics of intensional logic programs. Although we laid down some rules of inference for temporal logic, we did not define a sound and complete proof procedure for temporal logic programming. Therefore more work has to be done. For the other languages described in the dissertation, we conjecture that complete proof procedures can be developed. A non-trivial research problem is to find generalized proof procedures for intensional logic programming, independent of any particular language. Such a proof procedure may be based on consequence operations of Tarski (see [Woj88] for a more detailed exposition to the topic.) Of course, we also need rules of inference for choice predicates. Any proof procedure involving choice predicates must have some sort of a memory to keep track of the choices made.

In the dissertation, we restricted the discussion only to those intensional logics where the notion of a possible world was central. In Chapter 4, we abstracted the denotations of intensional operators as functions over $P(\mathcal{U})$, extending the complete Boolean algebra $< P(\mathcal{U}), \emptyset, \mathcal{U}, \neg, \cap, \cup >$. There are more general algebraic approaches to the semantics of intensional logic, including modal algebras and topological Boolean algebras [Woj88] [BS84]. For instance, a modal algebra is an extension of a Boolean algebra $< B, 0, 1, \neg, \cap, \cup >$ with an additional operator, say $\|\Box\|$, satisfying the following conditions:

- $\|\Box\|(1) = 1$,
- $\|\Box\|(X \cap Y) = \|\Box\|(X) \cap \|\Box\|(Y)$ for all $X, Y \in B$.

The first condition says that $\|\Box\|$ turns universal truth into universal truth; the second condition is the conjunctivity requirement for $\|\Box\|$.

A theory based on algebras can possibly extend to many other non-classical logic programming languages, such as multiple-valued schemes of Fitting [Fit88] and Blair [B+88]. However, possible worlds in general are not a feature of algebraic semantics and hence we can no longer talk about context-dependent properties of certain problems. Broadly speaking, intensional logic programming based on possible worlds semantics becomes an instance of the more general framework. In such a theory, we also loose the neighborhood semantics and some of the properties of intensional operators such as finitariness. However, continuity may compensate the loss of finitariness.

Although intensional operator definitions correspond to continuous functions in the underlying algebra, there are many continous functions over $P(\mathcal{U})$ which cannot be defined in terms of restricted intensional operator definitions of Chapter 7. For instance, consider the temporal operator **start**. If the restrictions are lifted, we can define **start** by the following program clause

$$\texttt{first start}(A) \leftarrow \texttt{first}\, A.$$

With respect to the clause, any formula of the form **start** B will rewrite into B provided that the current moment is 0. At any other moment t in time, **start** B is false, since the clause does not apply for $t \geq 1$. In the temporal logic, **start** cannot be defined by any monotonic formula, but it can be defined in temporal logic programming.

We can use recursion to define more complicated intensional operators such as the temporal operator **even** where **even** B reads "the current moment is even and B is true". Below is the definition of **even**.

$$\texttt{first even}(A) \leftarrow A.$$

$$\texttt{next next even}(A) \leftarrow \texttt{even}(\texttt{next next}\, A).$$

It can be shown that meta-rule invocation rule will correctly interpret these two clauses as the definition of **even**. Given a formula of the form **even**(B) and the

$t \in \omega$, if t is even, by pushing **next** inwards an even number of times (using the second clause), we obtain the formula $\mathbf{even}(\mathbf{next}^t\ B)$ and the current moment 0; then we reduce the formula to $\mathbf{next}^t\ B$ by the first clause. When t is not even, we can reduce the current moment only to 1 in which case neither clause is applicable. Thus $\mathbf{even}(B)$ fails.

Since the fixpoint theory in its current form does not suffice to explain the meaning of unrestricted definitions, we must develop another technique (perhaps based on higher-order logic) as the basis for further investigation. Another research problem here is to establish the expressive power of ILP with unrestricted intensional operator definitions, and to investigate some functional completeness results. For instance, as the examples given above suggest, temporal logic programming is more expressive than the temporal logic it is based on. Baudinet [Bau89] draws a similar conclusion for temporal logic programming in Templog.

8.3 Implementation Problems

Wadge [Wad85] [Wad88] suggested that intensional logic programming could be implemented by adapting an existing logic programming implementation. Possible worlds could be represented as tags on formulas, not as extra parameters to predicates. Then the matching algorithm could be adapted to consider tags separately from other terms.

For instance, in temporal logic programming, the query $\leftarrow$ `fib(N)` causes the implementation to produce time-tagged goals of the form `fib(N).0`, `fib(N).1`, `fib(N).2` etc. The goal `fib(N).0` matches the first clause, and we obtain a ground instance of the goal with `N` substituted by `0`. Similarly, the query `fib(N).1` matches the second clause, and we obtain a ground instance of the query with `N` substituted by `1`. Then the query `fib(N).2` matches the third clause in the program given

above, and generates the following query: `fib(X).1,fib(Y).0,(N is X+Y).0` and so on. These actions can be justified by the rules of inference described earlier.

Efficient implementations of intensional logic programming must combine features of an implementation based on a resolution type proof procedure with features of a dataflow implementation, such as that of [FW85]. Otherwise it would waste resources recomputing results over and over again. For instance, suppose we want to prove the query ← `fib(M), next fib(N)` at time 10. Then the goals `fib(M).10` and `fib(N).11` are generated. The answer to the first goal is the ground instance `fib(55).10`. If the implementation stores in an associative memory this answer and other goals proved so far, the second goal can be proved by just modus ponens and substitution from the third clause in the program.

David Rolston [Rol86] of Arizona State University is considering an implementation of Chronolog along these lines. Choice predicates should not complicate implementations with an associative memory too much. We need only keep a record of choices made, as they are made.

Bibliography

[Abi65] Alexander Abian. *The Theory of Sets and Transfinite Arithmetic.* W. B. Saunders Company, Philadelphia and London, 1965.

[Add65] John W. Addison. The method of alternating chains. In Leon Henkin John W. Addison and Alfred Tarski, editors, *The Theory of Models.* North–Holland Publishing Company, Amsterdam, 1965.

[AFMo86] T. Aoyagi, M. Fujita, and T. Moto-oka. Temporal logic programming language Tokio. In E. Wada, editor, *Logic Programming'85*, pages 138–147. Springer-Verlag, 1986. LNCS 221.

[AM87] M. Abadi and Z. Manna. Temporal logic programming. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 4–16, San Fransisco, CA, 1987.

[AvE82] Krzysztof R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of Association for Computing Machinery (ACM)*, 29:841–862, July 1982.

[AW76] Edward A. Ashcroft and William W. Wadge. Lucid – a formal system for writing and proving programs. *SIAM J. COMPUTING*, 5:336–54, September 1976.

[B+88] H.A. Blair et al. A logic programming semantics scheme, Part I. Technical Report LPRG–TR–88–8, Logic Programming Research Group, Syracuse University, 1988.

[Bar87] H. Barringer. The use of temporal logic in the compositional specification of concurrent systems. In A. Galton, editor, *Temporal Logics and Their Applications.* Academic Press, 1987.

[Bau88] M. Baudinet. On the semantics of temporal logic programming. Technical Report STAN–CS–88–1203, Computer Science Dept., Stanford University, Stanford, CA, June 1988.

[Bau89] M. Baudinet. Temporal logic programming is complete and expressive. In *Conference Record of the Sixteenth ACM Symposium on Principles of Programming Languages*, Austin, Texas, January 1989.

[BdCH88] Ph. Balbiani, Luis Fariñas del Cerro, and A. Herzig. Declarative semantics for modal logic programs. In *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems.* ICOT, 1988.

[BM73] G. Battani and H. Meloni. Interpreteur du language de programmation PROLOG. Technical report, Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille, 1973.

[BS84] R. Bull and K. Segerberg. Basic modal logic. In D. Gabbay and F. Guethner, editors, *Handbook of Philosophical Logic, Vol. II.* D. Reidel Publishing Company, 1984.

[C+73] A. Colmerauer et al. Un systeme de communication homme-machine en Francais. Technical report, Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille, 1973.

[CG81] K. Clark and S. Gregory. A relational language for parallel programming. In *Proceedings of the ACM Conference on Functional Programming and Computer Architectures*, pages 171–178, October 1981.

[CG86] K. Clark and S. Gregory. PARLOG: Parallel programming in logic. *ACM Trans. on Programming Languages and Systems*, 8:1–49, January 1986.

[Che80] Brian F. Chellas. *Modal Logic : An Introduction.* Cambridge University Press, Cambridge, 1980.

[CK73] C. C. Chang and H. J. Keisler. *Model Theory.* North–Holland Publishing Company, Amsterdam, 1973.

[CL73] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[dBK88] J.W. de Bakker and J.N. Kok. Uniform abstraction, atomicity and contraction in the comparative semantics of Concurrent Prolog. In *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems 1988.* ICOT, 1988.

[dC86] Luis Fariñas del Cerro. MOLOG: A system that extends Prolog with modal logic. *New Generation Computing*, 4:35–50, 1986.

[Fit87] Melvin Fitting. Enumeration operators and modular logic programming. *Journal of Logic Programming*, 4:11–21, 1987.

[Fit88] Melvin Fitting. Logic programming on a topological bilattice. *Fundamenta Informaticae*, XI:209–18, 1988.

[FL77] M. J. Fischer and R. E. Ladner. Propositional modal logic of programs. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 286–94, Boulder, Colorado, May 1977. ACM.

[Flo67] R. W. Floyd. Assigning meaning to programs. In *Proceedings of AMS Symposium on Applied Mathematics*, pages 19–31, Providence, RI, 1967. American Mathematical Society.

[FW85] A.A. Faustini and W.W. Wadge. An eductive interpreter for the functional language Lucid. Technical Report DCS–46–IR, Dept. of Computer Science, Univ. of Victoria, B.C., Canada, June 1985.

[Gab87] D. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics and Their Applications.* Academic Press, 1987.

[Gal86] Jean H. Gallier. *Logic for Computer Science : Foundations of Automatic Theorem Proving.* Harper and Row, New York, 1986.

[GCLS88] R. Gerth, M. Codish, Y. Lichtenstein, and E. Shapiro. Fully abstract denotational semantics for Flat Concurrent Prolog. Technical Report CS88–03, Dept. of Applied Math. and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, April 1988.

[Gol87] Robert Goldblatt. *Logics of Time and Computation.* CSLI – Center for the Study of Language and Information, Stanford University, 1987. Lecture Notes no:7.

[Gue88] I. Guessarian. A note on fixpoint techniques in data base recursive logic programs. *Informatique theorique et Applications/Theoretical Informatics and Applications*, 22:49–56, 1988.

[HA88] Carl Hewitt and Gul Agha. Guarded Horn clause languages: Are they deductive and logical? In *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems.* ICOT, 1988.

[Har84] David Harel. Dynamic logic. In D. Gabbay and F. Guethner, editors, *Handbook of Philosophical Logic, Volume II : Extensions of Classical Logic.* D. Reidel Publishing Company, 1984.

[HC68] G.E. Hughes and M.J. Creswell. *An Introduction to Modal Logic.* Methuen and Co Ltd, London, 1968.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[Kei71] H.J. Keisler. *Model Theory for Infinitary Logic.* North–Holland Publishing Company, Amsterdam, 1971.

[KM76] K. Kuratowski and A. Mostowski. *Set Theory, With an Introduction to Descriptive Set Theory.* PWN — Polish Scientific Publishers, Warszawa, 1976.

[Kow74] R. A. Kowalski. Predicate logic as programming language. In *Proceedings of IFIP'74*, pages 569–574, Amsterdam, 1974. North–Holland Publishing Company.

[Kow79] R. A. Kowalski. *Logic for Problem Solving.* North–Holland Publishing Company, Amsterdam, 1979.

[Llo84] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[Man74] Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill Inc., 1974.

[MF89] W. H. Mitchell and A. A. Faustini. The intensional logic language In-Tense. In *Proceedings of the 1989 International Symposium on Lucid and Intensional Programming*, Arizona State University, Tempe, Arizona, May 8 1989.

[Mon74] R. Montague. *Formal Philosophy, Selected Papers of Richard Montague.* Yale University Press, 1974. edited by Richmond Thomason.

[Mos86] Ben Moszkowski. *Executing Temporal Logic Programs.* Cambridge University Press, Cambridge, England, 1986.

[MP81] Z. Manna and A. Pnueli. Verification of concurrent programs: the temporal framework. In Boyer and Moore, editors, *Correctness Problem in Computer Science.* Academic Press, 1981.

[Mur88] M. Murakami. A declarative semantics of parallel logic programs with perpetual processes. In *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems.* ICOT, 1988.

[NA85] M.A. Nait Abdallah. On some topological properties of logic programs. In L. Budach, editor, *Fundamentals of Computing Theory, FCT'85.* Springer-Verlag, 1985. LNCS No:199.

[OW88a] M.A. Orgun and W.W. Wadge. CHRONOLOG: A temporal logic programming language and its formal semantics. University of Victoria, Canada, January 1988.

[OW88b] M.A. Orgun and W.W. Wadge. A theoretical basis for intensional logic programming. In *Proceedings of the 1988 International Symposium on Lucid and Intensional Programming*, pages 33–49, Sidney, B.C., April 7–8 1988.

[OW89a] M.A. Orgun and W.W. Wadge. A formal treatment of non-determinism in intensional logic programming. In *Proceedings of the 1989 International Symposium on Lucid and Intensional Programming*, Arizona State University, Tempe, Arizona, May 8 1989.

[OW89b] M.A. Orgun and W.W. Wadge. Towards a unified theory of intensional logic programming. Technical Report DCS-112-IR, Department of Computer Science, University of Victoria, March 1989. (a revised and extended version will appear in the *Journal of Logic Programming*).

[OW91] M. A. Orgun and W. W. Wadge. Theory and practice of temporal logic programming. In Luis Fariñas del Cerro and Martti Penttonen, editors, *Non-Classical Logic Programming.* to be published by the Oxford University Press, 1991.

[Pra80] Vaughan R. Pratt. Application of modal logic to programming. *Studia Logica*, XXXIX(2/3):257–274, 1980.

[Rob65] J.A. Robinson. A machine-oriented logic based on resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Rol86] D. Rolston. CHRONOLOG: A pure tense-logic-based infinite-object programming language for parallel symbolic execution. Ph.D. Proposal, Arizona State University, August 1986.

[Rol87] D. Rolston. Toward a tense-logic-based mitigation of the frame problem. In *Proceedings of the AAAI Workshop on the Logical Frame Problem*, Lawrence, Kansas, April 1987. Morgan and Kaufman.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic.* Springer-Verlag, 1971.

[Sak87] Y. Sakakibara. Programming in modal logic: An extension of PROLOG based on modal logic. In E. Wada, editor, *Logic Programming'86*, pages 81–91. Springer-Verlag, 1987. LNCS 264.

[Sar87] Vijay A. Saraswat. GHC: Operational semantics, problems and relationship with CP($\downarrow, |$). In *Proceedings of the 1987 Symposium on Logic Programming*, pages 347–358, San Fransisco, CA, Aug.31-Sept.4 1987.

[Sco70] D. Scott. Advice on modal logic. In K. Lambert, editor, *Philosophical Problems in Logic.* D.Reidel Publishing Company, 1970.

[Seg82] K. Segerberg. *Classical Propositional Operators.* Clarendon Press, Oxford, 1982.

[Sha87] E. Shapiro. A subset of Concurrent Prolog and its interpreter. In E. Shapiro, editor, *Concurrent Prolog: Collected Papers.* MIT Press, 1987.

[Sto77] Joseph E. Stoy. *Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[Ued86] K. Ueda. Guarded Horn Clauses. In E. Wada, editor, *Logic Programming'85*, pages 168–179. Springer-Verlag, 1986. LNCS 221.

[Var88] M. Vardi. A temporal fixpoint calculus. In *Conference Record of the Sixteenth ACM Symposium on Principles of Programming Languages*, pages 250–259, San Diego, CA, January 1988.

[vB84] J.F.A.K. van Benthem. Possible worlds semantics: A research program that cannot fail? *Studia Logica*, XLIII(4):379–93, 1984.

[vB88] J.F.A.K. van Benthem. *A Manual of Intensional Logic.* CSLI – Center for the Study of Language and Information, Stanford University, second edition, 1988. Lecture Notes no:1.

[vEdLF82] M.H. van Emden and G.J. de Lucena Filho. Predicate logic as a language for parallel programming. In K.L. Clark and S.A. Tarlund, editors, *Logic Programming.* Academic Press, 1982.

[vEK76] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23:733–42, 1976.

[vENA84] M.H. van Emden and M.A. Nait Abdallah. Top-down semantics of fair computations of logic programs. Technical Report CS–84–27, Dept. of Computer Science, University of Waterloo, October 1984.

[WA85] W.W. Wadge and E.A. Ashcroft. *Lucid, the Dataflow Programming Language.* Academic Press, 1985.

[Wad85] W.W. Wadge. Tense logic programming: a sane alternative. Dept of Computer Science, University of Victoria, 1985.

[Wad88] W.W. Wadge. Tense logic programming: a respectable alternative. In *Proceedings of the 1988 International Symposium on Lucid and Intensional Programming*, pages 26–32, Sidney, B.C., April 7-8 1988.

[Woj88] R. Wojcicki. *Theory of Logical Calculi.* Kluwer Academic Publishers, 1988.

[Yag84] Ali A. Yaghi. *An Intensional Implementation Technique for Functional Languages.* PhD thesis, Dept. of Computer Science, University of Warwick, Coventry, England, 1984.