Extending Temporal Logic Programming with Choice-Predicates Nondeterminism

Mehmet A. Orgun    William W. Wadge

Abstract

In temporal logic programming, a stream can be specified by a single-valued, time-varying predicate which, at any given moment in time, represents the corresponding element in the stream. However, due to inherent nondeterminism in logic programming, time-varying predicates do not necessarily represent single-valued relations at any given moment in time. Choice predicates are also time-varying predicates, but, in principle, they act like a dataflow node with multiple input lines which nondeterministically selects one of its inputs as output. Therefore they are guaranteed to be single-valued at all moments in time, and they can be regarded as representing "nondeterministic" streams. Users do not define choice predicates, they are supplied automatically for all predicates defined in temporal logic programs. Inputs to choice predicates are supplied by the corresponding predicates. When the connection between choice predicates and the corresponding predicates is established, we obtain non-Horn temporal logic programs as a result. The model-theoretic semantics of such a program is developed in terms of "minimal models". However, the logical structure of the program dictates which minimal models are constructible from the program. We in particular discuss a characterization of constructible minimal models as limits of chains of models obtained by alternating applications of two new mappings $NT_P$ and $C_P$. The paper also outlines a proof procedure for the temporal language Chronolog extended with choice predicates.

1 Introduction

In logic programming languages, streams are usually represented by infinitary data structures such as infinite lists. Through streams, non-terminating dataflow computations can be modeled in concurrent logic programming languages such as Parlog [14] and Concurrent Prolog [40]. However, the intended meaning of a concurrent program written in such a language is not characterized by the minimum model semantics of van Emden and Kowalski [42]. To explain the intended meaning of concurrent (infinitary) logic programs, several researchers [25, 28, 43] have proposed the greatest fixed-point semantics based on a mapping, called $T_P$, over Herbrand interpretations of a given logic program $P$. $T_P$ is the one-step ground modus-ponens function whose least fixed point characterizes the set of atomic consequences of $P$ (i.e., the minimum Herbrand model of $P$). The conclusion is that the declarative reading of a concurrent logic program is no longer relevant to explain its run-time behavior [20, 21, 31]. As Shapiro [40] puts it, "Computable consequences of a given concurrent logic program can be determined only by reference to the operational semantics." Moreover, due to inherent nondeterminism in logic programming, it is in general impossible to justify partial results coming from a computation that constructs an infinitary object with respect to any semantic approach.

In temporal logic programming [1, 2, 5, 8, 17, 31, 44], the notion of time is abstracted away from the object language. Owing to this abstraction, temporal logic programming (TLP) offers another approach to modeling non-terminating computations: a dataflow stream can be represented by a single-valued, time-varying predicate which, at any given moment in time, represents the corresponding element in the stream. There are temporal languages which are not based on
the logic programming paradigm and the Horn subset of temporal logic, for example, the interval
temporal logic language Tempura [27]. There are also first-order treatments of temporal modalities in the
logic programming framework, including those of Hrycej [22] and Brzoska [10, 11]. This paper in
particular focuses on TLP languages that are directly based on the Horn subset of temporal logic
and resolution-type proof procedures, such as Chronolog [29, 31, 37] and Templog [1, 2].

In Chronolog, non-terminating computations can be naturally modeled through time-varying
predicates. For instance, in the following Chronolog program, the $fib$ predicate represents the
$(t + 1)$-st Fibonacci number at time $t \geq 0$ and no other.

\[
\text{first } fib(0).
\text{first next } fib(1).
\text{next next } fib(N) \leftarrow \text{next } fib(X), \ fib(Y), \ N \text{ is } X + Y.
\]

Program clauses are read as assertions true at all moments in time. The temporal operators
\texttt{first} and \texttt{next} refer to the initial moment and the next moment in time respectively. According
to the program, $fib(0)$ would be true at time 0, $fib(1)$ at time 1, $fib(1)$ at time 2, $fib(2)$ at time 3,
and so on. A non-terminating computation is triggered by an attempt to prove a given goal at all
moments in time. For instance, given the goal $\leftarrow fib(X)$, an implementation of TLP would initiate
a non-terminating computation to prove the goal at each moment in time in turn, starting from
the initial moment, producing an infinite sequence of answer substitutions for $X$, that is, \{X/0\},
\{X/1\}, \{X/1\}, \{X/2\}, \{X/3\} and so on, for as long as time and resources permit. Since $fib$ is
single-valued at each moment in time, it is possible to regard the $fib$ predicate as a dataflow node
producing the stream \{0, 1, 1, 2, 3, \ldots\} of Fibonacci numbers element-wise. Therefore we also say
that $fib$ represents the Fibonacci stream.

In short, TLP can model non-terminating computations as an infinite series of finite computa-
tions, one for each moment in time. Answers obtained from each finite computation are justified
by the minimum model semantics [29, 31]. For instance, in the above example, $fib(0)$ is a logical
consequence of the program at time 0, and it can be shown to be so by a finite computation
(proof). In Orgun and Wadge [33], it is shown that Chronolog admits a sound and complete proof
procedure called TiSLD-resolution.

1.1 Motivation

There is a limitation of TLP in its current form when we want to model stream-oriented dataflow
computations: Because of the inherent nondeterminism in logic programming, time-varying pred-
cicates in general do not necessarily represent single-valued relations at each moment in time. The
problem is that, in many cases, we are interested in only one solution to a given goal, but we do
not want to make the choice unnecessarily specific in a given logic program. Perhaps the problem
specification itself does not constrain the programmer to produce one particular stream; therefore
it would be against the spirit of logic programming to force the programmer to make a specific
choice.

For instance, consider the game of Mastermind [39], in which a player picks a secret code
consisting of a list of $n$ distinct decimal digits, and a second player tries to decipher the code by
making guesses. Each guess is answered by the number of “bulls” (number of digits that appear
in exactly the same positions in both the code and the guess), and the number of “cows” (number
of digits that appear in both the code and the guess, but in different positions.) The game is
over when a guess has $n$ bulls. At each step of the game, there may be more than one possible
guess, but, there should be only one actual guess. In the beginning, any list of $n$ distinct digits is a
consistent guess, but as the game progresses, the number of consistent guesses decreases depending
on the actual guesses made and answers obtained for them.

The steps in the game naturally correspond to the moments in time. Therefore we may
regard actual guesses as forming a stream, determined by arbitrary choices made from possible
guesses. Let the $guess$ predicate represent the actual guesses, and the $consistent$ predicate all the

consistent guesses. In Chronolog, we might attempt to write the following clause to specify the guess predicate:

\[
guess(Code) \leftarrow \text{consistent}(Code).
\]

The problem with this clause is that \(\text{guess}\) is not necessarily single-valued; it represents exactly the same guesses as those represented by \(\text{consistent}\). As it is not called for by the specification, we cannot force \(\text{guess}\) to represent a particular stream either. A solution to the problem can be provided in concurrent logic programming owing to the committed-choice nondeterminism [14] and the underlying execution mechanism of concurrent logic languages. Supposing the definition of \(\text{consistent}\) is given elsewhere, the following clause would do the job:

\[
guess([\text{Code} \mid L]) \leftarrow \text{consistent}(Code), \text{guess}(L).
\]

Given the \(\text{goal} \leftarrow \text{guess}(L)\), an implementation of concurrent logic programming would produce the initial segments of a stream of actual guesses as partial output, until the code is deciphered. The implementation will commit to only one of the choices at each call to \(\text{consistent}(Code)\) and will not backtrack. Therefore the operational semantics will guarantee that the \(\text{guess}\) predicate represents a unique stream of actual guesses.

However, we would like to avoid such a solution to the problem that cannot be formalized in the object language because of its dependency on the underlying execution mechanism. With respect to the standard minimum-model semantics [25], the \(\text{guess}\) predicate represents all possible streams of consistent guesses, not a unique one. But the problem specification requires us to produce exactly one stream, chosen arbitrarily amongst all possible ones. We also believe that the way the language is implemented should be irrelevant in explaining the behavior of a given logic program. Therefore we propose “choice-predicates nondeterminism” [30, 44] as a solution to the problem for TLP.

1.2 Choice Predicates

Choice predicates are not defined by users in temporal logic programs, but they are supplied by implementations for each predicate used in a given program. The connection between choice predicates and the corresponding predicates is established automatically. A choice predicate, in principle, acts like a dataflow node with multiple (possibly infinitely many) input lines which arbitrarily selects one of its inputs as its output. The inputs are provided by the corresponding predicate. For the example given above, the choice predicate associated with the consistent predicate is denoted as \(\#\text{consistent}\). Note that the symbol “\(\#\)” is not an operator, it is part of the name of the choice predicate. The \(\#\text{consistent}\) predicate represents the unique guess chosen arbitrarily amongst the consistent guesses represented by \(\text{consistent}\). Supposing the \(\text{consistent}\) predicate is defined elsewhere, the \(\text{guess}\) predicate can be defined in Chronolog extended with choice predicates as follows:

\[
guess(Code) \leftarrow \#\text{consistent}(Code).
\]

If this is the only definition of \(\text{guess}\), the \(\text{guess}\) predicate is single-valued at each moment in time and it represents a stream of actual guesses over the course of time. This is because the \(\#\text{consistent}\) predicate is guaranteed to be single-valued with an arbitrary but definite choice made for each moment in time, and so is the \(\text{guess}\) predicate, because it depends on \(\#\text{consistent}\). The \(\#\text{consistent}\) predicate may be regarded as representing a nondeterministic stream. Given the goal \(\text{goal} \leftarrow \text{guess}(Code)\), an implementation would start a (potentially non-terminating) computation to prove the goal at each moment in time until the code is deciphered, producing a stream of actual guesses along the way.

The bookkeeping of the previous guesses and answers to them can be easily provided without any \texttt{assert} commands. For instance, let the \texttt{allGuesses} predicate represent a list of all actual
guesses at each step of the game, defined as follows:

\[
\text{first allGuesses}([\text{Code}]) \leftarrow \text{guess}(\text{Code}),
\]

\[
\text{next allGuesses}([\text{Code} | L]) \leftarrow \text{guess}(\text{Code}), \text{allGuesses}(L).
\]

The \text{allGuesses} predicate can be used in the definition of the \text{consistent} predicate in determining consistent guesses. The \text{reply} predicate representing answers to the actual guesses should be provided as input to the program, depending nondeterministically on the \text{guess} predicate. However, we do not develop the program any further. In summary, choice predicates allow for nondeterministic dataflow computations within a logic-programming formalism.

When the connection between each predicate and the corresponding choice predicate is established by non-Horn axioms (choice formulas [30]), we obtain non-Horn temporal logic programs as a result. Moreover, a first-order logic with equality is required to establish the connection. Therefore the meaning of a temporal logic program with choice predicates cannot be characterized by the minimum (temporal) model semantics.

As an alternative to introducing implicit choice formulas, we could regard the symbol “#” as a second-order choice operator. However, this approach would not work in practice, because the extensionality principle would imply that co-extensional predicates (those predicates that represent the same relation) have co-extensional choice predicates. For instance, suppose that we are given a program which includes the definition of a predicate, say \text{p}. Given the goal \(\text{goal} \leftarrow \text{#p}(X)\), before producing any results, any implementation should perform the impossible task of guaranteeing that the extensionality principle has not been violated for \text{p}, which means that the meaning of all predicates defined in the program must be checked and that the same term must be chosen for all those predicates which are co-extensional with \text{p}. A first-order treatment of the choice symbol avoids such impossible constraints about co-extensionality.

1.3 Results

This paper investigates the model-theoretic semantics of TLP with choice predicates. In particular we provide a general semantic framework which can be applied to temporal languages with certain properties such as the minimum-model semantics and the fixed-point semantics. It is shown that a given temporal logic program, plus copies of non-Horn choice formulas for each predicate used, has at least one minimal model. As the program does not necessarily single out any particular minimal model due to the choice-predicates nondeterminism, any minimal model can be regarded to be the meaning of the program. The notion of a minimal model is pure and simple, and it does not require a complex operational model to explain the meaning of the program, and thus program clauses can still be read declaratively.

It can be shown that not all of the minimal models of a given temporal logic program are “constructible” due to the logical structure of the program. This paper gives a characterization of constructible minimal models as the limit of an alternating chain of models constructed using two new mappings: an extended \(T_P\) operator and a \(C_P\) operator. We refer to constructible minimal models as “canonical models”, since these models are the ones dictated by the structure of the program. Perhaps the programmer has one of these canonical models in mind when writing a program. Minimal models and canonical models are analogous to minimal models and perfect models [34] of disjunctive databases.

The paper also outlines a proof procedure based on a temporal SLD-resolution with memory to keep track of the arbitrary choices made for each choice predicate in a given computation. The procedure is designed for a specific temporal language, namely Chronolog. In theory, choice predicates can model unbounded nondeterminism, because at any given moment in time, time-varying predicates may represent infinitely many terms. For instance, given the definition of a \text{nat} predicate that represents the set of natural numbers in successor notation, \text{#nat} represents a unique natural number chosen arbitrarily. Of course, there are infinitely many natural numbers, all of which are legitimate choices. In practice, since any canonical model is acceptable, it does not matter whether choices are made from a finite number of values or not, as long as a correct answer is produced. This is also how the correctness of an implementation is defined.
2 Temporal Logic Programming

Temporal logic programming (TLP) is a new form of logic programming based on temporal logic. There are a number of temporal languages: Chronolog [29, 37, 44, 31] and Templog [1, 2, 8] are based on a temporal logic with linear and non-ending time; Temporal Prolog [17] is based on both linear and branching time; the multidimensional intensional language InTense [26] restricted to just a temporal dimension is very similar to Chronolog, but with an unbounded past and future; and Tokio [5] is based on the same interval logic as Tempura [27]. We refer the reader to the literature for more details on TLP and its applications [15, 19].

In principle, choice predicates can be used in programs written in any of the temporal languages mentioned above, based on the logic programming paradigm. In this paper, we focus on Chronolog to demonstrate the basic ideas of TLP. This section outlines some of the properties of the underlying temporal logic of Chronolog. We also show how non-terminating computations and infinite objects such as streams can be modeled by time-varying predicates in Chronolog.

2.1 Temporal Logic

Temporal logic [35] is a kind of modal logic in which the set of “possible worlds” models a collection of moments in time (usually discrete, linearly ordered and without a last moment); see Chellas [13] for more details on modal logic. In the linear-time temporal logic of Chronolog, the collection of moments in time is the set \( \omega \) of natural numbers with its usual ordering relation \(<\). The temporal logic offers two temporal operators: first and next. Informally, the temporal operators refer to the initial moment and the next moment in time, respectively. The syntax of the temporal logic extends that of first-order logic with equality with two new formation rules: if \( A \) is a formula, so are \( \text{first} \ A \) and \( \text{next} \ A \). Note that the temporal operators are applied to formulas, not to terms of the language. We write \( \text{next}^n \) for \( n \) successive applications of next. In case \( n = 0 \), \( \text{next}^0 \) is the empty string. Throughout the paper, we refer to the underlying logic simply as TL.

The semantics of formulas of TL are provided by temporal interpretations. A temporal interpretation of TL assigns meanings at all moments in time to all basic elements of the language such as function symbols, predicate symbols and variables. Interpretations are extended upward to all terms and formulas of TL by a satisfaction relation \( \models \). The meaning of a formula of TL varies in time, but, we restrict the discussion to those temporal interpretations in which the values of variables and function symbols are “rigid”. The value of a rigid term is constant at all moments in time. The formal definition of a temporal interpretation is given as follows:

**Definition 1.** A temporal interpretation \( I \) of TL consists of a nonempty set \( D \), called the domain of the interpretation, over which the variables range, together with for each variable, an element of \( D \); for each \( n \)-ary function symbol, an element of \( D^n \); and for each \( n \)-ary predicate symbol, an element of \( \omega \to \mathcal{P}(D^n) \).

We now give the definition of the satisfaction relation \( \models \) in terms of temporal interpretations. In the following, the fact that a formula \( A \) is true at a moment \( t \) in time in some temporal interpretation \( I \) is denoted as \( \models_{I,t} A \). Let \( I(e) \) denote the value in \( D \) that \( I \) gives each term \( e \).

**Definition 2.** The semantics of elements of TL are given inductively by the following, where \( I \) is a temporal interpretation of TL, \( t \in \omega \), and \( A \) and \( B \) are formulas of TL.

1. If \( f(e_0, \ldots, e_{n-1}) \) is a term, \( I(f(e_0, \ldots, e_{n-1})) = I(f)(I(e_0), \ldots, I(e_{n-1})) \in D \).
2. For any \( n \)-ary predicate symbol \( p \) and terms \( e_0, \ldots, e_{n-1} \), \( \models_{I,t} p(e_0, \ldots, e_{n-1}) \) if and only if \( (I(e_0), \ldots, I(e_{n-1})) \in I(p)(t) \).
3. \( \models_{I,t} \neg A \) if and only if it is not the case that \( \models_{I,t} A \).
4. \( \models_{I,t} A \land B \) if and only if \( \models_{I,t} A \) and \( \models_{I,t} B \).
5. \( \models_{I,t} (\forall x)A \) if and only if \( \models_{I[d/x],t} A \) for all \( d \in D \), where the interpretation \( I[d/x] \) is just like \( I \) except that the variable \( x \) is assigned the value \( d \) in \( I[d/x] \).
6. For terms $e_0$ and $e_1$, $\models_{I,t} e_0 \equiv e_1$ if and only if $I(e_0) = I(e_1)$.

7. $\models_{I,t} \text{first } A$ if and only if $\models_{I,0} A$.

8. $\models_{I,t} \text{next } A$ if and only if $\models_{I,t+1} A$.

The symbol $\equiv$ is used to denote the object language equality symbol. In the definition, items (1) through (6) are independent of the temporal logic under discussion. They can be regarded as specifying a semantics scheme for temporal logics. Items (7) and (8) are language-dependent, as they define the semantics of the temporal operators of TL.

If a formula $A$ is true in a temporal interpretation $I$ at all moments in time, we say that $A$ is true in $I$, or $I$ is a model of $A$, and denote this fact as $\models_I A$. Moreover, $\models A$ denotes the fact that $A$ is true in any temporal interpretation. We use the notation $\Gamma \models A$ to denote the fact that $A$ is true in every model of $\Gamma$, where $\Gamma$ is a set of formulas. A temporal interpretation is a model of a set of formulas $\Gamma$ if it is a model of every formula in $\Gamma$. In other words, $\Gamma \models A$ means that $A$ is a logical consequence of $\Gamma$. We regard $\neg$, $\land$, and $\exists$ as primitives and assume the usual definitions of $\lor$, $\rightarrow$, $\leftrightarrow$ and ($\forall$) in terms of these primitives.

2.2 Temporal Logic Programs

In Chronolog programs, program clauses contain (any number of) applications of temporal operators to atomic formulas. Note that atomic formulas are made up from predicates and terms of the language, excluding the equality predicate. We adopt the clausal notation [24] for Chronolog programs. All variables in program clauses are assumed to be universally quantified. For convenience, we use upper-case letters for variables and lower-case letters for function and predicate symbols. In the following, we call formulas of the form \text{next}^k A and \text{first next}^k A as "temporal atoms", where $A$ is atomic and $k \geq 0$. Temporal atoms of the form \text{first next}^k A are also called "rigid temporal atoms".

Definition 3.

1. A program clause is the universal closure of a clause of the form $A \leftarrow B_0, \ldots, B_{n-1}$, where $n \geq 0$ and all $B_i$’s and $A$ are temporal atoms. If $n = 0$, the program clause is called a unit clause.

2. A goal clause is the universal closure of a clause of the form $\leftarrow B_0, \ldots, B_{n-1}$, where $n = 0$ and all $B_i$’s are rigid temporal atoms. If $n = 0$, the goal clause is called the empty clause.

A Chronolog program is (the conjunction of) a set of program clauses regarded as assertions true at all moments in time. Goal clauses are required to be rigid, that is, all temporal atoms in a goal clause are fixed to some moment in time. When this restriction is lifted, goal clauses may be open-ended. Open-ended goals are used to initiate non-terminating computations. Answer substitutions from TiSLD-derivations [33] are regarded as computed answers to a given goal. For convenience, we regard instances of goals obtained by applying answer substitutions to the goal as computed answers as well.

2.3 Non-terminating Computations

Concurrent logic programming languages such as Parlog [14] and Concurrent Prolog [40] support non-terminating dataflow computations by employing infinitary data structures. In contrast, in TLP languages such as Chronolog, non-terminating computations can be modeled by time-varying predicates. Consider Hamming numbers which are multiples of 2, 3 and 5 of the form

$$2^i3^j5^k$$

for some $i$, $j$ and $k$. 
We write a Chronolog program to produce all Hamming numbers in increasing order without any repetitions. In the program given below, the \textit{hamming} predicate is true of the \((t+1)\)-st Hamming number at any given moment \(t \geq 0\) in time and no other.

\[
\text{hamming}(X) \leftarrow \text{residue}([X \mid L]).
\]

\[
\text{first residue}([1]).
\]

\[
\text{next residue}(L_1) \leftarrow
\text{residue}([X \mid L]), \text{times}(2, X, X_2), \text{times}(3, X, X_3), \text{times}(5, X, X_5), \text{merge}([X_2, X_3, X_5], L, L_1).
\]

\[
\text{merge}(L_1, L_2, L_3) \leftarrow \ldots
\]

\[
\text{times}(X, Y, Z) \leftarrow \ldots
\]

We assume a standard definition for \textit{merge} in which two ordered lists are merged to produce a third ordered list with duplicates removed. The \textit{residue} predicate holds all those Hamming numbers produced but not yet consumed by the \textit{hamming} predicate.

According to the program, the \textit{residue} predicate is true of the residue list \([1]\) at time 0, which means that, by the first clause, 1 is the first Hamming number, that is, the seed of the Hamming stream. At time 1, by the third clause, 1 is removed from the residue list and then multiples of 1 are merged with the rest of the residue list (the empty list \([\ ]\)). Thus the residue predicate represents the residue list \([2, 3, 5]\) at time 1, meaning that the second Hamming number is 2. At time 2, again by the third clause, 2 is removed from the residue list and then multiples of 2 are merged with the rest of the residue list (the list \([3, 5]\)). The residue predicate represents the residue list \([3, 4, 5, 6, 10]\) at time 2, meaning that the third Hamming number is 3. At the following moments in time, the computation proceeds along these lines.

A non-terminating computation of Hamming numbers is triggered by the open-ended goal clause \(\leftarrow \text{hamming}(N)\). The goal clause is interpreted as an infinite series of rigid goals of the form \(\leftarrow \text{first hamming}(N), \leftarrow \text{first next hamming}(N), \leftarrow \text{first next next hamming}(N)\) and so forth. Each rigid goal is considered as independent and the answers to rigid goals obtained by TiSLD-resolution constitute the answers to the original goal. For instance, \text{first next hamming}(2)\ is such an answer and therefore is a logical consequence of the program by the soundness of TiSLD-resolution \[33\]. An implementation of Chronolog based on TiSLD-resolution produces answers to rigid subgoals as long as time and resources permit.

In concurrent (infinitary) logic programming, it is possible to specify the \textit{hamming} predicate as true of the infinite list of all Hamming numbers, for example, see \[41\] for the solution in Concurrent Prolog. Then given such a program and the goal \(\leftarrow \text{hamming}(L)\), an implementation of infinitary logic programming would start a nonterminating computation producing partial answers to the goal, that is, initial segments of the list of all Hamming numbers. However, these partial answers are not justified by the minimum-model semantics: the \textit{hamming} predicate would not be true of any term in the minimum Herbrand model of the program. Therefore the intended meaning of such a program cannot be reasoned about from its declarative reading. Several researchers \[25, 28, 43\] have proposed the greatest fixed-point semantics based on Herbrand domains with infinite terms to model the intended (operational) meaning of infinitary logic programs.

In Chronolog, an infinite stream can be represented by a single-valued, time-varying predicate such as \textit{hamming}; at each moment in time the predicate represents the corresponding element from the infinite stream. Therefore the answers obtained from rigid subgoals of an open-ended goal are justified by the minimum-model semantics of Chronolog programs \[29, 31\]. This guarantees that a correct Chronolog interpreter can construct the minimum model of the program. Therefore program clauses can be read declaratively.

In Chronolog, we can also specify “intermittent streams” \[45\], that is, streams that do not necessarily have a value at every moment in time. For instance, consider the following clause:

\[
\text{hamming}_2(X) \leftarrow \text{hamming}(X), \text{mod}_2(X, 0).
\]
where $\mod_2$ specifies modulo 2 arithmetic. The $\text{hamming}_2$ predicate represents an intermittent stream of Hamming numbers that are divisible by 2 without remainder. When a given Hamming number is not divisible by 2 at some moment in time, it will be picked out of the Hamming stream and $\text{hamming}_2$ will not be true of any value at that moment.

3 Choice Predicates

In the Fibonacci and Hamming numbers examples, both of the $\text{fib}$ and $\text{hamming}$ predicates represent single-valued relations at any given moment in time. But, in the game of Mastermind, we cannot force the $\text{guess}$ predicate to be single-valued, since any of the possible guesses represented by the $\text{consistent}$ predicate are legitimate choices. Choice predicates are proposed as a solution to the problem. We now give other examples of the use of choice predicates to model a nondeterministic, non-terminating computation, and a mutual exclusion problem.

3.1 Nondeterministic Programming

Suppose we are required to produce an arbitrary stream of natural numbers starting from 0 in increasing order. The committed-choice mechanism [14, 41] of concurrent logic languages can be utilized to define the $\text{stream}$ predicate in the following concurrent (infinitary) logic program:

\[
\text{stream}([0|L]) \leftarrow \text{restOfStream}([0|L]).
\]

\[
\text{restOfStream}([X,Y|L]) \leftarrow X < Y, \text{restOfStream}([Y|L]).
\]

Given a goal like $\leftarrow \text{stream}(L)$, an implementation of concurrent logic programming produces partial answers to the query, that is, initial segments of an arbitrary stream of increasing numbers starting with 0, and never terminates. The implementation will commit to only one pair of $X$ and $Y$ at each call to $X < Y$ and will not backtrack.

It is possible to regard $\text{stream}(L)$ as a dataflow node producing a stream of natural numbers, one at a time. However, as mentioned before, the intended meaning of the program is not characterized by the minimum-model semantics of van Emden and Kowalski [42]. The minimum Herbrand model (i.e., the set of atomic consequences) of the program is the empty set. On the other hand, with respect to the greatest fixed-point semantics [28, 43], the $\text{stream}$ predicate represents all increasing infinite sequences of natural numbers. Hence we cannot force the $\text{stream}$ predicate to represent a single arbitrary stream with respect to any semantic approach. In Chronolog, the $\text{stream}$ predicate cannot be specified.

A solution in Chronolog with choice predicates can be provided by defining the time-varying $\text{stream}$ 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 $\text{stream}$ predicate will be true of a value (number) which is greater than the value generated at time $t$. We first define a predicate to formalize the possible values the $\text{stream}$ predicate could possibly represent at each moment in time. The definition relies on the fact that the $\text{stream}$ predicate is single-valued.

\[
\text{first values}(0).
\]

\[
\text{next values}(Y) \leftarrow \text{stream}(X), X < Y.
\]

At time 0, there is only one possible value, that is, 0. From then on, the possible values at any given moment in time are those numbers greater than the value represented by the $\text{stream}$ predicate at the previous moment in time. For instance, at time 1, the $\text{values}$ predicate is true of all values $Y$ such that $Y > 0$. The definition of the $\text{stream}$ predicate completes the program:

\[
\text{stream}(X) \leftarrow \#\text{values}(X).
\]

Here $\#\text{values}$ is the choice predicate associated with $\text{values}$, and it is guaranteed to be single-valued with an arbitrary but definite choice made at each moment in time. Then the $\text{stream}$
predicate represents a stream of output values chosen nondeterministically from those values the values predicate represents over the collection of moments in time. Choices made for the values 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 potentially represents an output stream. This means that we must supply choice predicates for every predicate used 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 in programs; they are supplied by the implementation automatically. Now programs may contain choice predicates appearing in the bodies of program clauses, and in goals.

The connection between values and the corresponding choice predicate #values is established by the following non-Horn “choice formulas”.

\[ F1. (\forall X)(#values(X) \rightarrow values(X)). \]

\[ F2. (\forall X)(#values(X) \rightarrow (\forall Y)(#values(Y) \rightarrow X = Y)). \]

\[ F3. (\exists X)(values(X)) \leftrightarrow (\exists Y)(#values(Y)). \]

We read all of these formulas as assertions true at all moments in time. F1 states that if the #values predicate is true of some term, the values predicate is true of the same term as well; F2 asserts the uniqueness of the term which the #values predicate is true of, and F3 states that whenever values is true of some term, the #values predicate is true of some term as well (not necessarily the same term) and vice versa.

Temporal logic programs now implicitly include these formulas for each predicate used. Since choice formulas do not belong to the Horn subset of logic, we have non-Horn temporal logic programs as a result. 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 nondeterministic streams depending 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 (see below).

### 3.2 Modeling Mutual Exclusion

Choice-predicates nondeterminism can be used to solve mutual-exclusion problems such as that of the dining philosophers [36] and resource-sharing by multiple processes. A solution to the dining philosophers problem in the choice-predicates extension of Chronolog is given elsewhere [31]. In this section, a solution to a resource-sharing problem is given. Suppose that we have a shared resource which can only be used by at most one process at any given moment in time. Having been granted access to the resource, a process uses it for only one unit of time and then releases it. There is no queuing of processes or any form of a priority scheme; therefore all the waiting processes compete for the resource without any restrictions. We require that each process be given a unique identifier (an integer value).

The resource is modeled by the inUse predicate, which is true of idle if there are no processes using the resource or true of the identifier of the process that has acquired access to the resource. The waiting predicate represents the identifiers of all processes that are waiting for access to the resource at any given moment in time, and waitList the list of the identifiers of waiting processes. The waiting predicate is defined in terms of the waitList and member predicates.

\[ \text{waiting}(Idno) \leftarrow \text{waitList}([H|T]), \text{member}(Idno, [H|T]). \]

The clause is defined for nonempty waiting lists. Since integer values are used as process identifiers, the definition of member can be easily provided in terms of the diff predicate, which is true of two given values if one of them is greater than the other. In the program, it is possible that #waiting represents an “intermittent” stream (it has no value when the waiting list is empty).
The following is the definition of the single-valued *inUse* predicate.

\[
\begin{align*}
\text{first } \text{inUse}(\text{idle}). \\
\text{next } \text{inUse}(\text{idle}) & \leftarrow \text{waitList}([]) . \\
\text{next } \text{inUse}(\text{Idno}) & \leftarrow \#\text{waiting}(\text{Idno}).
\end{align*}
\]

The first clause states that the resource is in the idle state at the initial moment in time. The second clause states that the resource will be in the idle state at the next moment if there are no processes waiting at the current moment. The third clause states that the resource will be used by the process identified by *Idno* selected nondeterministically amongst all waiting processes represented by waiting. With respect to the declarative reading of program clauses, all waiting processes have an equal chance of being selected by the choice predicate \#\text{waiting} at any given moment in time. The choice mechanism does not favor one selection over the others, and it is oblivious to the previous choices made already. In fact, each of the choices will be represented in some minimal model of the program (see below).

We now give the definition of \text{waitList}. Let the \text{insert} predicate represent the list of processes that request access to the resource at any given moment in time.

\[
\begin{align*}
\text{first } \text{waitList}([]) . \\
\text{next } \text{waitList}(\text{In}) & \leftarrow \text{waitList}([]), \text{insert}(\text{In}) . \\
\text{next } \text{waitList}(\text{L}) & \leftarrow \\
& \text{waitList}([\text{H}\mid \text{T}]), \text{inUse}(\text{Idno}), \text{insert}(\text{In}), \\
& \text{remove}(\text{Idno}, [\text{H}\mid \text{T}], \text{L}_1), \text{merge}(\text{L}_1, \text{In}, \text{L}). \\
& \text{remove}(\text{Idno}, \text{L}, \text{L}_1) \leftarrow \cdots \\
& \text{merge}(\text{L}_1, \text{L}_2, \text{L}) \leftarrow \cdots
\end{align*}
\]

We assume standard definitions for \text{merge} and \text{remove}. Just as in the definition of the \text{member} predicate, the \text{diff} predicate can be used in the definition of the \text{remove} predicate.

Note that the \text{insert} predicate is supplied as input to the program, depending nondeterministically on the \text{inUse} predicate (the identifier of the process that is using the resource at a particular moment in time must not be a member of the insertion list at that moment.) The \text{inUse} predicate is regarded as the output of the program; we assume that processes are aware of its values and take them as instructions to be obeyed. In particular, the process \text{Idno} uses (has access to) the resource at a given moment in time if and only if the atom \text{inUse}(\text{Idno}) is true at that moment.

## 4 Model-Theoretic Semantics

The declarative semantics of temporal logic programs are given in terms of the minimum-model semantics based on temporal Herbrand interpretations; see [29, 31] for Chronolog, and [8, 9] for Templog. However, choice-predicate extensions of these languages do not enjoy the minimum-model semantics because of the implicit choice formulas. In this section, we develop a minimal-model semantics for temporal logic programs with choice predicates from a language-independent perspective. Any temporal logic programming language can be used as the base language provided that its underlying logic conforms to the semantics scheme given earlier (except for the semantics of temporal operators) and programs written in the language enjoy the minimum-model semantics. Chronolog, restricted InTense, and Templog are some of the candidates for the choice of the base language.

### 4.1 Temporal Herbrand Interpretations

Given a temporal logic program \( P \) (in the base language), the temporal Herbrand interpretations of \( P \) can be characterized as follows. Let \( B_P \) be the set of all ground atomic formulas that can be
constructed from the predicate and function symbols appearing in $P$. $B_P$ is called the temporal Herbrand base of the program. The domain $D$ of temporal Herbrand interpretations of $P$ is the Herbrand universe $U_P$ constructed out of function symbols and constants appearing in $P$. A temporal Herbrand interpretation $I$ of $P$ assigns subsets of the set $\omega$ of moments in time to ground atoms in $B_P$. In other words, we have $I \in [B_P \rightarrow \mathcal{P}(\omega)]$.

It is not difficult to see that this formulation of temporal Herbrand interpretations conforms to the semantics scheme given earlier. Let $\mathcal{F}(P) \equiv [B_P \rightarrow \mathcal{P}(\omega)]$ be the set of temporal Herbrand interpretations of $P$. Then, for any given $I \in \mathcal{F}(P)$, there is a corresponding temporal interpretation $H$ with $U_P$ as the domain. In fact, the correspondence is established as follows. Let $\text{Pred}$ be the set of predicate symbols appearing in $P$, and $\mathcal{P}$ the set of predicate models a dataflow node producing the stream of odd numbers $\langle 0, 1, 3, 5, \ldots \rangle$ at time 0, $\langle 1, 2, 4, 6, \ldots \rangle$ at time 1, $\langle 2, 4, 6, 8, \ldots \rangle$ at time 2, $\langle 3, 6, 9, 12, \ldots \rangle$ at time 3, and so on in another. In words, the stream predicate models a dataflow node producing the stream of odd numbers $\langle 0, 1, 3, 5, \ldots \rangle$ with respect to the former model; and the stream of even numbers $\langle 0, 2, 4, 6, \ldots \rangle$ with respect to the latter. When the model $\cap$-intersection operation is applied to two such models, the stream predicate is true of 0 at time 0, and is not true of any term at any other moment in time; and similarly for the #values predicate because it represents the same relation as stream. But at time 1, the values predicate is assigned the relation $\{\langle n \rangle \mid n > 0\}$ in the model $\cap$-intersection of the two models, not a single-valued relation. This implies that the choice formulas for stream and values will not be true at all moments in time. Therefore we conclude that the model $\cap$-intersection operation does
not preserve modelhood for arbitrary sets of models. This negative result has serious implications on the model-theoretic semantics of extended programs.

Since the set of temporal Herbrand models of an extended program $P$ is not closed under the model $\cap$-intersection operation, extended programs do not enjoy the minimum-model semantics. However, the set $M(P)$ is not a totally unstructured collection. Indeed, we prove an important result which states that the set $M(P)$ is closed under the model $\cap$-intersection operation of nonempty downwards chains. Then we can appeal to a fundamental result from set theory, known as Zorn’s lemma, to show that the set $M(P)$ contains minimal models, each of which characterizes the declarative semantics of extended programs.

4.2 Choice Formulas, Extended Programs

Let TL be the underlying logic of the base language. We differentiate the choice predicates from the others by the following. Let $\text{Pred}$ denote the set of all predicate symbols of TL, other than the choice predicates. Then the choice predicate related to any $p \in \text{Pred}$ is denoted by $\#_p$ and can be used only in the bodies of program clauses in temporal logic programs, and in goals. In the following, choice formulas are defined for all $p \in \text{Pred}$ with non-zero arities, not just unary predicates. Therefore we can specify single-valued $n$-ary relations as well.

We now define non-Horn axioms for establishing the relationship between predicates and choice predicates. Any temporal logic program with choice predicates implicitly includes these axioms.

**Definition 5 (Choice formulas).** Let $p \in \text{Pred}$ be an $n$-ary predicate symbol (for $n > 0$). We define $\Psi_p$ as the set of choice formulas associated with $p$ as follows.

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

We also refer to these three formulas, in the given order, as $F_1$, $F_2$ and $F_3$. Read choice formulas as assertions true at all moments in time.

Intuitively, the choice formulas for any $p \in \text{Pred}$ together state that at any moment $t$ in time, 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 a temporal logic program $P$ extended with choice formulas given above, is called an “extended (temporal logic) program”. The formal definition is given below.

**Definition 6 (Extended programs).** Let $P$ be a temporal logic program in which choice predicates are used in the bodies of program clauses. Let $\Psi_p = \bigcup_{p \in \text{Pred}} \Psi_p$. Then $P \cup \Psi$ is called an extended (temporal logic) program.

The temporal Herbrand base $B_P$ of an extended program $P$ includes all those ground atomic formulas constructed out of choice predicates and ground terms in the Herbrand universe $U_P$ of $P$ as well as those atomic formulas for non-choice predicates. Recall that choice predicates are not defined by program clauses in $P$.

We treat the equality predicate symbol $\doteq$ in a different way and restrict ourselves to those interpretations in which the denotation of $\doteq$ satisfies the following condition: Let $P$ be an extended program and $I$ a temporal Herbrand interpretation of $P$. Then we have that for all $t \in \omega$, $[\doteq]_I^t = \{ \langle e, e \rangle \mid e \in U_P \}$. In other words, the denotation of $\doteq$ is an invariant of moments in time. To simplify the following presentation, we ignore those ground atoms related to the equality relation in the temporal Herbrand base of any extended program. Therefore we still consider temporal Herbrand interpretations of $P$ as elements of $\mathcal{F}(P)$.

We now formulate three model-theoretical conditions for temporal Herbrand interpretations of an extended program to be a model. Naturally, these model-theoretical conditions are the semantic counterparts of the choice formulas given above.
Definition 7. Let $P$ be an extended program and $I$ a temporal Herbrand interpretation of $P$. Let $p$ be an $n$-ary predicate symbol ($p \in \text{Pred}$). We define three model-theoretic conditions on $I$ as follows:

C1. $[p(\bar{c})]^I \subseteq [p(\bar{c})]^I$ for all $\bar{c} \in (U_P)^n$.

C2. for all $\bar{c}$ and $\bar{s} \in (U_P)^n$, if $\bar{c} \neq \bar{s}$, then $[p(\bar{c})]^I \cap [p(\bar{s})]^I = \emptyset$.

C3. $\bigcup_{\bar{c} \in (U_P)^n} [p(\bar{c})]^I = \bigcup_{\bar{c} \in (U_P)^n} [p(\bar{c})]^I$.

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 a temporal Herbrand interpretation to be a model of the choice formulas.

Lemma 1. Let $P$ be an extended program, $\Psi$ the set of choice formulas for $P$, and $I \in \mathcal{F}(P)$. Then $\models_I \Psi$ if and only if $I$ satisfies C1, C2 and C3 for all predicate symbols $p \in \text{Pred}$.

Proof. Since the model-theoretical conditions are meant to be 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 if and only if $\models_I F_1$ and $\models_I F_3$ for all $p \in \text{Pred}$. We give the detailed proof of the F2–C2 pair.

Suppose that $I$ satisfies C2, but it is not a model of F2 for some $p \in \text{Pred}$. Then for some $\bar{c}$ and $\bar{s} \in (U_P)^n$ where $\bar{c} \neq \bar{s}$, F2 is false at some $t \in \omega$. This implies that $t \in [p(\bar{c})]^I$ and $t \in [p(\bar{s})]^I$, which results in the contradiction $t \in [p(\bar{c})]^I \cap [p(\bar{s})]^I$. Thus $I$ must be a model of F2. Conversely, suppose that $I$ is a model of F2, but it does not satisfy C2. Pick any $\bar{c}$ and $\bar{s} \in (U_P)^n$ where $\bar{c} \neq \bar{s}$ and $[p(\bar{c})]^I \cap [p(\bar{s})]^I = \emptyset$. At any $t$ chosen from the intersection, the following instance of F2 is false in $I$: $\#p(\bar{c}) \rightarrow (p(\bar{s}) \rightarrow \bar{c} \equiv \bar{s})$. Thus $I$ must satisfy C2. \hfill $\square$

4.3 Minimal Models

We now show that, given an extended program $P$, $\mathcal{M}(P)$ is closed under the model $\cap$-intersection of downwards chains and thus, by the dual of the well-known Zorn’s lemma, it contains minimal elements. The temporal Herbrand interpretation that corresponds to the entire Herbrand base of an ordinary temporal logic program—namely the one that assigns $\omega$ to every atom in $B_P$—is in general not a model of $P$, because it may fail to satisfy some of the choice formulas whenever a predicate is true of more than one ground term at a moment in time. In turn, we lose the model existence property that says the top element in $\mathcal{F}(P)$ is a model of the program. The first question is whether an extended logic program $P$ is consistent (has models) or not. The following lemma gives an affirmative answer.

Lemma 2. Let $P$ be an extended program. Then $P$ has a model, that is, $\models_I P$ for some temporal Herbrand interpretation $I \in \mathcal{F}(P)$.

Proof. We construct a temporal Herbrand model $I$ of $P$ as follows: at all moments in time, all the ordinary predicates will be true of every term, whereas choice predicates will be true of an arbitrarily chosen term. Let $p \in \text{Pred}$ be an $n$-ary predicate symbol and $\bar{s}$ an arbitrary element of $(U_P)^n$. Let $I$ be a temporal Herbrand interpretation of $P$ satisfying the condition that $[p(\bar{c})]^I = \omega$ for all $p(\bar{c}) \in B_P$, and $[p(\bar{s})]^I = \omega$ for $p(\bar{s}) \in B_P$. It can be verified that $I$ is a model of the program clauses in $P$, because choice predicates do not appear in the heads of program clauses. In order to show that all choice formulas in $P$ are also true in $I$, we need to establish that $I$ satisfies all of C1, C2, and C3. We omit the details. \hfill $\square$

If we have a nonempty downwards chain of temporal Herbrand models of an extended program $P$, it suffices to show that the model $\cap$-intersection operation over such a chain preserves the necessary and sufficient conditions (C1, C2 and C3) for a temporal Herbrand interpretation of $P$ to be a model.
Lemma 3. Let $P$ be an extended program and $M = \langle I_\alpha \rangle_{\alpha \in S}$ be a downwards nonempty chain of temporal Herbrand models of $P$, ordered by $\sqsubseteq$. Then $\sqcap M \triangleq \sqcap_{\alpha \in S} I_\alpha$ is also a model of $P$.

Proof. We know that the model $\sqcap$-intersection operation preserves modelhood for temporal program clauses [8, 9, 29, 31]. It can then be shown that over a downwards chain of models of $P$, the model $\sqcap$-intersection operation preserves all of C1, C2, and C3. We omit the details. \qed

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 4 (The dual of Zorn’s lemma [3]). Let $X$ be a nonempty family of sets which is closed under intersections of nonempty downwards chains; then $X$ has a minimal element.

We now have the following result of the theory of extended programs:

Theorem 1. Let $P$ be an extended program and $M(P)$ be the set of all temporal Herbrand models of $P$. Then $M(P)$ has a minimal element.

Proof. We know that $M(P)$ is nonempty by Lemma 2. Then the theorem follows from Lemmas 3 and 4. \qed

In short, minimal models constitute the declarative semantics of extended programs. As mentioned earlier, any implementation of TLP that supports choice predicates must construct one such minimal model of an extended program. Again, this is how the correctness of the implementation is defined.

5 Canonical Models

It can be shown that any proof procedure that uses program clauses to prove goals and treats choice formulas as integrity constraints is not going to be able to prove some goals even when they are true of the program with respect to a minimal model. We first observe that the logical structure of an extended program dictates which minimal temporal Herbrand models an implementation can construct. Consider the following program (call it $P$),

\[
p(a).
\]
\[
p(b) \leftarrow \#p(a).
\]

and the goal $\leftarrow \#p(b)$. In order to prove the goal from $P$, we use the second clause and obtain the new goal $\leftarrow \#p(a)$. This means that, in order to prove $\#p(b)$, we must prove $\#p(a)$, which in turn implies that both $\#p(a)$ and $\#p(b)$ must be true at the same time. Since choice predicates represent single-valued relations, we have a contradiction (choice formulas of $p$ will fail). Therefore there is no proof of the original goal at any moment in time, and yet it is true in some minimal model, for example, the model that assigns $\omega$ to $p(a)$, $p(b)$ and $\#p(a)$, and $\emptyset$ to $\#p(b)$. In this model, the truth of $\#p(b)$ is not implied by any of the program clauses, but by program clauses together with choice formulas. Since choice formulas are implicit and they are not observed by the user, this model may not be considered as the “intended meaning” of the program.

Consider the goal $\leftarrow \#p(X)$. If the first clause is used to prove the goal, the answer substitution is \{X/a\}, and we are done. If the second clause is used, we run into the same problem discussed above. Therefore we conclude that the answer to the goal at any moment in time is the ground atom $\#p(a)$ or the answer substitution \{X/a\}. The structure of $P$ forces us to commit to choosing $a$ for $\#p$ before $p(b)$ can be proved. Therefore it is possible that any implementation can only construct exactly one and the same minimal model, that is, the one that assigns $\omega$ to all of $p(a)$, $p(b)$, and $\#p(a)$; and $\emptyset$ to $\#p(b)$. There are surely many minimal models of $P$, but only one of them is “constructible” in this regard.
In the following, we give a characterization of constructible minimal models by modifying and extending the fixed-point semantics of temporal logic programs. The standard fixed-point semantics [25] is a bottom-up approximation method to the meaning of a given logic program, and thus it is closely related to the operational semantics. Minimal models that are obtained as the limits of alternating chains of models using two new mappings (see below) are called constructible or canonical models. Since there may be more than one canonical model of a given extended program \( P \), any of its canonical models can be regarded as its intended meaning.

### 5.1 The Mappings \( \mathcal{N}T_P \) and \( C_P \)

The fixed-point semantics of temporal logic programs are given as an extension of the fixed-point semantics of ordinary logic programs [7, 25, 42]. Here we assume a standard definition for the one-step modus-ponens operation \( T_P \) for temporal logic programs; for example, see Orgun and Wadge [29, 31] for Chronolog and Baudinet [8, 9] for Templog. For our purposes in this paper, we do not need to know what \( T_P \) does to a given temporal Herbrand interpretation, but only its semantic properties. We know that such a \( T_P \) operation is continuous. The least fixed-point semantics also tells us that the minimum model of \( P \) is constructible bottom-up, by iterative applications of \( T_P \) starting from the empty model. However, since there may be more than one minimal model of a given extended program, the least fixed point of \( T_P \) alone is not sufficient to characterize the operational semantics of the program.

In the following, we first define the mapping \( \mathcal{N}T_P \) (read as "new \( T_P \") as the one-step modus-ponens operation for program clauses in an extended program. Let \( T_P \) be the mapping available from the base language.

**Definition 8.** Let \( P \) be an extended program and \( I \) a temporal Herbrand interpretation of \( P \). Let \( \mathcal{N}T_P \in \mathcal{F}(P) \to \mathcal{F}(P) \), where \( \mathcal{N}T_P(I) \) satisfies the conditions

\[
\left[ \#p(\vec{e}) \right]^{\mathcal{N}T_P(I)} = \left[ \#p(\vec{e}) \right]^I \quad \text{for all } \#p(\vec{e}) \in B_P
\]

and

\[
\left[ p(\vec{e}) \right]^{\mathcal{N}T_P(I)} = \left[ p(\vec{e}) \right]^I \quad \text{for all } p(\vec{e}) \in B_P.
\]

By definition, \( \mathcal{N}T_P \) leaves the denotations of choice predicates intact. The mapping \( \mathcal{N}T_P \) shares most of the properties of \( T_P \) for ordinary temporal logic programs; for example, it can be shown that \( \mathcal{N}T_P \) is continuous. For a given temporal logic program \( P \), it is also shown that a temporal Herbrand interpretation is a model of \( P \) if and only if \( T_P(I) \subseteq I \) [8, 9, 29, 31]. However, the models of an extended program \( P \) cannot be characterized in terms of \( \mathcal{N}T_P \) in a similar fashion. We can only show that a temporal Herbrand interpretation \( I \) is a model of the program clauses in \( P \) (not necessarily a model of \( P \)) if and only if \( \mathcal{N}T_P(I) \subseteq I \).

**Lemma 5.** Let \( P \) be an extended logic program, \( P_C \) the set of program clauses in \( P \), and \( I \in \mathcal{F}(P) \). Then \( \models_I P_C \) if and only if \( \mathcal{N}T_P(I) \subseteq I \).

**Proof.** Once we observe that \( \mathcal{N}T_P \) does not affect choice predicates, the proof of the lemma is similar to that of temporal logic programs; how it is done depends on the choice for the base language.

In order to formalize the nondeterminism introduced by choice predicates, we define another mapping, which we call \( C_P \), as follows. Let \( I \) be a temporal Herbrand interpretation of an extended program \( P \). For all \( p \in \text{Pred} \) and for all \( t \in \omega \), define

\[
E_{p,t}(I) = \{ \vec{e} \mid t \in [p(\vec{e})]^I \},
\]

\[
E_{\#p,t}(I) = \{ \vec{e} \mid t \in [\#p(\vec{e})]^I \}.
\]

In other words, \( E_{p,t}(I) \) denotes the set of ground terms that \( p \) is true of at time \( t \); and similarly \( E_{\#p,t}(I) \) denotes the set of ground terms that \( \#p \) is true of at time \( t \). Let \( S = \{(p,t) \mid E_{p,t}(I) \neq \emptyset \} \)
\( \emptyset \) and \( E_{\#p,t}(I) = \emptyset \), i.e., for any \((p,t) \in S\), no choice has been made for \( p \) at time \( t \). Let \( \prod \) denote the Cartesian product operation. Define \( \text{Cset}(I) = \prod_{(p,t) \in S} E_{p,t}(I) \). Any member of \( \text{Cset}(I) \), the choice set of \( I \), represents a choice function which given any \((p,t) \in S\), returns an arbitrary term chosen from \( E_{p,t} \). The formal definition of the \( \text{C}_P \) operation is given below.

**Definition 9.** Let \( P \) be an extended program and \( I \in \mathcal{F}(P) \). Let \( \text{C}_P \in \mathcal{F}(P) \rightarrow \mathcal{P}(\mathcal{F}(P)) \) where, for all \( \alpha \in \text{Cset}(I) \), \( I_\alpha \in \mathcal{C}_P(I) \) if and only if for all \( p \in \text{Pred} \), and for all \( \vec{e} \in (U_P)^n \),

\[
[p(\vec{e})]^{I_\alpha} = [p(\vec{e})]^I
\]

and

\[
[\#p(\vec{e})]^{I_\alpha} = [\#p(\vec{e})]^I \cup \{ t \mid \vec{e} = \alpha(p,t) \}.
\]

The \( \text{C}_P \) operation returns all possible immediate extensions of a given temporal Herbrand interpretation of \( P \), determined by arbitrary choice functions. The interpretations in \( \text{C}_P(I) \) for any given \( I \) are indexed by \( \alpha \)'s in \( \text{Cset}(I) \). By the definition of \( \text{C}_P \), given any \( I \in \mathcal{F}(P) \), we have that for all \( J \in \mathcal{C}_P(I) \), \( I \subseteq J \). The \( \text{C}_P \) operation does not lose any information.

The ordinal powers of \( \mathcal{N}^{TP} \) relative to any temporal Herbrand interpretation \( I \) of an extended program \( P \) are defined as follows: \((\mathcal{N}^{TP} \uparrow \omega)(I) \overset{\text{def}}{=} \bigcup_{n \in \omega}(\mathcal{N}^{TP} \uparrow n)(I)\), where \((\mathcal{N}^{TP} \uparrow 0)(I) = I\) and \((\mathcal{N}^{TP} \uparrow n+1)(I) = \mathcal{N}^{TP}(\mathcal{N}^{TP} \uparrow (n-1))(I)\) for \( n > 0 \). Let \( I_0 \) denote \( \cap \mathcal{F}(P) \), which is the greatest lower bound in \( \mathcal{F}(P) \). Observe that \( I_0 \) assigns the empty set \( \emptyset \) to all ground temporal atoms in \( B_P \). In other words, for all predicate symbols \( p \in \text{Pred} \) and for all \( \vec{e} \in (U_P)^n \),

\[
[p(\vec{e})]^{I_0} = [\#p(\vec{e})]^{I_0} = \emptyset.
\]

We have that \( I_0 \) is a model of choice formulas of \( P \).

**Lemma 6.** Let \( P \) be an extended program. Then \( I_0 \) is a model of choice formulas \( \bigcup_{p \in \text{Pred}} \Psi_p \).

**Proof.** \( I_0 \) trivially satisfies all of C1, C2 and C3 for all predicate symbols in \( \text{Pred} \) and hence is a model of choice formulas of \( P \) by Lemma 1. \( \square \)

### 5.2 Alternating Chains of Models

The mappings \( \mathcal{N}^{TP} \) and \( \text{C}_P \) interact nicely: if we start from \( I_0 \) and alternatively apply \( \mathcal{N}^{TP} \uparrow \omega \) and \( \text{C}_P \), we obtain an “alternating chain” of interpretations of \( P \). 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 \( \text{C}_P \) operations. The method of alternating chains is well-known in the theory of models, for example, see Addison [4].

We first show that the limits (upper bounds) of these alternating chains are in fact models of \( P \). In so doing, we use an important result from the theory of models known as Chang-Löś-Suszko theorem. The theorem applies to a class of formulas, called universal-existential formulas (\( \forall \exists \)-formulas) defined below.

**Definition 10.** A formula is universal-existential if it has the form

\[
(\forall X_0) \cdots (\forall X_{m-1})(\exists Y_0) \cdots (\exists Y_{n-1})A
\]

where \( A \) is quantifier-free.

Every program clause in a temporal 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 2 (Chang-Łoś-Suszko [12]). Let \( \Gamma \) be a set of formulas. Then \( \Gamma \) is equivalent to a set of \( \forall \exists \)-formulas if and only if the least upper bound of any upwards chain of models of \( \Gamma \) is a model of \( \Gamma \).

We now formalize the notion of alternating chains of interpretations of an extended program \( P \) originating from \( I_\emptyset \) as follows.

Definition 11. Let \( P \) be an extended program. A \( P \)-chain is a sequence \( S \) of temporal Herbrand interpretations of \( P \) satisfying the following conditions.

\begin{itemize}
  \item \( S_0 = I_\emptyset \),
  \item \( S_{2n+1} = (NT_P \uparrow \omega)(S_{2n}), \) for \( n \geq 0 \),
  \item \( S_{2n+2} \in C_P(S_{2n+1}), \) for \( n \geq 0 \).
\end{itemize}

For any given extended program \( P \), there are many \( P \)-chains because of the branching introduced by the \( C_P \) operation. The following lemmas show that in any given \( P \)-chain, all interpretations obtained by \( C_P \) operations are models of the choice formulas, and all interpretations obtained by \( NT_P \uparrow \omega \) operations are models of the program clauses in \( P \).

Lemma 7. Let \( P \) be an extended program, \( \Psi \) the set of choice formulas for \( P \), and \( S \) a \( P \)-chain. Given \( \langle I_n, I_{n+1} \rangle \in S \), where \( I_{n+1} \in C_P(I_n) \), we have that \( \models I_{n+1} \Psi \).

Proof. Since \( I_n \) and \( I_{n+1} \) are both in a \( P \)-chain, only \( C_P \) operations can affect the denotations of choice predicates. We know by Lemma 6 that \( I_\emptyset \) (the seed of \( P \)-chains) is a model of choice formulas of \( P \). Then the definition of \( C_P \) guarantees that no arbitrary terms are introduced in the denotations of choice predicates by \( C_P \) operations and all of \( C1, C2 \) and \( C3 \) are therefore satisfied. Thus \( \models I_{n+1} \Psi \) by Lemma 1.

Lemma 8. Let \( P \) be an extended program, \( P_C \) the set of program clauses in \( P \), and \( S \) a \( P \)-chain. Given \( \langle I_n, I_{n+1} \rangle \in S \), where \( I_{n+1} = (NT_P \uparrow \omega)(I_n) \), we have that \( \models I_{n+1} P_C \).

Proof. We outline the proof. For all \( t \in \omega \) and \( p(\bar{e}) \in B_P \), that \( t \in [p(\bar{e})]^{I_n} \) is implied by a ground instance of some clause in \( P \), because both \( I_n \) and \( I_{n+1} \) are in a \( P \)-chain starting from \( I_\emptyset \). Therefore the \( NT_P \uparrow \omega \) operation will not remove \( t \) from the denotation of \( p(\bar{e}) \). As choice predicates do not appear in the heads of program clauses, their denotations are solely determined by \( C_P \) operations. Then it can be shown that \( \langle (NT_P \uparrow k)(I_n) \rangle_{k \in \omega} \) is also a chain whose least upper bound is \( (NT_P \uparrow \omega)(I_n) \). Since \( NT_P \) is continuous, we have that \( NT_P((NT_P \uparrow \omega)(I_n)) \subseteq (NT_P \uparrow \omega)(I_n) \) by a version of a result given for stratified logic programs by Apt et al [6]. Therefore \( (NT_P \uparrow \omega)(I_n) \) is a model of \( P_C \) by Lemma 5.

We now show that the least upper bounds of \( P \)-chains are models of extended programs. In other words, if we start from \( I_\emptyset \), then alternating applications of \( NT_P \uparrow \omega \) and \( C_P \) will eventually produce a model of \( P \).

Theorem 3. Let \( P \) be an extended program, \( S \) a \( P \)-chain, and \( \sqcup S = \sqcup n \in \omega I_n \) the least upper bound of \( S \). Then \( \models \sqcup S P \).

Proof. By construction, \( \sqcup S \) is the common least upper bound of two interleaving chains in \( S \), namely, \( \langle S_{2n} \mid n \in \omega \rangle \) and \( \langle S_{2n+1} \mid n \in \omega \rangle \). In other words, we have that \( \sqcup S = \sqcup n \in \omega S_{2n} = \sqcup n \in \omega S_{2n+1} \). Since \( P \) is a set of universal-existential formulas, by the Chang-Łoś-Suszko theorem (2), we have that \( \sqcup S \) is a model of both program clauses and choice formulas. Hence \( \models \sqcup S P \).

The following corollary is a stronger result than Theorem 3. It states that, when we start from \( I_\emptyset \), alternating applications of \( NT_P \uparrow \omega \) and \( C_P \) lead to a minimal model. We call such minimal models canonical models.
Corollary 1. Let \( P \) be an extended program, \( S \) a \( P \)-chain, and \( \sqcup S = \sqcup_{1 \leq n \leq \omega} I_n \) be the least upper bound of \( S \). Then \( \sqcup S \) is minimal.

Proof. By Theorem 3, \( \sqcup S \) is a model of \( P \). Suppose that it is not minimal. Since the minimality condition concerns only non-choice predicates, we consider only those ground atoms related to any ground instance of any clause in \( P \). By Theorem 3,\footnote{Theorem 3} \( \sqcup S \) is a model of \( P \). Suppose that \( t \in [p(\vec{e})]^{\sqcup S} \), but this is not implied by any ground instance of any clause in \( P \). Then it must be the case that for some \( I_n \in S, t \in [p(\vec{e})]^{I_n} \). But for some \( k \in \omega \), we must have that \( I_n \subseteq I_{n+k} \subseteq (NTP \uparrow \omega)(I_{n+k}) \). This entails that \( t \in [p(\vec{e})]^{I_{n+k}} \) but \( t \not\in [p(\vec{e})]^{(NTP \uparrow \omega)(I_{n+k})} \), because the \( NTP \) operation does not keep \( t \) in the intension of \( p(\vec{e}) \) if it is not implied by any program clause. This leads to the contradiction that \( S \) cannot be a chain. Thus \( \sqcup S \) must be minimal. \( \square \)

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 canonical model. Then it is possible that any implementation can only construct that canonical model.

Definition 12. Let \( P \) be an extended program. We say that \( P \) is operationally deterministic if and only if \( \text{card}(M) = 1 \), where \( M = \{ \sqcup S \mid S \text{ is a } P\text{-chain} \} \) and \( \text{card}(X) \) is the cardinality of the set \( X \).

The program given above that defines \( p \) is an example of an operationally deterministic program. Its logical structure singles out the canonical model in which the ground atom \( #p(a) \) is assigned to \( \omega \). The \( NTP \) operation on \( I_0 \) leads us to the interpretation in which \( p(a) \) gets \( \omega \); the \( CP \) operation on that interpretation forces the choice for \( #p \) by assigning \( \omega \) to \( #p(a) \); finally, another \( NTP \) operation results in \( p(b) \) being assigned to \( \omega \), and we are done. We may consider that canonical model as the intended meaning of the program. Perhaps the programmer has this model in mind when writing the program. Programs that define single-valued relations (such as the Fibonacci example) are also candidates to be considered as operationally deterministic programs. The properties of such programs need to be investigated.

6 A Proof Procedure

In [33], it is shown that Chronolog with an unbounded past and future admits a sound and complete proof procedure called TiSLD-resolution. TiSLD-resolution (read as a Timely SLD-resolution) is an extension of the refutation procedure called SLD-resolution [7]. In this section, we first summarize the axioms and rules of inference of the temporal logic TL of Chronolog, and discuss TiSLD-resolution with the restriction to TL. We extend TiSLD-resolution with memory to keep track of choices made in a given TiSLD-derivation to guarantee the correctness of computed answers under the presence of choice formulas.

6.1 Axioms, Rules of Inference

Let the notation \( \vdash A \) denote the fact that \( A \) is a theorem of TL. The notation \( \Gamma \vdash A \) means that \( A \) is deducible from a set \( \Gamma \) of formulas in TL. The following axioms (theorems) state some of the important properties of the temporal operators. Let \( \nabla \) stand for either of \text{first} and \text{next}.

1. Temporal operator cancellation rule.
   \[ \text{E. } \nabla(\text{first } A) \leftrightarrow \text{first } A. \]

   The axiom E states that initial truths persist, an initial truth being any formula within the scope of a \text{first} operator.

2. Temporal operator distribution rules.
   \[ \text{D1. } \nabla(A \land B) \leftrightarrow (\nabla A) \land (\nabla B). \]
\textbf{D2.} \(\nabla(\neg A) \leftrightarrow \neg(\nabla A)\).

These axioms state that the temporal operators commute with the Boolean operators \(\land\) and \(\neg\) (and also with the defined operators \(\lor\), \(\rightarrow\) and \(\leftrightarrow\)). The axioms also capture the fact that the Boolean operators work pointwise in time.

3. Rigidness of variables.

\textbf{V.} \(\nabla(\forall x)(A) \leftrightarrow (\forall x)(\nabla A)\).

This axiom stipulates that the values of individual variables range over extensions (data values), not intensions (time-varying values). It is an instance of the so-called Barcan formula combined with its converse [23].


\textbf{R1.} If \(\vdash A\), then \(\vdash \text{first } A\).

\textbf{R2.} If \(\vdash A\), then \(\vdash \text{next } A\).

These rules are instances of the rule of generalization from temporal logic [35].

The presentation of an axiomatic system for TL begs the question whether it is complete with respect to the given semantic scheme. However, this is beyond the scope of this paper. The correctness (soundness) of the axioms and the rules of inference is also established.

\textbf{Lemma 9 ([33]).} \textit{All of the axioms and the rules of inference given above are valid with respect to the semantics scheme of TL.}

We assume that the rules of inference given above are extended to consider the notion of deducibility from a set of formulas. For instance, now R1 reads “if \(\vdash A\), then \(\vdash \text{first } A\)”. The formal properties of the deducibility relation are not given in this paper. However, it should be kept in mind that, under the presence of the rules of inference for the temporal operators, a form of a deduction theorem does not hold for \(\vdash\). If it did, given \(A \vdash A\), we could derive \(A \vdash \text{first } A\) by rule R1 and then \(\vdash A \rightarrow \text{first } A\) by the deduction theorem. But it can be shown that \(A \rightarrow \text{first } A\) is not a valid formula of TL.

We also need the notion of a “rigid formula”. By the temporal-operator introduction rules, a sequence of initial truths can be obtained from any given formula. For instance, given a formula \(A\) of TL, by systematic applications of R1 and R2, the following initial truths can be formed:

\[
\begin{align*}
\text{first } A & \quad \text{(by R1)}, \\
\text{first next } A & \quad \text{(by R2, R1)}, \\
\text{first next next } A & \quad \text{(by R2, R2, R1)}
\end{align*}
\]

and so on. The value of any of these formulas is an invariant of time. Initial truths obtained from a formula \(A\) are called “rigid instances” of \(A\).

6.2 TiSLD-Resolution

TiSLD-resolution is applied to a set of rigid program clauses and goal clauses. For a given temporal logic program \(P\), rigid instances of program clauses are obtained by the rules R1 and R2. We also assume that in rigid instances of program clauses, superfluous applications of temporal operators are eliminated by using the axioms. By Lemma 9, \(P \vdash C\) for every rigid instance \(C\) of any program clause in \(P\). In the following, we assume familiarity with the concepts of substitution, unification and refutation procedures; see Lloyd [25] for a detailed discussion.

Given a Chronolog program \(P\) and a goal \(G\), a TiSLD-derivation of \(P \cup \{G\}\) consists of a sequence \(G_0, G_1, \ldots\) of goal clauses, where \(G_0 = G\), a sequence \(C_0, C_1, \ldots\) of variants (up to renaming) of rigid instances of program clauses in \(P\) and a sequence \(\theta_0, \theta_1, \ldots\) of substitutions.
At every step of a TiSLD-derivation, some rigid temporal atom from the current goal is selected and it is unified with the conclusion (head) of a rigid instance of a program clause after renaming of the variables in the rigid instance. A new goal is produced by replacing the selected temporal atom in the goal by the premise (body) of the rigid instance and then the substitution (mgu) obtained from the unification process is applied to the new goal.

Consider the Fibonacci program given earlier and the goal $← \textit{first next next fib}(N)$ (call it $G_0$). The following are the steps in a TiSLD-refutation of the program and the goal. The only temporal atom in the goal $G_0$ is selected and then matched with the head of a rigid instance of the third program clause by unification after renaming the variable $N$ in the clause.

$$G_0 = ← \textit{first next next fib}(N),$$

$$C_0 = \textit{first next next fib}(N') ← \textit{first fib}(X), \textit{first fib}(Y), \textit{first} N' \text{ is } X + Y \quad \text{(third program clause, by R1 and axioms)}$$

$$\theta_0 = \{N/N'\}.$$ 

Then a new goal is produced after replacing the selected temporal atom in $G_0$ by the body of $C_0$. The substitution $\theta_0$ is applied to the new goal. Any of the three temporal atoms can be selected for the next step in the derivation; we select the first one. The temporal atom $\textit{first next fib}(X)$ in $G_1$ is matched with the second program clause in the program.

$$G_1 = (← \textit{first next fib}(X), \textit{first fib}(Y), \textit{first} N' \text{ is } X + Y)\theta_0$$

$$C_1 = \textit{first next fib}(1) \quad \text{(second program clause)},$$

$$\theta_1 = \{X/1\}.$$ 

In the new goal $G_2$, the selected temporal atom from $G_1$ is removed. The substitution $\theta_1$ is applied to the new goal. We select the first temporal atom in the goal and match it with the first program clause.

$$G_2 = (← \textit{first fib}(Y), \textit{first} N' \text{ is } X + Y)\theta_1$$

$$C_2 = \textit{first fib}(0) \quad \text{(first program clause)},$$

$$\theta_2 = \{Y/0\}.$$ 

When the selected temporal atom in $G_2$ is removed, there is only the arithmetic predicate left in the goal. This predicate can be formalized as a ternary addition predicate, so that $N'$ may be unified with the result of the addition operation.

$$G_3 = (← \textit{first} N' \text{ is } 1 + Y)\theta_2$$

$$C_3 = \textit{first add}(1, 0, 1) \quad \text{(addition relation)},$$

$$\theta_3 = \{N'/1\}.$$ 

The new goal $G_4$ is the empty goal, meaning that the derivation is successful. A successful TiSLD-derivation is called a TiSLD-refutation. The composition of mgu’s $\theta_0$, $\theta_1$, $\theta_2$ and $\theta_3$ is regarded as a computed answer substitution for the original goal; indeed, a correct one. In the answer to the original goal, $N$ is substituted by 1. It follows that $\textit{first next next fib}(1)$ is a logical consequence of the Fibonacci program.

### 6.3 Revised TiSLD-resolution

TiSLD-resolution does not apply to Chronolog with choice predicates. There is no provision made in TiSLD-resolution to keep track of the choices made in a given TiSLD-derivation. If the same choice predicate appears more than once at different stages of a derivation, it must be guaranteed that only one choice is made for each moment in time to satisfy choice formulas as integrity
constraints. Otherwise, inconsistencies may arise and we can no longer trust the outcome of the derivation.

For instance, suppose that we are given an extended program in which the \( p \) predicate is true of the terms \( a \) and \( b \) at all moments in time. We cannot allow the temporal atoms first next \( \#p(a) \) and first next \( \#p(b) \) to appear in the same derivation. Otherwise, some of the choice formulas of \( p \) will not be satisfied. In order to guarantee that the choice for \( \#p \) is unique at any given moment in time, we impose extra conditions on TiSLD-derivations to ensure that choices are made consistently. The resulting proof procedure will be referred to as “revised TiSLD-resolution”.

Given a Chronolog program \( P \) with choice predicates and a goal \( G \), a revised TiSLD-derivation of \( P \cup \{G\} \) now includes a non-decreasing sequence \( S_0, S_1, \ldots \) of sets of rigid temporal atoms in which only choice predicates appear, as well as sequences of goals, program clauses, and substitutions as before. \( S_0 \) is always the empty set. \( S_i \) at the \( i \)-th step of the derivation contains all the temporal atoms for choice predicates that are selected up to step \( i \). At every step of the revised TiSLD-derivation, some rigid temporal atom from the current goal is selected. Depending on whether the selected atom contains a choice predicate or not, there are two cases in which the derivation may proceed:

**Case 1.** When the selected temporal atom does not contain a choice predicate, the derivation proceeds much in the same way as defined earlier. In addition, we set \( S_{i+1} \) to be \( S_i \theta_i \), where \( \theta_i \) is the substitution from the unification process and \( S_i \theta_i \) is the set obtained by applying \( \theta_i \) to every member of \( S_i \).

**Case 2.** When the selected temporal atom contains a choice predicate, there are two subcases to consider. Let first next \( \#p(\vec{c}) \) be the selected temporal atom.

1. **If there are no temporal atoms in \( S_i \) of the form first next \( \#p(\vec{s}) \), the selected atom is unified with the conclusion (head) of some rigid instance of a program clause after renaming of the variables in the rigid instance. In other words, we do not differentiate between \( p \) and \( \#p \) and the definition of \( p \) is used to prove the selected atom. A new goal is produced by replacing the selected atom in the goal by the premise (body) of the rigid instance, and then the substitution (mgu) (say \( \theta_i \)) from the unification process is applied to the new goal. In addition, we set \( S_{i+1} \) to be \( S_i \theta_i \cup \{(\text{first next} \#p(\vec{c}))\theta_i\} \).

2. **If there is a temporal atom in \( S_i \) of the form first next \( \#p(\vec{s}) \), then \( \vec{c} \) and \( \vec{s} \) are pairwise unifiable for some substitution, say \( \theta_i \). A new goal is produced by applying the substitution \( \theta_i \) to the current goal after removing the selected atom from the goal. In addition, we set \( S_{i+1} \) to be \( S_i \theta_i \).

The sets \( S_i \)'s serve as memory in a given derivation. It is essential to keep track of all the choices made in a given derivation in \( S_i \)'s to avoid inconsistencies. At every step of the derivation, substitutions are applied to \( S_i \)'s to narrow down possible values for choice predicates at the following steps. The composition of the mgu's \( \theta_0, \theta_1, \ldots \) is still regarded as a computed answer substitution to the initial goal.

If there are still some variables left in the final answer, any term from the Herbrand universe of the program can be substituted for the variables. However, to ensure a correct interpretation of the answer, all those variables within the scope of a choice predicate must be quantified existentially. This is because different choices of terms will result in the answer being justified with respect to different (canonical) minimal models of the program. All those variables not occurring within the scope of a choice predicate are quantified universally.

In general, computed answers obtained from TiSLD-derivations that involve choice predicates are not necessarily logical consequences of the program (they may or may not be true in every model of the program due to the choice-predicates nondeterminism). For instance, given an extended program with the following two clauses:

\[
p(a).
\]
\[
p(b).
\]
and the goal $\leftarrow \#p(X)$, then neither $\#p(a)$ nor $\#p(b)$ as a computed answer is true in every model of the program at any given moment $t$ in time. This is because in all models of the program, only one of $\#p(a)$ and $\#p(b)$ will be true at time $t$, not both. In contrast, since the definition of $p$ does not involve any choice predicates, both of $p(a)$ and $p(b)$ are logical consequences of the program.

In short, the “soundness” of revised TiSLD-resolution can be established by showing that computed answers are justifiable relative to a canonical model of a given extended program $P$. This can be done by embedding a given TiSLD-derivation into an alternating chain of models of $P$. The soundness of revised TiSLD-resolution is therefore a weaker notion than that of TiSLD-resolution which says that computed answers are logical consequences of Chronolog programs [33]. In this paper, we do not elaborate on the soundness issues of revised TiSLD-resolution any further.

7 Conclusions

We have introduced choice predicates for modeling nondeterministic, non-terminating, dataflow computations in TLP. We have developed a model-theoretic semantics for temporal logic programs with choice predicates in terms of minimal models. The minimal-model semantics can be applied to the temporal languages Chronolog [29, 31, 37, 44], Templog [1, 2, 9], and the multidimensional intensional language InTense [26] when it is restricted to just a temporal dimension. A more general model-theoretic semantics framework is outlined in Orgun and Wadge [32], which can be applied to all the languages mentioned above. As Temporal Prolog [17] is based on a richer subset of temporal logic than those of Chronolog and Templog, the minimal-model semantics does not apply to a possible extension of Temporal Prolog with choice predicates.

Tempura [27] is also a temporal programming language based on interval logic. However, it is not based on the Horn subset of temporal logic, and its execution mechanism is not based on resolution-type proof procedures. In Tempura, programs are systematically transformed into a sequence of state descriptions over an interval that satisfies the original program. In Tempura, variables are intensional (they represent a sequence of values), whereas in Chronolog predicates are intensional (they represent a sequence of relations). In this respect, Tempura is closer to the dataflow language Lucid [45] than to Chronolog. Therefore it also offers a good model for modeling dataflow computations. However, Chronolog with choice predicates can also model nondeterministic dataflow computations, because it is based on logic programming which has inherent nondeterminism. It also avoids potential problems associated with dealing with (possibly infinite) sequences of values for each variable.

The language Tokio [5] is based on the same interval logic as Tempura, but it has features which go beyond logic programming. Therefore the minimal model semantics does not apply to a possible extension of Tokio with choice predicates. At this stage, we do not consider first-order treatments of temporal modalities in the logic programming framework, such as those of Hrycej [22] and Brzoska [10, 11], as candidates for the choice predicates extension.

We have given a characterization of constructible (canonical) minimal models using two new mappings: the $\text{NT}_{\text{P}}$ operator and the $\text{CP}$ operator. The $\text{NT}_{\text{P}}$ operator is based on the $\text{T}_{\text{P}}$ operator provided from the base language whereas the $\text{CP}$ operator is language-independent. Again, Chronolog, Templog, and restricted InTense are some of the languages with the desired properties; the relevant fixed-point results can be found in the literature [8, 9, 31, 32].

The proof procedure of Chronolog [33] is revised to allow for choice predicates. Since the standard fixed-point semantics [25] establishes the link between the model-theoretic and operational semantics of logic programs and the notion of a canonical model is partly based on an extended fixed-point semantics, we conjecture that any canonical model can be constructed by using TiSLD-resolution (i.e., temporal atoms true in the canonical model can be proved using TiSLD-resolution.) However, this is a form of a “completeness” result and it is a non-trivial research problem. Note that the proof procedure of Templog [1, 8] could also be extended to support choice predicates.

Wadge [44] suggested that Chronolog could be implemented by adapting an existing logic-programming implementation. Moments in time 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. This approach is in spirit with the labeled deduction proposed by Gabbay [18] for Temporal Prolog. Note that an implementation of Chronolog must rely on TSLDR-resolution for correctness. However, efficient implementations of Chronolog must combine features of logic-programming implementations (unification, backtracking) with features of dataflow implementations (tagging, associative memory [16]). Otherwise they would waste resources proving the same goals over and over again. Choice predicates should not complicate implementations too much. We need only keep a permanent record of choices made, as they are made. For correctness, the implementation should ensure that there is only one choice made for any predicate at any moment in time, so that choice formulas are satisfied as integrity constraints. Rolston [38] proposed an approach to the parallel execution of logic programs which is also capable of executing temporal languages such as Chronolog. His approach is partly based on the ideas discussed above.

Acknowledgements

The work of W. W. Wadge has been supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). Thanks are due to two anonymous referees for their helpful comments and suggestions on an earlier draft of this paper.

References


