J. Symbolic Computation (1996) **22**, 699–720

Dealing with Multiple Granularity of Time in Temporal Logic Programming

CHUCHANG LIU[†] AND MEHMET A. ORGUN[‡]

Department of Computing, Macquarie University, NSW 2109, Australia

(Received 31 October 1995)

Chronolog(MC) is an extension of logic programming based on a linear-time temporal logic with multiple granularity of time called *TLC*. A Chronolog(MC) program consists of a clock definition, a clock assignment and a program body. Each predicate symbol appearing in the program body is associated with a local clock through the clock definition and assignment. This paper investigates the logical basis of the language, presents a clocked temporal resolution where time-matching is essential, and in particular proposes three algorithms for time-matching. The paper also discusses the declarative semantics of Chronolog(MC) programs in terms of clocked temporal Herbrand models. It is shown that Chronolog(MC) programs also satisfy the minimum model semantics. The language can be used to model a wide range of simulation systems and other relevant tasks where the notion of dynamic change is central.

1. Introduction

An important activity in computer science is the invention, analysis and application of formal *logics* which are designed to specify, reason about and represent algorithms, programs and systems. Recently, there is a substantial interest in temporal logic which has been widely used as a formalism for program specification and verification (Manna and Pnueli, 1981), reasoning about time (Sadri, 1987) and modeling temporal databases (Chomicki, 1994; Gabbay and McBrien 1991; Gagné and Plaice, 1995).

Some researchers have recently suggested that temporal logic can be directly used as a programming language. For instance, Tokio (Aoyagi *et al.*, 1986) is a logic programming language based on interval temporal logic; Templog (Abadi and Manna, 1989; Baudinet, 1992) and Chronolog (Orgun and Wadge, 1992) are based on linear-time temporal logics; Temporal Prolog (Gabbay, 1987, 1991) is based on linear and branching time temporal logics; and METATEM (Fisher, 1994; Fisher and Reynolds, 1995) is a framework for the direct execution of temporal logics within which programs are represented as sets of temporal rules of a particular form. There are also a number of other temporal logic

[†] E-mail: `cliu@mpce.mq.edu.au`
[‡] E-mail: `mehmet@mpce.mq.edu.au`

programming languages (Brzoska, 1995; Hrycej, 1993; Frühwirth, 1995). For more information, we refer the reader to the extensive surveys of Fisher and Owens (1995), and Orgun and Ma (1994).

1.1. CHRONOLOG WITH MULTIPLE CLOCKS

Chronolog is suitable for specifying time-dependent properties of certain problems in a natural way (Orgun and Wadge, 1992; Liu and Orgun, 1995). It is based on a linear-time temporal logic (Goldblatt, 1987; Rescher and Urquhart, 1971) in which the collection of moments in time is modeled by the set of natural numbers with its usual ordering relation $<$. The temporal logic has two temporal operators, `first` and `next`. The intuitive meanings of these operators are as follows:

- `first` A: A is true at the initial moment in time,
- `next` A: A is true at the next moment in time.

In Chronolog, all predicates are actually considered to be defined on the global clock, i.e. the sequence of natural numbers $\langle 0, 1, 2, \ldots \rangle$. However, in some applications one may find that it is necessary to consider "local times". For instance, in analysis of distributed computations, complications arise when it is time to decide how a process is to perform its function and when it does an action. The processes involved in such a computation have their own local time. A process only "sees" local events, such as sending a message to and receiving a message from other processes through its buffers (in the case of asynchronous communication). That is, one process is not always defined at all the moments in time on the global clock. To describe such systems, it is more natural to introduce multiple granularity of time or provide local clocks to model the behavior of processes.

Granularity of time is also a key issue in temporal databases. In some applications, the granularity is days, in others it can be seconds or years, or a combination of multiple granularity. Wiederhold *et al.* (1991) also recognized the problem, and provided an algebra in which data with multiple granularities were converted to a uniform model of data based on time intervals. Such an approach requires interpolation of data with multiple granularities over intervals, using a history operator **H**, based on certain assumptions. Granularity of time is also important in ecological modeling (Mota *et al.*, 1995), where we may need to consider how a number of actions performed at different levels of time interact with each other.

We propose to extend Chronolog by introducing multiple clocks (Liu, 1995; Liu and Orgun, 1995) so that it can be used in applications such as above which require multiple granularity of time. The resulting language is called Chronolog with Multiple Clocks, or Chronolog(MC) for short. It is based on a linear-time temporal logic called *TLC* (for *Temporal Logic with Clocks*) in which there is a local clock associated with each predicate symbol. In *TLC*, each formula is also clocked, determined by the clocks of predicate symbols which appear in the formula. A Chronolog(MC) program consists of three parts—a clock definition, a clock assignment and a program body—and these parts can be viewed as independent Chronolog programs. Each predicate symbol appearing in the program body is associated with a local clock through the clock definition and assignment.

The clocked extension has brought an essential change on the meanings of the temporal operators of Chronolog. In the extension, the temporal operators `next` and `first` keep their intuitive meanings "the initial moment in time" and "the next moment in time" respectively, but these meanings depend on the actual clocks given.

Chronolog(MC) can be used in applications where the notion of dynamic change is central, and it can be applied to the specification of a wide range of simulation systems and other relevant tasks (Liu and Orgun, 1995). For instance, we can simulate a distributed computation with local clocks defined for all the processes involved in it, so that we can make analysis using scheduling theory to determine its timing properties in relation to its use of computational resources (Liu and Orgun, 1996).

1.2. STRUCTURE OF THE PAPER

In this paper, we study the logical basis of Chronolog(MC). In particular, we also propose a clocked temporal resolution as a refutation procedure for the execution of Chronolog(MC) programs. Orgun and Wadge (1993) have shown that Chronolog admits a sound and complete proof procedure, called TiSLD-resolution, and established the equivalence of the declarative and operational semantics of Chronolog programs. This paper discusses the operational semantics of Chronolog(MC) programs in terms of the new temporal resolution, and presents the declarative semantics for such programs in terms of clocked temporal Herbrand models by extending their results.

2. Temporal Logic *TLC*

In this section, we introduce the temporal logic *TLC*, which is a clocked extension of the underlying logic *TL* of the original Chronolog.

2.1. FORMULAE

Any formula of classical first-order logic is also a formula of *TLC*. In addition to the formation rules of first-order logic, we have a new formation rule to produce new formulae: any *TLC* formula A may be prefixed by a temporal operator to form a new formula of the form `first` A or `next` A.

A formula of *TLC* is also called a temporal formula. Particularly, if A is an atomic formula of first-order logic, then we say that A is a temporal atomic formula, or simply, an atom of *TLC*; and if A is an atom of *TLC*, so are `first` A and `next` A. We also say that an atom is *pure* if it does not contain any temporal operators.

2.2. CLOCKS

In *TLC*, the set ω of natural numbers models the collection of moments in time, and clocks are sequences over ω. Formally, we define that the global clock is the increasing sequence of all natural numbers: $\langle 0, 1, 2, \ldots \rangle$, and a local clock is a subsequence of the global clock. In other words, a local clock is a strictly increasing sequence of natural numbers, either infinite or finite: $\langle t_0, t_1, t_2, \ldots \rangle$. In particular, the empty sequence $\langle\rangle$ which does not contain any moments in time is also a local clock, called the empty clock. By definition, the global clock is also a local clock.

Let $\mathcal{CK}$ denote the set of all local clocks. We now define an ordering relation on the elements of $\mathcal{CK}$ as follows: for any $ck_1, ck_2 \in \mathcal{CK}$, $ck_1 \sqsubseteq ck_2$ iff for all $t \in ck_1$, we have

$t \in ck_2$, where the expression $t \in ck_i$ denotes the fact that t is a moment in time on the clock ck_i. It is easy to show that $(\mathcal{CK}, \sqsubseteq)$ is a complete lattice. Thus, we can define

$$ck_1 \sqcap ck_2 \overset{\text{def}}{=} g.l.b.\{ck_1, ck_2\} \qquad ck_1 \sqcup ck_2 \overset{\text{def}}{=} l.u.b.\{ck_1, ck_2\}$$

where $ck_1, ck_2 \in \mathcal{CK}$, *g.l.b.* stands for "greatest lower bound" and *l.u.b.* for "least upper bound" under the relation $\sqsubseteq$. The definitions can be extended to any given set of clocks.

We now give the definition of a clock assignment, which assigns local clocks for all predicate symbols.

Definition 2.1. *A clock assignment ck of TLC is a map from the set $\mathcal{SP}$ of predicate symbols to the set $\mathcal{CK}$ of clocks, i.e. $ck \in [\mathcal{SP} \to \mathcal{CK}]$. The notation $ck(p)$ denotes the clock which is associated with a predicate symbol p on a given clock assignment ck.*

We now extend the notion of a clock assignment to formulae.

Definition 2.2. *Let A and B be formulae, $p(x_1, x_2, \ldots, x_n)$ a pure atomic formula, and ck a clock assignment. We define a clock assignment ck^* on formulae of TLC as follows:*

$$\begin{array}{ll}
ck^*(p(x_1, x_2, \ldots, x_n)) = ck(p) & ck^*(\neg A) = ck^*(A) \\
ck^*(A \wedge B) = ck^*(A) \sqcap ck^*(B) & ck^*(A \vee B) = ck^*(A) \sqcap ck^*(B) \\
ck^*(\texttt{first}\ A) = ck^*(A) & ck^*(\texttt{next}\ A) = ck^*(A) \\
ck^*((\forall x)A) = ck^*(A) & ck^*((\exists x)A) = ck^*(A).
\end{array}$$

We now establish the result that every formula of *TLC* can be clocked. The proof of the lemma is straightforward by induction on the structure of formulae.

Lemma 2.1. *Let A be a formula of TLC, $\mathcal{SP}_{\mathcal{A}}$ the set of predicate symbols occurring in A, and ck a clock assignment. Then $ck^*(A) = \sqcap_{p \in \mathcal{SP}_{\mathcal{A}}} ck(p)$ (i.e., every formula of TLC can be clocked).*

Since $ck^*(A)$ is completely determined by the given clock assignment *ck*, we refer to the clock of A under *ck* simply as $ck(A)$. In notation, $ck(A) = \sqcap_{p \in \mathcal{SP}_{\mathcal{A}}} ck(p)$.

In the following, we always use *ck* to represent a clock assignment and simply call it a clock, and we also use the notation $ck(A), ck(B), \ldots, ck_0, ck_1, \ldots$ to represent local clocks. Particularly, we denote the global clock as *gck*.

Definition 2.3. (rank) *Given a local clock $ck_i = \langle t_0, t_1, t_2, \ldots \rangle$. We define the rank of t_n on ck_i to be n, written as $rank(t_n, ck_i) = n$. Inversely, we write $t_n = ck_i^{(n)}$, which means that t_n is the moment in time on ck_i whose rank is n.*

Obviously, for the global clock, we have that $rank(t, gck) = t$ and $gck^{(t)} = t$.

2.3. temporal interpretations, semantics

In *TLC*, at a given time $t \in \omega$, the value of a formula can be true, false or undefined, depending on the clocks of predicate symbols appearing in it. The meaning of a predicate symbol p is actually a partial mapping from ω to $P(\mathbf{D}^n)$ where n is the arity of p, $\mathbf{D}$ is the domain of discourse, $\mathbf{D}^n$ is the n-folded Cartesian product of $\mathbf{D}$, and $P(\mathbf{D}^n)$ is the power set of $\mathbf{D}^n$. For any $t \in ck(p)$, the mapping is naturally defined, i.e., there will be a corresponding subset of $\mathbf{D}^n$; otherwise the image is undefined.

A temporal interpretation together with a clock assignment assigns meanings to all the basic elements of *TLC*.

Definition 2.4. *A temporal interpretation I on a given clock ck of TLC 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 $[\mathbf{D}^n \to \mathbf{D}]$; and for each n-ary predicate symbol p, an element of $[ck(p) \to P(\mathbf{D}^n)]$.*

Now we give the definition of the satisfaction relation $\models$. In the following, the notation $\models_{I,ck,t} A$ denotes the fact that a formula A is true at moment $t(\in ck(A))$ under I on ck.

Definition 2.5. *Let I be a temporal interpretation on a given clock ck of TLC, and A and B formulae of TLC. The semantics of elements of TLC are given inductively by the following:*

(1) *If $f(e_0, \ldots, e_{n-1})$ is a term, then $I(f(e_0, \ldots, e_{n-1})) = I(f)(I(e_0), \ldots, I(e_{n-1}))$.*
(2) *For any n-ary predicate symbol p and terms $e_0, \ldots, e_{n-1}$ and any $t \in ck(p)$, $\models_{I,ck,t} p(e_0, \ldots, e_{n-1})$ iff $\langle I(e_0), \ldots, I(e_{n-1})\rangle \in I(p)(t)$.*
(3) *For any $t \in ck(A)$, $\models_{I,ck,t} \neg A$ iff it not the case that $\models_{I,ck,t} A$.*
(4) *For any $t \in ck(A \wedge B)$, $\models_{I,ck,t} (A \wedge B)$ iff $\models_{I,ck,t} A$ and $\models_{I,ck,t} B$.*
(5) *For any $t \in ck(A)$, $\models_{I,ck,t} (\forall x)A$ iff $\models_{I[d/x],ck,t} A$ for all $d \in \mathbf{D}$ where the interpretation $I[d/x]$ is just like I except that the varible x is assigned the value d in $I[d/x]$.*
(6) *For any $t \in ck(A)$, $\models_{I,ck,t}$ `first` A iff $\models_{I,ck,ck(A)^{(0)}} A$.*
(7) *For any $t \in ck(A)$, $\models_{I,ck,t}$ `next` A iff $\models_{I,ck,ck(A)^{(i+1)}} A$, where $i = rank(t, ck(A))$.*

Let $\models_{I,ck} A$ denote the fact that A is true under I on ck, in other words, $\models_{I,ck} A$ if and only if $\models_{I,ck,t} A$ for all $t \in ck(A)$. Let $\models_{ck} A$ denote the fact that A is true in any temporal interpretation on ck and $\models A$ denote the fact that A is true in any temporal interpretation on any clock. If $\models_{I,ck} A$, then we say that the temporal interpretation I on ck is a model on ck of the formula A.

2.4. axioms and inference rules

Let A be a formula of *TLC* and ck a clock. We write

$\vdash A$: A is a theorem of *TLC*
$\vdash_{ck} A$: A is a theorem on ck of *TLC*
$\vdash_{ck,t} A$: A is a theorem at moment $t(\in ck(A))$ on ck of *TLC*.

The following axioms (theorems) related to the temporal operators state some important properties of the extended language. Let A and B be formulae of the language. Read $\leftrightarrow$ as "if and only if".

Axioms

A1. $\vdash$ `first first` $A \leftrightarrow$ `first` A.
A2. $\vdash$ `first` $(\neg A) \leftrightarrow \neg$(`first` A).
A3. $\vdash$ `next` $(\neg A) \leftrightarrow \neg$(`next` A).
A4. $\vdash$ `first` $(\forall x)(A) \leftrightarrow (\forall x)$(`first` A).
A5. $\vdash$ `next` $(\forall x)(A) \leftrightarrow (\forall x)$(`next` A).
A6. $\vdash_{ck}$ `first` $(A \wedge B) \leftrightarrow$ (`first` A)$\wedge$(`first` B), where $ck(A)^{(0)} = ck(B)^{(0)}$.
A7. $\vdash_{ck}$ `next` $(A \wedge B) \leftrightarrow$ (`next` A)$\wedge$(`next` B), where $ck(A) = ck(B)$.

Inference rules

In addition to substitution and Modus Ponens(MP), we have the following rules:

R1. If $\vdash_{ck} A$, then $\vdash_{ck}$ `first` A, when $ck(A)$ is a non-empty clock.
R2. If $\vdash_{ck} A$, then $\vdash_{ck}$ `next` A, when $ck(A)$ is infinite.
R3. If $\vdash_{ck} B \rightarrow A$, $\vdash_{ck,t} B$ and $t \in ck(B \rightarrow A)$, then $\vdash_{ck,t} A$.

R1 and **R2** are called the temporal operator introduction rules. The rule **R2** holds only when there is always a next moment in time on the clock $ck(A)$. Using **R3**, we are allowed to consider the case when $ck(A) \neq ck(B)$.

In this paper, we do not attempt to discuss the completeness of the axiomatic system for *TLC*. However, the correctness (soundness) of the axioms and the rules is straightforward.

LEMMA 2.2. *The axioms A1–A7 and the rules R1–R3 are valid with respect to the semantics scheme for TLC.*

2.5. CLOCKED ATOMS

We say that an atom is fixed-time if it has an application of `first` followed by a number of applications of `next`. Any fixed-time atom is fixed to some moment in time on the local clock associated with the formula which the atom is currently involved in. For example, suppose that `p(x)` appears in formula A and $ck(A) = \langle 2, 5, 8, \ldots \rangle$. Then, for $ck(A)$, `first p(x)` is fixed to moment 2, `first next p(x)` is fixed to moment 5 and so on. Since different formulae may have different local clocks, the same predicate appearing in different formulae may have different fixed forms even when it is fixed to the same moment in time in the different formulae. Therefore we need the following definition.

DEFINITION 2.6. *Let Q be a pure atom, ck_i a local clock, $t \in ck_i$, $ck_i \sqsubseteq ck(Q)$, and $n = rank(t, ck_i)$. Then* `first next`$(n)Q$ *is a fixed-time atom on ck_i, fixed to the moment t in time. Furthermore we call* `first next`$(n)Q|^t_{ck_i}$ *a clocked fixed-time atom on ck_i and t the current time of the clocked fixed-time atom.*

LEMMA 2.3. *Let Q be a pure atom appearing in a formula A and $t \in ck(A)$. Then*

`first next`$(n)Q$ *is true on $ck(A)$ iff* `first next`$(m)Q$ *is true on $ck(Q)$*

where $n = rank(t, ck(A))$ and $m = rank(t, ck(Q))$.

PROOF. We first show the following assertion by induction on n. For any given n,

`first next`$(n)Q$ is true on $ck(A)$ iff Q is true at time $ck(A)^{(n)}$.

For $n = 0$, according to the satisfaction relation given by Definition 2.5 and the fact that Q is a subformula of A, we have that `first` Q is true on $ck(A)$ iff Q is true at time $ck(A)^{(0)}$, and the assertion is obviously true.

Assume the assertion is true for $n = k$. By the inductive hypothesis, `first next(`k`)` `next` Q is true on $ck(A)$ iff `next` Q is true at time $ck(A)^{(k)}$. Moreover, by Definition 2.5, `next` Q is true at time $ck(A)^{(k)}$ iff Q is true at time $ck(A)^{(k+1)}$. Therefore we have that `first next(`k`) next` Q, i.e. `first next(`$k+1$`)` Q is true on $ck(A)$ iff Q is true at time $ck(A)^{(k+1)}$, which proves the assertion.

Similarly, for any given m, we can also show that

`first next(`m`)` Q is true on $ck(Q)$ iff Q is true at time $ck(Q)^{(m)}$.

By Definition 2.2 and from the assumption that $t \in ck(A)$ and Q is a pure atom in A, we have $t \in ck(Q)$. Let $n = rank(t, ck(A))$ and $m = rank(t, ck(Q))$, then we have $ck(A)^{(n)} = t = ck(Q)^{(m)}$. Therefore,

`first next(`n`)` Q is true on $ck(A)$ iff `first next(`m`)` Q is true on $ck(Q)$.

□

The lemma can simply be reformulated as:

`first next(`n`)` $Q|^{t}_{ck(A)}$ is true iff `first next(`m`)` $Q|^{t}_{ck(Q)}$ is true.

It actually forms the basis of the clocked TiSLD-resolution (see Section 4).

3. Program Structure in Chronolog(MC)

A Chronolog(MC) program consists of three components: $P = P_c \bowtie P_a \bowtie P_b$ where P_c, P_a and P_b are the clock definition, the clock assignment and the program body of the program P, respectively. The symbol $\bowtie$ means "jointing", that is, P_c, P_a and P_b jointly form the program P. P_c and P_a can be viewed as two independent programs. The clock ck of P is totally determined by P_c and P_a.

P_c is an ordinary Chronolog program which specifies all the local clocks involved in the program body. Program clauses in P_c are of the form A `<-` $B_0, \ldots, B_{n-1}$ where A and all B_i's are (temporal) atomic formulae. Informally, any given local clock ck_i can be represented as a meta-Chronolog program, which specifies the predicate $\texttt{ck}_\texttt{i}$ that is true at all the moments in time appearing on the clock ck_i:

first cki(cki$^{(0)}$).
next cki(S) <- cki(T), K=rank(T,cki), S is cki$^{(K+1)}$.

Since P_c is an ordinary Chronolog program, the definition of each `cki` may or may not "represent" an actual clock. We stipulate that each `cki` satisfy the following clock constraints:

Definition 3.1. (clock constraints)

- *For any successful query* `first next(m) cki(X)`*, we have that* $\texttt{m} \leq \texttt{X}$.
- *For any pair of successful queries* `first next(m) cki(X)` *and* `first next(m) cki(Y)`*, we have that* $\texttt{X} = \texttt{Y}$.

- *For any pair of successful queries* `first next(m) cki(X)` *and* `first next(n) cki(Y)`*, if* $\mathtt{m} < \mathtt{n}$*, then* $\mathtt{X} < \mathtt{Y}$*.*

The intuitive meaning of these constraints are as follows. The first one says that the rank of a moment on a clock is not greater than the moment. The second one says that clocks are single-valued at each moment. When the second constraint is relaxed, we have branching time. The third one says that clocks can only tick forwards, that is, `cki` defined by the representation is monotonic. In short, the last two constraints ensure that clocks are linear. The main motivation for the first constraint is computational (see Section 4).

Now the problem is that we have, in general, no way of checking whether clock constraints are satisfied by each clock definition. We can include some axioms in P_c formalizing the clock constraints, but then the semantics of clock definitions will be complicated. In this paper we impose a syntactic restriction on clock definitions. Program clauses which are allowed to appear in clock definitions for any given local clock `cki` can be only of the following form:

```
first cki(n).
next cki(N) <- cki(M), N is E(M), N > M.
```

where $\mathtt{n} \in \omega$ is the initial value of the clock, and `E(X)` is a single-valued function from ω to ω. The second clause specifies the "next" value of the clock using its current value. The declarative semantics of clock definitions are given in Section 5.

P_a is a typical Prolog program, which assigns local clocks for all the predicate symbols in the program body. It consists of several facts written in a simplified form such as

```
is-ck(p,cki).
```

The clause says that ck_i is the local clock associated with the predicate symbol p, where ck_i is defined in P_c and p appears in P_b. Note that there is only one clock assignment for any given predicate symbol. In short, P_a is the glue that binds a clock definition and a program body.

The program body P_b consists of rules and facts. It looks like an ordinary Chronolog program, but the meanings of all the program clauses appearing in P_b depend on a given clock, i.e. the clock definition P_c and clock assignment P_a. Two programs with different clock definitions or different clock assignments may give different results even when they have the same program body.

3.1. AN EXAMPLE PROGRAM

The following is a simple Chronolog(MC) program, which we call the Cat-and-Mouse (CAM) system. Suppose that there is a room where it is quiet only at odd moments in time when nobody occupies it, and the room is occupied at even moments in time by a cat and a mouse in turns. Also, there is an alarm set in the room. It makes a long sound and a short sound alternately, and the interval between two sounds is 3 time units. The first sound made by the alarm is short.

We define three predicates:

`occupies(X)`: `X` occupies the room.

`quiet:`	it is quiet in the room.
`alarm(X):`	The alarm makes `X` sound, where `X` is either `long` or `short`.

In the following program, the clock definition defines three local clocks, i.e. ck_1, ck_2, ck_3; and the clock assignment assigns clocks for three predicate symbols in the program body. Note that the global clock is not included in the clock definition; it is accessible using the symbol *gck*. When the clock of a predicate symbol is not assigned, it is assumed to be the global clock.

```
% CLOCK DEFINITION (ck1, ck2, ck3) %
first ck1(0).
next ck1(N) <- ck1(M), N is M+2.
first ck2(1).
next ck2(N) <- ck2(M), N is M+2.
first ck3(1).
next ck3(N) <- ck3(M), N is M+3.

% CLOCK ASSIGNMENT (ck) %
is-ck(occupies,ck1).
is-ck(quiet,ck2).
is-ck(alarm,ck3).

% PROGRAM BODY %
first occupies(mouse).
next occupies(cat) <- occupies(mouse).
next occupies(mouse) <- occupies(cat).
first quiet.
next quiet <- quiet.
first alarm(short).
next alarm(long) <- alarm(short).
next alarm(short) <- alarm(long).
```

Note that `ck1`, `ck2`, and `ck3` all satisfy the syntactic restriction on clock definitions stated above. We omit the condition `N` $>$ `M` from the clock definitions, because it is satisfied trivially.

Given the above program, suppose that we want to find who occupies the room at the next (even) moment in time after the initial moment 0. The query can be represented as the fixed-time goal:

```
<- first next occupies(X).
```

The answer to the goal is `X = cat` at time 2.

Goals can be fixed-time as above or open. If all atoms of a goal are fixed-time, then the goal is a fixed-time goal; otherwise it is an open goal. An open goal stands for a series of independent fixed-time goals. For example, the open goal `<- occupies(X)` stands for a series of independent fixed-time goals of form `<- first next(n) occupies(X)` for all $n \in \{0, 1, 2, 3, \ldots\}$.

In principle, any number of fixed-time goals can be executed in parallel. We call this form of parallelism "context parallelism", which is exploited in the parallel execution model for Chronolog (Liu *et al.*, 1995). We are studying an extension of the model for Chronolog(MC), incorporating the time-matching algorithms (see below). Chronolog(MC) programs keep all forms of parallelism of original Chronolog programs. Furthermore, because of introducing multiple clocks, reasoning about time can also be performed in parallel.

4. Clocked TiSLD-Resolution

TiSLD-resolution (Orgun and Wadge, 1993) is a proof procedure for Chronolog, which is applied to a set of canonical (fixed-time) program clauses and goal clauses. The correctness of TiSLD-resolution is based on the idea that the value of a given formula can be expressed in terms of the values of its canonical instances. In this section, we propose a clocked TiSLD-resolution for the execution of Chronolog(MC) programs as an extension of TiSLD-resolution.

4.1. temporal-unification

We need the notion of a canonical instance of a formula A. A canonical instance of A is a formula whose value is an invariant of time, that is, a formula which is fixed to a particular moment in time with respect to a given clock ck.

Definition 4.1. *Let A be a formula, and ck a clock. For any moment $t \in ck(A)$, the canonical instance of A fixed to that moment is* $\texttt{first next}(rank(t, ck(A)))\ A$.

Canonical instances of a formula can be obtained by the rules of inference.

The value of a given formula in a temporal interpretation can be expressed in terms of its canonical instances. The intuitive idea is that, for any given moment in time on the local clock $ck(A)$, we can find a canonical instance of the formula fixed to that moment in time and then combine the values of the canonical instances.

Lemma 4.1. *Let A be a formula, I a temporal interpretation, and ck a clock of TLC. Then $\models_{I,ck} A$ if and only if $\models_{I,ck} A_t$ for all canonical instances A_t of A on $ck(A)$.*

Proof. $\models_{I,ck} A$ iff $\models_{I,ck,t} A$ for all $t \in ck(A)$, iff $\models_{I,ck,t} \texttt{first next}(rank(t, ck(A)))\ A$, for all $t \in ck(A)$, iff $\models_{I,ck} \texttt{first next}(rank(t, ck(A)))\ A$, for all $t \in ck(A)$, because the value of the initial truths $\texttt{first next}(t, rank(A)))\ A$ are invariants of the moments on $ck(A)$, iff $\models_{I,ck} A_t$ for all canonical instances A_t of A, for all $t \in ck(A)$. $\square$

By Lemma 2.3, if two clocked fixed-time atoms have the same pure atom and the same current time, then they have the same truth value. Therefore, we have the following definition.

Definition 4.2. *Let $A|^t_{ck_1}$ and $B|^s_{ck_2}$ be two clocked fixed-time temporal atoms. If $t = s$ and the pure atoms contained in A and B can be unified, then we say that $A|^t_{ck_1}$ and $B|^s_{ck_2}$ can be unified, and the substitution θ, which unifies the pure atoms, is a substitution unifying $A|^t_{ck_1}$ and $B|^s_{ck_2}$.*

Note that there may be a difference between the numbers of applications of `next`'s in $A|^t_{ck_1}\theta$ and $B|^s_{ck_2}\theta$. But, we are still allowed to write that $A|^t_{ck_1}\theta = B|^s_{ck_2}\theta$ to express the fact that they are unified. For example, suppose that we have $ck_1 = \{0, 1, 2, 3, \ldots\}$ and $ck_2 = \{0, 2, 4, \ldots\}$, then the temporal atoms `first next next p(30)`$|^2_{ck_1}$ and `first next p(X)`$|^2_{ck_2}$ can be unified by the substitution $\{\mathtt{X}/30\}$. The result of the substitution can be either `first next next p(30)`$|^2_{ck_1}$ or `first next p(30)`$|^2_{ck_2}$. The user can choose either one of them as the result in a proof procedure.

Thus, in Chronolog(MC), the concept of temporal-unification has been extended, so that we are allowed to make unification on different local clocks.

4.2. ALGORITHMS FOR TIME-MATCHING

In Chronolog(MC), temporal-matching involves the matching of a selected clocked temporal atom from the current goal and the variant of a program clause. The matching includes two aspects: (i) matching the pure atoms, and (ii) matching the current time.

The matching of pure atoms is trivial; while time-matching is involved in the execution of clock definitions and clock assignments. Therefore, in executing a Chronolog(MC) program, time-matching, or reasoning about time in the refutation procedure is essential. Suppose that we are given a Chronolog(MC) program $P = P_c \bowtie P_a \bowtie P_b$. The following three algorithms are used for time-matching.

Algorithm A is used to find all local clocks (different from the global clock) associated with the predicate symbols appearing in any given clause in P_b. Let SP denote a set of predicate symbols and SC a set of local clocks.

Algorithm A (find local clocks for a given clause A_0 `<-` $A_1, \ldots, A_n$.)

```
(1) Initialization. Set $SP = \{\}$, $SC = \{\}$ and $i = 0$.
(2) Get the predicate symbol $p$ from $A_i$.
(3) If $p \in SP$, go to step 5.
(4) Given the query <- is-ck($p$, X), execute the clock assignment $P_a$ using a standard
    Prolog implementation to obtain the answer. If the execution is successful and the
    answer is $\mathtt{X} = \mathtt{ck_j}$, set $SP = SP \cup \{p\}$ and $SC = SC \cup \{\mathtt{ck_j}\}$.
(5) Set $i = i + 1$. If $i \leq n$, go to step 2.
(6) Output $SC$, then stop.
```

A given clock assigns a unique local clock for each predicate symbol appearing in the program body. For any predicate symbol p, if there is a clause in the form of `is-ck(`p`, X)` in the clock assignment of the program, then `X` is its local clock, otherwise it is assumed that its local clock is the global clock gck.

Suppose that $p_1, p_2, \ldots, p_r$ are all the predicate symbols appearing in a given clause C, which have a local clock different from the global clock, and $ck(p_1) = ck_1, ck(p_2) = ck_2, \ldots, ck(p_r) = ck_r$, then the following clauses: $\mathtt{is\text{-}ck(p_1, ck_1)}$, $\mathtt{is\text{-}ck(p_2, ck_2)}, \ldots,$ $\mathtt{is\text{-}ck(p_r, ck_r)}$ should be included in the clock assignment of the program. Therefore, by running Algorithm A, we may obtain SC=$\{\mathtt{ck_1}, \mathtt{ck_2}, \ldots, \mathtt{ck}_r\}$. If SC is the empty set, then all predicate symbols have the global clock `gck` as their local clock.

Thus, by Definition 2.2, we have $ck(C) = \mathtt{ck_1} \sqcap \mathtt{ck_2} \sqcap \ldots \sqcap \mathtt{ck_r} \sqcap \mathtt{gck}$. The global clock `gck` is there because there may be other predicate symbols in C running on it. In addition, by Lemma 2.1, we can show that, for any $t \in \omega$, $t \in ck(C)$ if and only if $t \in \mathtt{ck_i}$

for all $\texttt{ck}_\texttt{i} \in SC$. In other words, $ck(C)$ is totally determined by the set of local clocks in SC.

Given any $t \in \omega$, Algorithm B is used to determine whether $t \in \texttt{ck}_\texttt{i}$ for any given local clock `cki` which has been defined in the clock definition of a program.

Algorithm B (Check whether $t \in \texttt{ck}_\texttt{i}$.)

```
(1) Set $k = t$.
(2) Given the query <- first next($k$) ck$_i$($t$), execute the clock definition using a
    standard Chronolog implementation to obtain the answer. If the execution is suc-
    cessful, i.e. the answer is "Yes", go to step 5.
(3) Set $k = k - 1$. If $k \geq 0$, go to step 2.
(4) Output "the fact that $t \in \texttt{ck}_\texttt{i}$ is not true", then stop.
(5) Output "$t \in \texttt{ck}_\texttt{i}$", then stop.
```

The constraints on clock definitions proposed in Section 3 together with the correctness of the Chronolog implementation (Orgun and Wadge, 1993) guarantee the correctness of Algorithm B. If $t \in \texttt{ck}_\texttt{i}$, then Algorithm B terminates and gives the fact that $t \in \texttt{ck}_\texttt{i}$; if t does not belong to $\texttt{ck}_\texttt{i}$, then Algorithm B terminates and gives the fact that $t \in \texttt{ck}_\texttt{i}$ is not true. In fact, the first constraint ensures that the search space for the given query is finite, and therefore the algorithm terminates.

Algorithm C is used to find the rank of a time-value on the local clock $ck(C)$ for any given clause C of the program body of a program.

Algorithm C (Given $SC = \{\texttt{ck}_1, \texttt{ck}_2, \ldots, \texttt{ck}_\texttt{r}\}$ by Algorithm A and $t \in ck(C)$, find $rank(t, ck(C))$)

```
(1) Let $RANK = t$ and $i = RANK - 1$
(2) If $i < 0$, go to step 8.
(3) Set $j = 1$.
(4) Check whether $i \in \texttt{ck}_\texttt{j}$ (by Algorithm B). If so, go to step 6.
(5) let $RANK = RANK - 1$, go to step 7.
(6) Set $j = j + 1$. If $j \leq r$ go to step 4.
(7) Let $i = i - 1$. Go to step 2.
(8) Output $rank(t, ck(C)) = RANK$, then stop.
```

Based on the correctness of Algorithm B, the correctness of Algorithm C can be proved by induction on t. Note that if SC is the empty set, then the algorithm terminates immediately with the result $t = rank(t, ck(C))$.

Consider the matching of a selected clocked temporal atom, say $A|^t_{ck_0}$, from the current goal and the variant of a program clause, say $C = H$ `<-` $B_1, \ldots, B_n$. Suppose $A|^t_{ck_0}$ and H contain the same predicate symbol and their pure atoms have been matched. Because $t \in ck_0$, to check if the current time is matched, we only need to check whether $t \in ck(C)$ is true or not.

To do this, we can first find all the local clocks (different from the global clock) associated with the predicate symbols appearing in C by Algorithm A. Suppose we have obtained $SC = \{\texttt{ck}_1, \texttt{ck}_2, \ldots, \texttt{ck}_\texttt{r}\}$. Then, we need to check whether $t \in \texttt{ck}_\texttt{i}$ for all i, $1 \leq i \leq r$ by Algorithm B. If so, then we have $t \in ck(C)$.

The fact that $t \in ck(C)$ means that the current time can be matched. In other words, there exists a canonical instance of C, whose head can be matched with $A|_{ck_0}^{t}$.

Suppose that $A|_{ck_0}^{t}$ and the head H of the clause C has been matched, then we have to find $rank(t, ck(C))$, and obtain the clocked form of the canonical instance of C. This task can be performed by Algorithm C.

4.3. PROOF PROCEDURE

We know that, to obtain the answer to an open goal, we have to obtain the answers from all fixed-time subgoals of that goal, which are regarded as independent computations.

By Lemma 2.1, every clause (formula) in a given Chronolog(MC) program can be clocked, so can every goal. Let P be a Chronolog(MC) program and $G = \texttt{<-}\ L_0, \ldots, L_s$ a fixed-time goal. Since G can be clocked, by Definition 2.6, we may rewrite the goal as

$$\texttt{<-}\ L_0|_{ck(G)}^{t_{L_0}}, \ldots, L_s|_{ck(G)}^{t_{L_s}}.$$

The clocked fixed-time form for G is called its rewritten form.

Definition 4.3. *Given a Chronolog(MC) program P and a fixed goal $G = \texttt{<-}\ L_0, \ldots, L_s$. A clocked TiSLD-derivation of $P \cup \{G\}$ is a sequence of triples with the following form:*

$$E = \{\langle G_0, C_0, \theta_0\rangle, \langle G_1, C_1, \theta_1\rangle, \ldots\}$$

where G_0 is the rewritten form of G as above and $G_0, G_1, \ldots$ are goal clauses whose atoms are all clocked fixed-time; $C_0, C_1, \ldots$ the variants (renamed and clocked) of canonical instances of program clauses in P; and $\theta_0, \theta_1, \ldots$ the substitutions.

In the refutation procedure of Chronolog(MC), when a clocked temporal atom from the goal is selected, it is matched against program clauses by the clocked temporal matching and unification. A new goal is produced after replacing the selected atom in the goal by the body of the matching canonical instance and then substitution (i.e. the variable bindings) obtained from unification is applied to the new goal.

Suppose that at a step of a proof procedure, we have the TiSLD-derivation

$$E = \{E_0, E_1, \ldots, E_i\}$$

where $E_k = \langle G_k, C_k, \theta_k\rangle$, $k = 0, \ldots, i$. In particular, suppose that we have

$$E_i = \langle(\texttt{<-}\ A_0|_{ck_0}^{t_{A_0}}, \ldots, A_l|_{ck_l}^{t_{A_l}}, \ldots, A_{s-1}|_{ck_{s-1}}^{t_{A_{s-1}}}), (A\ \texttt{<-}\ B_0, \ldots, B_{m-1}), \theta_i\rangle$$

and at the next step, A_l is the selected temporal atom in the clocked fixed-time goal G_i via the computation rules, and it is shown that $t_{A_l} \in ck(C_k)$ by time-matching Algorithms A and B, and furthermore we have $A_l|_{ck_l}^{t_{A_l}}\theta_i = A|_{ck(C_k)}^{t_{A_l}}\theta_i$ with mgu θ_i, then

$$E_{i+1} = \langle G_{i+1}, C_{i+1}, \theta_{i+1}\rangle$$

where

$$\begin{aligned} &G_{i+1} \\ &= (\texttt{<-}\ A_0|_{ck_0}^{t_{A_0}}, \ldots, A_{l-1}|_{ck_{l-1}}^{t_{A_{l-1}}}, B_0|_{ck(C_k)}^{t_{B_0}}, \ldots, B_{m-1}|_{ck(C_k)}^{t_{B_{m-1}}}, A_{l+1}|_{ck_{l+1}}^{t_{A_{l+1}}}, \ldots, A_{s-1}|_{ck_{s-1}}^{t_{A_{s-1}}})\theta_i. \end{aligned}$$

Note that the clocked fixed-time atoms $B_0|_{ck(C_k)}^{t_{B_0}}, \ldots, B_{m-1}|_{ck(C_k)}^{t_{B_{m-1}}}$ are all obtained by the time-matching algorithm C, based on the canonical instance of the formula C_k whose head is matched with $A_l|_{ck_l}^{t_{A_l}}$. Now the TiSLD-derivation we obtain is

$$E = \{E_0, E_1, \ldots, E_i, E_{i+1}\}.$$

For any list E associated with a successful TiSLD-derivation, we have that for some $n \geq 0$, $G_n =$ `<-`, in which case the list E has length n with the last element $\langle G_{n-1}, C_{n-1}, \theta_{n-1}\rangle$.

An essential difference between the Clocked TiSLD-resolution and the original TiSLD-resolution is that at every step of a clocked TiSLD-derivation all atoms in the goal clause and the selected canonical instances (variant) of a program clause are clocked.

Recall the CAM system. Suppose that we are given the fixed-time goal G:

```
<- first occupies(X), first alarm(Y).
```

From the clock definition and assignment of the program, we have $ck($`occupies`$) =$ `ck1`, $ck($`alarm`$) =$ `ck3`, `ck1` $= \langle 0, 2, 4, 6, 8, 10, \ldots\rangle$ and `ck3` $= \langle 1, 4, 7, 10, \ldots\rangle$, so $ck(G) = \langle 4, 10, \ldots\rangle$. Therefore, the rewritten form of the goal G is:

$$\texttt{<- first occupies(X)}|_{ck(G)}^{4}\texttt{, first alarm(Y)}|_{ck(G)}^{4}.$$

The proof of the goal is as follows:

$G_0 =$ `<- first occupies(X)`$|_{ck(G)}^{4}$`, first alarm(Y)`$|_{ck(G)}^{4}$.

$C_0 =$ `first next next occupies(mouse)`$|_{ck1}^{4}$
  `<- first next occupies(cat)`$|_{ck1}^{2}$.

$\theta_0 = \{$`X/mouse`$\}$.

$G_1 =$ `<- first next occupies(cat)`$|_{ck1}^{2}$`, first alarm(Y)`$|_{ck(G)}^{4}$ θ_0.

$C_1 =$ `first next occupies(cat)`$|_{ck1}^{2}$ `<- first occupies(mouse)`$|_{ck1}^{0}$.

$\theta_1 = \{$`X/mouse`$\}$.

$G_2 =$ `<- first occupies(mouse)`$|_{ck1}^{0}$`, first next alarm(y)`$|_{ck(G)}^{4}$ θ_1.

$C_2 =$ `first occupies(mouse)`$|_{ck1}^{0}$.

$\theta_2 = \{$`X/mouse`$\}$.

$G_3 =$ `<- first alarm(Y)`$|_{ck(G)}^{4}$ θ_2.

$C_3 =$ `first next alarm(long)`$|_{ck3}^{4}$ `<- first alarm(short)`$|_{ck3}^{1}$.

$\theta_3 = \{$`X/mouse, Y/long`$\}$.

$G_4 =$ `<- first alarm(short)`$|_{ck3}^{1}$ θ_3.

$C_4 =$ `first alarm(short)`$|_{ck3}^{1}$.

$\theta_4 = \{$`X/mouse, Y/long`$\}$.

When the selected temporal atom in G_4 is replaced by the body of C_4, we have that $G_5 =$ `<-`, meaning that the refutation is successful. The answer to the goal is `X` $=$ `mouse`, `Y` $=$ `long`.

Based on clock constraints (Definition 3.1) and the correctness of time-matching algorithms, the soundness of clocked TiSLD-resolution can be proved by induction on the length of a refutation. To show the completeness of clocked TiSLD-resolution, we need to consider the fact that a given Chronolog(MC) program (body) can be expressed as the set of all canonical instances of program clauses in the program (body). Therefore we can first show that clocked TiSLD-resolution is a complete proof procedure when restricted to canonical ground instances of program clauses, then lift a ground clocked TiSLD-refutation to a clocked TiSLD-refutation. Proofs of the analogous results from Orgun and Wadge (1993) can be adapted for the purpose.

5. Declarative Semantics

The program body is the main component of a Chronolog(MC) program, and its semantics is therefore naturally presented as the declarative semantics of the program. However, the clock definition and assignment of a Chronolog(MC) program can be viewed as procedures attached to the program body; as independent programs, they naturally have their own semantics. Therefore, the semantics for the entire program depends on the semantics of the clock definition and assignment.

5.1. THE SEMANTICS OF CLOCK DEFINITIONS

Let $P = P_c \bowtie P_a \bowtie P_b$ be a Chronolog(MC) program. The clock assignment P_a is a set of Horn clauses, i.e. a Prolog program. The semantics results for Prolog programs can be found from a number of references, such as van Emden and Kowalski (1976) and Lloyd (1987). P_c is an ordinary Chronolog program, therefore we can directly obtain the declarative semantics of clock definitions from the declarative semantics of Chronolog programs; Orgun and Wadge (1992, 1993) showed that the minimum temporal Herbrand model of a Chronolog program exists, and it is characterized by the intersection of all the temporal Herbrand models of the program.

P_c, as an ordinary Chronolog program, has the property that all predicates run on the global clock $\langle 0, 1, 2, 3, \ldots \rangle$. The declarative semantics of P_c can be developed in terms of temporal Herbrand models as follows. Herbrand universe of P_c, denoted by U_{P_c}, is generated by constants and function symbols that appear in P_c. The Herbrand base B_{P_c} of P_c consists of all those canonical temporal atoms generated by predicate symbols appearing in P_c with terms in U_{P_c} used as arguments. Subsets of B_{P_c} are regarded as temporal Herbrand interpretations of P_c.

Let I be a temporal interpretation of P_c with U_{P_c} as its domain. Then I is identified with a subset H of B_{P_c} by the following:

$$\langle e_0, \ldots, e_{n-1} \rangle \in I(p)(t) \text{ iff } \texttt{first next}(t)\, p(e_0, \ldots, e_{n-1}) \in H,\ t \in \omega.$$

Thus, we have the following results which are implied by the analogous results for Chronolog (Orgun and Wadge, 1992, 1993).

THEOREM 5.1. *Let $P = P_c \bowtie P_a \bowtie P_b$ be a Chronolog(MC) program. Then*

(1) $\models_{B_{P_c}} P_c$.
(2) $\cap M_c = \cap_{\alpha \in S_c} I_\alpha$ *is a temporal Herbrand model of* P_c *where* $M_c = \{I_\alpha\}_{\alpha \in S_c}$ *be a non-empty set of temporal Herbrand models of* P_c.
(3) $MMOD(P_c) \overset{\text{def}}{=} \cap_{\alpha \in S_c} I_\alpha$ *is the minimum temporal Herbrand model of* P_c, *where* $\{I_\alpha \mid \models_{I_\alpha} P_c\}_{\alpha \in S_c}$ *is the set of temporal Herbrand models of* P_c.
(4) $MMOD(P_c) = \{A \in B_{P_c} \mid P_c \models A\}$.

Recall that clock definitions have a restricted form of Chronolog clauses. We need to show that all the clocks defined in P_c satisfy the clock constraints given in Section 3.

LEMMA 5.1. *Let P_c be a clock definition. For each predicate symbol* `cki` *defined in P_c,* `cki` *satisfies the clock constraints with respect to $\mathit{MMOD}(P_c)$.*

PROOF. The lemma can be proved by induction, based on the form of the syntactic restrictions for clock definitions (see Section 3).

We first show, by induction on `m`, that `cki` satisfies the first constraint: for any element `first next(m) cki(X)` in $\mathit{MMOD}(P_c)$, we have that $\mathtt{m} \leq \mathtt{X}$.

For $\mathtt{m} = \mathtt{0}$, according to the restriction on the clock definition, we have $\mathtt{X} \in \omega$, so $\mathtt{m} \leq \mathtt{X}$, and therefore `cki` satisfies the constraint.

Assume `cki` satisfies the constraint when $\mathtt{m} = \mathtt{k}$. Consider the case when $\mathtt{m} = \mathtt{k} + \mathtt{1}$. The fact that `first next(k+1) cki(X)` belongs to $\mathit{MMOD}(P_c)$ means that it matches a canonical instance of the second clause of the restriction with the following form:

```
first next(k+1) cki(X) <-
      first next(k) cki(Y),
      first next(k) X is E(Y),
      first next(k) X > Y.
```

and `first next(k) cki(Y)`, `first next(k) X is E(Y)`, `first next(k) X` $>$ `Y` are all successful queries. Hence `first next(k) cki(Y)` also belongs to $\mathit{MMOD}(P_c)$, and therefore, by the inductive hypothesis, we have $\mathtt{k} \leq \mathtt{Y}$. Thus, from the facts that `first next(k) X is E(Y)`, and `first next(k) X` $>$ `Y` are successful queries and `E` is a single-valued function from ω to ω, we have that $\mathtt{k} + \mathtt{1} \leq \mathtt{Y} + \mathtt{1} \leq \mathtt{X}$, which proves the assertion that `cki` satisfies the first constraint.

Based on the requirement that `E` is a single-valued function within the restriction on clock definitions, we can similarly show, by induction on `m`, that `cki` satisfies the second clock constraint.

To prove that `cki` satisfies the third clock constraint, i.e. to prove that, for any pair `first next(m) cki(X)` and `first next(n) cki(Y)` in $\mathit{MMOD}(P_c)$, if $\mathtt{m} < \mathtt{n}$, then $\mathtt{X} < \mathtt{Y}$, we need adopt a double-induction on `m` and `n`. We omit the details of the proof because of space limitations. □

The lemma implies that the clock represented by each `cki` can be recovered as follows:

$$\langle t \mid \mathtt{first\ next(}k\mathtt{)\ cki(}t\mathtt{)} \in \mathit{MMOD}(P_c)\rangle_{k\in\omega}.$$

5.2. CLOCKED TEMPORAL HERBRAND MODELS

Let $P = P_c \bowtie P_a \bowtie P_b$ be a Chronolog(MC) program. Since the predicate symbols appearing in the program body P_b have their own local clocks, the semantics of P depends on the given clock *ck*. We know that P is true in a temporal interpretation I on a given clock *ck* if and only if all program clauses in P_b are true in I on *ck*. By Lemma 4.1, a program clause is true in I on *ck* if and only if all canonical instances of the clause are true in I on *ck*. Therefore, as far as the declarative semantics is concerned, we can regard P as the set of all canonical instances of the program clauses in P_b.

In the program, the given clock *ck* is determined by P_c and P_a. To discuss clocked temporal Herbrand models, we have to know how to obtain *ck*. For a given predicate symbol p appearing in P_b, we want to know how to find $ck(p)$. Formally, we have:

DEFINITION 5.1. *Let $P = P_c \bowtie P_a \bowtie P_b$ be a Chronolog(MC) program and SP the set of all predicate symbols appearing in P_b. Then the clock ck of P_b is obtained as follows: For all $p \in SP$,*

- $ck(p) = \langle t \mid \texttt{first next(}k\texttt{) cki(}t\texttt{)} \in MMOD(P_c)\rangle_{k\in\omega}$ *if* $\texttt{is-ck(}p\texttt{, cki)} \in P_a$*;*
- $ck(p) = \langle 0, 1, 2, 3, \ldots\rangle = gck$ *if p is not assigned a local clock in P_a.*

Let U_P denote the clocked Herbrand universe of P which is generated by constants and function symbols that appear in P_b. The clocked temporal Herbrand base B_P of P consists of all those canonical temporal atoms generated by predicate symbols appearing in P_b with terms in U_P used as arguments. We regard subsets of B_P as clocked temporal Herbrand interpretations of P. Let I be a temporal interpretation on the clock *ck* of P with U_P as its domain. Then I is identified with a subset H of B_P by the following:

$$\langle e_0, \ldots, e_{n-1}\rangle \in I(p)(t) \text{ iff } \texttt{first next(}rank(t, ck(p))\texttt{)}p(e_0, \ldots, e_{n-1}) \in H, t \in ck(p).$$

Thus we have the result:

LEMMA 5.2. *Let $P = P_c \bowtie P_a \bowtie P_b$ be a Chronolog(MC) program, and ck the clock determined by P_c and P_a. Then $\models_{B_P, ck} P_b$.*

PROOF. For any program clause $C \in P_b$, we want to show that $\models_{B_P,ck} C$. In other words, we want to show, by Lemma 4.1, that $\models_{B_P,ck} C_t$ for all canonical instances C_t of C on $ck(C)$. To do this, we need only to prove that any ground canonical instance of C is true under the interpretation B_P on *ck*.

Assume (A <- $B_0, \ldots, B_{n-1}$) is a ground canonical instance of C, where $A, B_0, \ldots, B_{n-1}$ are all temporal atoms. By Definition 2.2, the clock for a ground canonical instance of a given program clause is the same clock for that clause. Therefore, we can rewrite the ground canonical instance as $(A|^{t_A}_{ck(C)} \texttt{ <- } B_0|^{t_{B_0}}_{ck(C)}, \ldots, B_{n-1}|^{t_{B_{n-1}}}_{ck(C)})$.

Consider $A|^{t_A}_{ck(C)}$. Assume

$$A|^{t_A}_{ck(C)} = \texttt{first next(}rank(t_A, ck(C))\texttt{)}p(e_0, \ldots, e_{i-1})|^{t_A}_{ck(C)},$$

where $p(e_0, \ldots, e_{i-1})$ is a pure ground atom. Since $t_A \in ck(C)$, again by Definition 2.2, we have $t_A \in ck(p)$. Also, by Lemma 2.3, we have

$$\begin{aligned}&\texttt{first next(}rank(t_A, ck(C))\texttt{)}p(e_0, \ldots, e_{i-1})|^{t_A}_{ck(C)}\\ &= \texttt{first next(}rank(t_A, ck(p))\texttt{)}p(e_0, \ldots, e_{i-1})|^{t_A}_{ck(p)}.\end{aligned}$$

According to the definition of B_P, $\texttt{first next(}rank(ck(p), t_A)\texttt{)}p(e_0, \ldots, e_{(i-1)})|^{t_A}_{ck(p)} \in B_P$, i.e. its truth value is true under the interpretation B_P on *ck*. Therefore $A|^{t_A}_{ck(C)}$ is true under B_P on *ck*, so is the ground canonical instance of C. □

The above lemma says that the entire clocked temporal Herbrand base B_P of a program $P = P_c \bowtie P_a \bowtie P_b$ is a clocked temporal model of the program (program body). The following lemma shows that the set of clocked temporal Herbrand models of a given Chronolog(MC) program is also closed under intersection.

LEMMA 5.3. *Let $P = P_c \bowtie P_a \bowtie P_b$ be a Chronolog(MC) program, ck the clock determined by P_c and P_a, and $M = \{I_\alpha\}_{\alpha \in S_b}$ a non-empty set of clocked temporal Herbrand models of P_b. Then $\cap M = \cap_{\alpha \in S_b} I_\alpha$ is a clocked temporal Herbrand model of P_b.*

PROOF. Obviously, $\cap M$ is a clocked Herbrand interpretation of P. We now show, by contradiction, that it is a model of P_b.

Suppose $\cap M$ is not a model of P_b. Then there exists at least one clause of P_b, say C, which is false under the interpretation $\cap M$. That means that there exists at least a ground canonical instance of C, say (A <- $B_0, \ldots, B_{n-1}$), which is false under $\cap M$. We therefore have that B_0 and ... and B_{n-1} are all true but A is false under $\cap M$.

We rewrite the ground canonical instance as $(A|^{t_A}_{ck(C)}$ <- $B_0|^{t_{B_0}}_{ck(C)}, \ldots, B_{n-1}|^{t_{B_{n-1}}}_{ck(C)})$. As in the above lemma, we can assume that

$$\begin{aligned}
A|^{t_A}_{ck(C)} &= \texttt{first next(}\mathit{rank}(t_A, ck(C))\texttt{)}p(e_0, \ldots, e_{i-1})|^{t_A}_{ck(C)} \\
&= \texttt{first next(}\mathit{rank}(t_A, ck(p))\texttt{)}p(e_0, \ldots, e_{i-1})|^{t_A}_{ck(p)}. \\
B_0|^{t_{B_0}}_{ck(C)} &= \texttt{first next(}\mathit{rank}(t_{B_0}, ck(C))\texttt{)}p_0(r_0, \ldots, r_{i_0-1})|^{t_{B_0}}_{ck(C)} \\
&= \texttt{first next(}\mathit{rank}(t_{B_0}, ck(p_0))\texttt{)}p_0(r_0, \ldots, r_{i_0-1})|^{t_{B_0}}_{ck(p_0)}. \\
&\ldots \\
B_{n-1}|^{t_{B_{n-1}}}_{ck(C)} &= \texttt{first next(}\mathit{rank}(t_{B_{n-1}}, ck(C))\texttt{)}p_{n-1}(s_0, \ldots, s_{i_{(n-1)}-1})|^{t_{B_0}}_{ck(C)} \\
&= \texttt{first next(}\mathit{rank}(t_{B_{n-1}}, ck(p_{n-1}))\texttt{)}p_{n-1}(s_0, \ldots, s_{i_{(n-1)}-1})|^{t_{B_{n-1}}}_{ck(p_{n-1})}.
\end{aligned}$$

where $p(e_0, \ldots, e_{i-1}), p_0(r_0, \ldots, r_{i_0-1}), \ldots, p_{n-1}(s_0, \ldots, s_{i_{(n-1)}-1})$ are pure ground atoms.

Since $B_0|^{t_{B_0}}_{ck(C)}, \ldots, B_{n-1}|^{t_{B_{n-1}}}_{ck(C)}$ all are true under $\cap M$,

$$\begin{gathered}
\texttt{first next(}\mathit{rank}(t_{B_0}, ck(p_0))\texttt{)}p_0(r_0, \ldots, r_{i_0-1})|^{t_{B_0}}_{ck(p_0)}, \\
\ldots \\
\texttt{first next(}\mathit{rank}(t_{B_{n-1}}, ck(p_{n-1}))\texttt{)}p_{n-1}(s_0, \ldots, s_{i_{(n-1)}-1})|^{t_{B_{n-1}}}_{ck(p_{n-1})}
\end{gathered}$$

all belong to $\cap M$, and therefore they all belong to any model I_α, $I_\alpha \in M$. On other side, since $A|^{t_A}_{ck(C)}$ is false under $\cap M$, $\texttt{first next(}\mathit{rank}(t_A, ck(p))\texttt{)}p(e_0, \ldots, e_{i-1})|^{t_A}_{ck(p)}$ does not belong to $\cap M$. Therefore there exists at least one model in M, say I, such that $\texttt{first next(}\mathit{rank}(t_A, ck(p))\texttt{)}p(e_0, \ldots, e_{i-1})|^{t_A}_{ck(p)}$ does not belong to I. Thus, we have that C is false under I, which contradicts our assumption that I is a model of P_b. □

THEOREM 5.2. *Let $P = P_c \bowtie P_a \bowtie P_b$ be a Chronolog(MC) program, ck the clock determined by P_c and P_a, and $M = \{I_\alpha \mid \models_{I_\alpha, ck} P\}_{\alpha \in S_b}$ the set of clocked temporal Herbrand models of P_b. Then $MMOD(P_b) \overset{\text{def}}{=} \cap_{\alpha \in S_b} I_\alpha$ is the minimum clocked temporal Herbrand model of P_b.*

PROOF. By Lemma 5.2, M is non-empty. Furthermore, by Lemma 5.3, $\cap M = \cap_{\alpha \in S_b} I_\alpha$ is a clocked temporal Herbrand model of P_b, and it is therefore the minimum clocked temporal Herbrand model. □

The following theorem says that the minimum clocked temporal Herbrand model of any Chronolog(MC) program consists of all ground canonical temporal atoms that are logical consequences of the program. Its proof is adapted from an analogous proof for Chronolog.

THEOREM 5.3. *Let* $P = P_c \bowtie P_a \bowtie P_b$ *be a Chronolog(MC) program, and ck the clock determined by* P_c *and* P_a. *Then* $MMOD(P_b) = \{A \in B_P \mid P \models_{ck} A\}$.

PROOF. A is a logical consequence of P on ck
iff $P_b \cup \{\neg A\}$ is unsatisfiable on ck
iff $P_b \cup \{\neg A\}$ have no clocked temporal Herbrand models
iff $\neg A$ is false under all clocked temporal Herbrand models of P_b
iff A is true under all clocked temporal Herbrand models of P_b
iff $A \in MMOD(P_b)$. $\Box$

The minimum clocked temporal Herbrand model of a Chronolog(MC) program can also be characterized by fixpoint theory. To do this, we may need discuss the fixpoint semantics in two levels (the clock definition level and the program body level). Because of space limitations, no details for the fixpoint semantics are given in this paper.

6. Conclusions

Chronolog(MC) is based on the temporal logic *TLC* which allows predicate symbols, and formulae, to be defined on different clocks. We have presented its logical basis and discussed the declarative semantics of Chronolog(MC) programs in terms of clocked temporal Herbrand models. We have proposed a clocked TiSLD-resolution as the proof procedure for executing Chronolog(MC) programs.

One application of Chronolog(MC) is simulation (Liu and Orgun, 1995, 1996). In describing a simulation system, we may consider that there are two parts which can been split semantically: one describing the functional aspects of the processes involved in the system, and the other describing their temporal aspects. The program body of a Chronolog(MC) program can be used for describing the functional aspects of each process involved in the system; and the clock definition and assignment together give the description of timing properties about the behavior of those processes.

There are a number of papers dealing with granularity of time. Ciapessoni *et al.* (1993) define a many-sorted first order logic language, whose semantics includes a temporal universe which consists of a finite set of disjoint and differently grained "temporal domains". Each temporal domain contains temporal instances expressed in the corresponding granularity. Cukierman *et al.* (1995) propose "time units classes" which are actually related with the time domains defined in Ciapessoni *et al.* (1993). Ladkin (1986) proposed a representation of time by non-convex intervals and introduced a standard form for an interval which represents an instance of one of various time units. Badaloni and Berati (1994) use different time scales for temporal planning systems to reduce the complexity of a real problem. In their notion of temporal granularity, a set of disjoint temporal domains $T_1, \ldots, T_n$, which are characterized by metrics, is defined and for any i, the domain T_{i+1} is a refinement of T_i. Mota *et al.* (1995) proposed a new time granularity theory based on modular temporal clauses, which is suitable for the specification of ecological models; Euzenat (1995) gives an algebraic approach to granularity in time representation, which uses granularity change operators for converting (upward and downward) qualitative time relationship from one granularity to another. Also, Hobbs (1985) proposes a formal characterization of the general notion of granularity, but gives no special attention to time granularity.

In *TLC*, local clocks can be any subsequences of the global clock. In other words, it does not require every granularity is a refinement of another one. The multiple granularity of time therefore is more flexible in time representation and describing timing properties of systems. Furthermore, in Chronolog(MC) the presentation of multiple granularity of time in a program is explicitly given by a clock definition and a clock assignment. The user is free to choose the granularity for each predicate symbol through clock assignments and definitions. Also, when the second clock constraint is relaxed, branching time is obtained. Then, given a goal, executing the clock definition involves finding a branch of time in which the goal can be proved.

There are also a number of papers that discuss branching time logics. Rescher and Urquhart (1971) considered different axiomatizations of branching-time structures. Abrahamson (1979) proposed additional ideas about branching time logics; then Ben-Ari *et al.* (1981) defined the unified branching time system (UB) which was extended by defining the computation tree logic (CTL) (Clark *et al.*, 1981, 1983). For more information on branching time logics, we refer the reader to the survey of Penczek (1995).

Future work includes completing a theoretical study about *TLC*, including the completeness of clocked TiSLD-resolution and the computational complexity of the time-matching algorithms. We are also considering a more general form of clock definitions in which clock constraints are expressed as additional axioms. Another extension is to allow the definition of clocks in terms of the same or other clocks. For example, the following clock definition specifies a Fibonacci-like clock $\langle 1, 2, 3, 5, 8, \ldots \rangle$:

```
% CLOCK DEFINITION (ckf) %
first ckf(1).
first next ckf(2).
next(2) ckf(N) <- next ckf(X), ckf(Y), N is X+Y.
```

Although Chronolog(MC) does not allow the addition of new clocks of finer granularity as a dense temporal logic might, it is possible to relax the requirement that each predicate symbol is assigned a clock in a given program. In other words, we may designate a set of predicate symbols as "input" predicates whose clocks are determined at run-time by the external environment. Such an extension may be useful when the clocks of some predicate symbols cannot be determined in advance and hence cannot be programmed into the program.

We may also enrich *TLC* with two standard temporal operators, $\Box$ and $\Diamond$, defined as follows:

$$\Box A \stackrel{\text{def}}{=} \texttt{first}\ A \wedge \Box(\text{next}\ A)$$

$$\Diamond A \stackrel{\text{def}}{=} \texttt{first}\ A \vee \Diamond(\text{next}\ A).$$

We read $\Box$ as "always" and $\Diamond$ as "sometime". The recursive definition of temporal operators $\Box$ and $\Diamond$ gives the following informal semantics: a formula of the form $\Box A$ is true at time t in a clocked temporal interpretation I just in the case A is true at all moments in time on the local clock $ck(A)$; a formula of the form $\Diamond A$ is true at time t in a clocked temporal interpretation I just in the case A is true at some moments in time on the local clock $ck(A)$. These new operators can improve the modeling power of TLC.

Acknowledgements

This work has been supported in part by an Australian Research Council (ARC) Grant. C. Liu has been supported by an Australian Postgraduate Award (APA) and an MPCE Postgraduate Research Scholarship at Macquarie University. Thanks are also due to two anonymous referees for their useful comments and suggestions and for pointing us to some relevant work on granularity of time.

References

Abadi, M., Manna, Z. (1989). Temporal logic programming. *J. Symbolic Computation* **8**, 277–295.

Abrahamson, K. (1979). Model logics for concurrent programs. In *LNCS* **70**, Berlin Heidelberg, Springer-Verlag.

Aoyagi, T. *et al.* (1986). Temporal logic programming language Tokio. In Wada, E., ed., *Logic Programming'85, LNCS* **221**, pp. 138–147. Berlin Heidelberg, Springer-Verlag.

Badaloni, S., Berati, M. (1994). Dealing with time granularity in a temporal planning system. In Gabbay, D.M., Ohlbach, H.J., eds, *Proceedings of ICTL'94, First International Conference on Temporal Logic*, Bonn, Germany, July 1994. (Published in *Lecture Notes in Artificial Intelligence* **827**, pp. 101–116. Berlin Heidelberg, Springer-Verlag.)

Baudinet, M. (1992). A simple proof of the completeness of temporal logic programming. In Fariñas del Cerro, L., Penttonen, M., eds, *Intensional Logics for Programming*, pp. 51–83. New York, Oxford University Press.

Ben-Ari, M., Manna, Z., Pnueli, A. (1981). The temporal logic of branching time. In *Proc. 8th Annual ACM Symposium on Principles of Programming Languages*, pp. 164–176, New York, ACM Press.

Brzoska, C. (1995). Temporal logic programming with metric and past operators. In Fisher, M., Owens, R., eds, *Executable Modal and Temporal Logics (IJCAI'93 Workshop)*. (Published in *Lecture Notes in Artificial Intelligence* **897**, pp. 21–39. Berlin Heidelberg, Springer-Verlag).

Ciapessoni, E., Corsetti, E., Montanari, A., San Pietro, P. (1993). Embedding time granularity in a logical specification language for synchronous real-time systems. *Science of Computer Programming* **20**, pp. 141–171.

Chomicki, J. (1994). Temporal query languages: A survey. In Gabbay, D.M., Ohlbach, H.J., eds, *Proc. ICTL'94, First International Conference on Temporal Logic*, Bonn, Germany, July 1994. (Published in *Lecture Notes in Artificial Intelligence* **827**, pp. 506–534. Berlin Heidelberg, Springer-Verlag.)

Clarke, E.M., Emerson, E.A. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. In *LNCS* **131**, pp. 52–71, Berlin Heidelberg, Springer-Verlag.

Clarke, E.M., Emerson, E.A., Sistla, A.P. (1983). Automatic verification of finite state concurrent systems using temporal logic specifications: a practical approach. *Proc. 10th Annual ACM Symposium on Principles of Programming Languages*, pp. 117–126, New York, ACM Press.

Cukierman, D., Delgrande, J. (1995). A language to express time intervals and repetition. In *TIME'95: The 2nd International Workshop on Temporal Representation and Reasoning*, April 26, 1995, Melbourne Beach, Florida.

Euzenat, J. (1995). An algebraic approach to granularity in time representation. In *TIME'95: The 2nd International Workshop on Temporal Representation and Reasoning*, April 26, 1995, Melbourne Beach, Florida.

Fisher, M. (1994). A survey of Concurrent METATEM—The language and its applications. In Gabbay, D.M., Ohlbach, H.J., eds, *Proc. ICTL'94, First International Conference on Temporal Logic*, Bonn, Germany, July 1994. (Published in *Lecture Notes in Artificial Intelligence* **827**, pp. 480–505. Berlin Heidelberg, Springer-Verlag.)

Fisher, M., Owens, R. (1995). An introduction to executable modal and temporal logics. In Fisher, M., Owens, R., eds, *Executable Modal and Temporal Logics (IJCAI'93 Workshop)*. (Published in *Lecture Notes in Artificial Intelligence* **897**, pp. 1–20, Berlin Heidelberg, Springer-Verlag.)

Fisher, M., Reynolds, M. (1995). The power of METATEM rules. In *Proc. IJCAI-95 Workshop on Executable Temporal Logics*, pp. 35–49. Montreal, Canada.

Frühwirth, T. (1995). Temporal logic and annotated constraint logic programming. In Fisher, M., Owens, R., eds, *Executable Modal and Temporal Logics (IJCAI'93 Workshop)*. (Published in *Lecture Notes in Artificial Intelligence* **897**, pp. 58–68, Berlin Heidelberg, Springer-Verlag.)

Gabbay D.M. (1987). Modal and temporal logic programming. In Galton, A., ed., *Temporal logics and their applications*, pp. 197–237. San Diego, CA, Academic Press.

Gabbay D.M. (1991). Modal and temporal logic programming II (A temporal Prolog machine). In Dodd, T., Owens, R., Torrance, S., eds, *Logic Programming—Expanding the Horizon.* Intellect Books Ltd.

Gabbay D.M., McBrien, P. (1991). Temporal Logic & Historical Databases. In *Proc. 17th Very Large Data Bases Conference*, pp. 423–430, Los Altos, CA, Morgan Kauffman.

Gagné, J.-R., Plaice, J. (1995). A non-standard temporal deductive database system. In *Proc. IJCAI-95 Workshop on Executable Temporal Logics*, pp. 51–62. Montreal, Canada.

Goldblatt, R. (1987). Logics of time and computation. *Lecture Notes* No: 7. CSLI—Centre for the Study of Language and Information, Stanford University.

Hrycej, T. (1993). A temporal extension of Prolog. *J. Logic Programming* **15**, 113–145.

Hobbs, J.R. (1985). Granularity. In *Proc. IJCAI-85: The Ninth International Joint Conference on Artificial Intelligence*, volume 1, pp. 1–4, Los Altos, CA, Morgan-Kaufman.

Ladkin, P. (1986). Primitives and units for time specification. In *Proc. AAAI-86: The Fifth National Conference on Artificial Intelligence*, pp. 354–359, Los Altos, CA, Morgan-Kaufman.

Liu, C. (1995). A temporal logic language with time granulation. In *The 30th Anniversary of the Australian Association for Logic Conference*, Armidale, Australia.

Liu, C., Orgun, M.A. (1995). Chronolog as a simulation language. In *Proc. IJCAI-95 Workshop on Executable Temporal Logics*, pp. 109–119. Montreal, Canada.

Liu, C., Orgun, M.A. (1996). Executing specifications of distributed computations with Chronolog(MC). In *Proc. ACM Symposium on Applied Computing 1996*, pp. 393–400. Marriott Hotel, Philadelphia, USA, February 18–20, 1996, New York, ACM Press.

Liu, C., Orgun, M.A., Zhang, K. (1995). A framework for exploiting parallelism in Chronolog. In *Proc. First IEEE International Conference on Algorithms and Architectures for Parallel Processing*, pp. 163–172. Brisbane, Australia. Piscataway, NJ, IEEE Press.

Lloyd, J.W. (1987). *Foundations of Logic Programming*, 2nd edn, Berlin Heidelberg, Springer-Verlag.

Manna, Z., Pnueli, A. (1981). Verification of concurrent programs: the temporal framework. In R.S. Boyer, J.S. Moore, eds, *Correctness Problem in Computer Science*, pp. 215–273. Academic Press.

Mota, E. *et al.* (1995). Time granularity in simulation of models of ecological systems. In *Proc. IJCAI-95 Workshop on Executable Temporal Logics*, pp. 77–94. Montreal, Canada.

Orgun, M.A., Ma, W. (1994). An overview of temporal and modal logic programming. In Gabbay, D.M., Ohlbach, H.J., eds, *Proc. ICTL'94, First International Conference on Temporal Logic*, Bonn, Germany, July 1994. (Published in *Lecture Notes in Artificial Intelligence* **827**, pp. 445–479. Berlin Heidelberg, Springer-Verlag.)

Orgun, M.A., Wadge, W.W. (1992). Theory and practice of temporal logic programming. In Fariñas del Cerro, L., Penttonen, M., eds, *Intensional Logics for Programming*, pp. 23–50. New York, Oxford University Press.

Orgun, M.A., Wadge, W.W. (1993). Chronolog admits a complete proof procedure. In *Proc. 6th International Symposium on Lucid and Intensional Programming*, pp. 120–135. Quebec City, Canada.

Penczek, W. (1995). Branching time and partial order in temporal logics. In Bolc, L., Szalas, A., eds, *Time and Logic: A Computational Approach*, pp. 179–228. London, UCL Press.

Rescher, N., Urquhart, A. (1971). *Temporal Logic*, Berlin Heidelberg, Springer-Verlag.

Sadri, F. (1987). Three approaches to temporal reasoning. In Galton, A., ed., *Temporal Logics and Their Applications*, pp. 121–168. San Diego, CA, Academic Press.

van Emden, M.H., Kowalski, R.A. (1976). The semantics of predicate logic as a programming language. *J. Association for Computing Machinery* **23**, 733–742.

Wiederhold, G., Jajoida, J., Litwin, W. (1991). Dealing with granularity of time in temporal databases. In *Advanced Information Systems Enginering: Proc. Third International Conference CAiSE'91*, pp. 124–140, Berlin Heidelberg, Springer-Verlag.