Time-Extraction for Temporal Logic—Logic Programming and Local Process Time

TAKASHI HATTORI

Research Institute for Mathematical Sciences, Kyoto University, Kita-shirakawa, Sakyo-ku, Kyoto 606, Japan

Received June 26, 1987; revised November 30, 1988

Temporal logic is useful to describe a variety of computer systems such as operating systems and real-time process control systems, where explicit treatment of time plays an essential role. In the logic, the notion of time is represented by a sequence of states at each point in time, which is called a time stream. In distributed environments, it can allow simple descriptions of processes to deal with each process as if it had its own proper time stream where a proper time stream, called an extracted time stream, consists of the events which are essential to the process and are extracted from the original universal time stream. It is proved that, for given formulas which are interpreted in one of the extracted time streams, there exist certain formulas such that they are interpreted in the universal time stream and are equivalent to the given formulas. This time-extraction is applied to the temporal prolog in order to decompose a program into pieces, each of which works in its own time stream. In the same way as logical formulas, a program with time-extraction can be transformed to an equivalent program without time-extraction. It is also proved that the transformations preserve equivalence in the sense of model-theoretic semantics. © 1994 Academic Press, Inc.

1. INTRODUCTION

Logical formulas have been widely used in order to describe programming semantics. For logic programming, logical formulas are programs, which is not perfectly natural when the programs deal with dynamic objects because logical formulas represent static assertions by nature. In this respect, temporal logic is more useful than ordinal logic because the former can describe the notion of time explicitly. There are still some difficulties, however, when we use temporal logic directly as a programming language.

Actions of some software such as operating systems and real-time controlling systems depend on sequences of events that happen outside the computers. As mentioned later, the time in temporal logic is discrete; each period does not need to be associated to a physical interval. Rather, we often “notch” the time by events, which allows a clear view of the system. If a single program had to respond to all kinds of events by itself, its behavior would be too complicated to give a simple description. Although the usual solution for this problem is to make several processes so that the job is divided among them, whose benefits are well known as modular
programming, the notion of a process has not yet been formalized in the temporal logic. (The multiprocess network logic [8] presented spatial modalities that are orthogonal to temporal modalities.) We should presume that it would be much easier to write part of the program without any attention to the events which are not totally relevant to its own proper job. This can be accomplished by introducing a process for each part of the program such that the process has its own local time. As a result, the program virtually works on multiple time streams in which the processes are executed and are synchronized with each other at some point in time. On the other hand, it might be necessary to combine these "virtual" time streams into a single "real" time stream in order to increase efficiency.

Let us consider a more concrete example. Suppose we want to describe \( r \) holds when \( p \) and \( q \) happen in this order, then using a "previous operator" \( \bullet \), one of the modal operators in logic, we may write

\[
\bullet p \land q \rightarrow r.
\]

However, this formula simply states that \( r \) holds when \( q \) happens at the very next point in time at the point when \( p \) happens." In this case, some auxiliary predicates are needed in order to remember the state of \( \langle p \text{ happened and } q \text{ has not yet happened} \rangle \) because there can exist some time points in which neither \( p \) nor \( q \) happen. Therefore we should introduce a new predicate \( a \) and write

\[
\begin{align*}
p &\rightarrow a \\
\bullet a \land \neg q &\rightarrow a \\
\bullet a \land q &\rightarrow r.
\end{align*}
\]

One way to prevent such redundancy is to introduce a new binary temporal operator such as \textit{atnext}, \textit{until} to specify complex temporal relations [2]. Numerous kinds of temporal operators will be needed, however, when we want to write more complicated sequences of events. Rather, we introduce the notion of \textit{time-extraction}, where we allow multiple time streams and each time stream is assigned to a sequence of events or, in other words, a process. This approach requires no extra temporal operators and allows simple descriptions with the explicit notion of process.

In the following section, first we define the notion of time-extraction and examine the relationship between formulas in "virtual" and "real" time streams. Next we apply time-extraction to the temporal prolog [9] and give an algorithm which transforms a program using extraction to an equivalent and possibly more efficient program without extraction. The possibility of generalizing time-extraction for other modal logic is also mentioned.
2. First-Order Linear-Time Temporal Logic

The modal logic is different from the ordinal logic at the point that it has some modal operators, in addition to logical ones. Its model consists of a set of possible worlds (worlds for short) and a set of accessibility relations. Each world is an interpretation of predicate and function symbols, which is exactly the same as the model of the first-order logic. The truth value of a formula can be different in each world. The visibility relation, called modality, is a set of ordered pairs of worlds and is associated to a modal operator. In a certain world, the truth value of a formula with a modal operator depends on the truth values of the formula in all visible worlds designated by the associated relation.

The temporal logic is a kind of modal logic, in which each world represents a state at a specific point in time and modalities specify temporal relations. We define an interpretation of the temporal logic as a finite or infinite sequence of interpretations of the first-order logic, which we call a time stream in this paper. Note that "time" in the temporal logic is discrete.

We list some of the temporal operators with their intuitive meanings:

- $\square p$  $p$ will be true forever from now
- $\boxdot p$  $p$ has been true until now
- $\Diamond p$  $p$ will become true at some time in the future
- $\Diamond p$  $p$ was true at some point in the past
- $\bigcirc p$  $p$ is true at the next point in time
- $\lozenge p$  $p$ was true at the previous point in time.

In the following let us concentrate on the temporal operators $\square$ and $\lozenge$; $\boxdot$ and $\bigcirc$ can be treated in the same way because their definitions are obtained by reversing past and future in the definitions of $\square$ and $\lozenge$. $\Diamond$ and $\Diamond$ can be defined as $\sim \square \sim$ and $\sim \lozenge \sim$, respectively. Let $S = (w(0), w(1), \ldots)$ be an interpretation; let $f$ be a formula. Given a world $w(i)$ contained in $S$ and an assignment $\pi$ of variables, we define truth values of $\square f$ and $\lozenge f$ as

- $\square f$ is true at $w(i)$ in $S$ with $\pi$ iff $f$ is true at $w(j)$ in $S$ with $\pi$ for all $j \geq i$
- $\lozenge f$ is true at $w(i)$ in $S$ with $\pi$ iff $i > 0$ and $f$ is true at $w(i-1)$ in $S$ with $\pi$.

We write

- $S, w(i), \pi \models f$ if $f$ is true at $w(i)$ in $S$ with $\pi$,
- $S, w(i) \models f$ if $S, w(i), \pi \models f$ for all $\pi$, and
- $S \models f$ if $S, w(i) \models f$ for all $w(i)$ in $S$. 
and \( \not\models \) denotes a negation of \( \models \). \( f \) is said to be valid in \( S \) if \( S \models f \). A set of formulas \( A \) is also said to be valid in \( S \) if all formulas in \( A \) is valid in \( S \), and we write \( S \models A \).

An interpretation \( S' \) is called a substream of \( S \) when

\[
S' = (w(i_0), w(i_1), \ldots), \quad \text{where } j < j' \text{ implies } i_j < i_{j'}.
\]

Let \( f \) be a formula which does not contain any free variable. \( S' \) is called a time-extraction or simply an extraction of \( S \) regarding to \( f \) when \( S' \) is a substream of \( S \) and \( w(i_j) \in S' \) iff \( S, w(i_j) \models f \) (see Fig. 1). We call \( f \) a key of the extraction. \( S | f \) denotes an extraction of \( S \) regarding \( f \).

Suppose we have a formula interpreted in one extraction and a formula in another extraction. When we examine the relationships between these two formulas, it would be inconvenient if we had to treat them in the separate models connected to the universal time streams via time-extraction. Conversely, if it is possible to "bring back" the formulas to the universal time stream, we will be able to deal with a number of formulas, each of which is interpreted in a respective extraction. As shown below, if a set of formulas \( A \) which is interpreted in \( S | f \) is given, we can give a set of formulas \( \langle f, A \rangle \) in \( S \), which is a counterpart of \( A \). We can also give a condition \( \llangle f, A \rrangle \) which guarantees equivalence between \( A \) and \( \langle f, A \rangle \). Note that \( \llangle f, A \rrangle \) depends on both \( f \) and \( A \). We call \( \langle f, A \rangle \) and \( \llangle f, A \rrangle \) as an embedding and an anchor of \( A \) regarding \( f \).

To prepare for the definition of \( \langle f, A \rangle \) and \( \llangle f, A \rrangle \), we define \( \langle f, g \rangle \) for formulas \( f \) and \( g \) recursively as

1. \( \langle f, g \rangle = g \) if \( g \) is an atomic formula
2. \( \langle f, g \rangle = \neg \langle f, g' \rangle \) if \( g = \neg g' \)
3. \( \langle f, g \rangle = \langle f, g' \rangle \lor \langle f, g'' \rangle \) if \( g = g' \lor g'' \)
4. \( \langle f, g \rangle = \exists x \langle f, g' \rangle \) if \( g = \exists x g' \)
5. \( \langle f, g \rangle = \Box (f \rightarrow \langle f, g' \rangle) \) if \( g = \Box g' \)
6. \( \langle f, g \rangle = \bullet p(x_1, \ldots, x_n) \) if \( g = \bullet g' \),

where \( p \) is a new predicate and \( x_1, \ldots, x_n \) are all free variables in \( g' \).

We call the predicate \( p \) introduced in part 6 as a status predicate for \( g' \). A different status predicate is assigned to another occurrence of the same subformula \( g' \). The status predicates are distinguished from other predicates.

Now we define \( \langle f, A \rangle \) and \( \llangle f, A \rrangle \):

\[
\langle f, A \rangle = \bigcup_{g \in A} \{ f \rightarrow \langle f, g \rangle \}.
\]
Let \( p \) be a status predicate for \( g' \), where \( \Diamond g' \) is a subformula occurring in \( A \). Then \( \langle f, A \rangle \) contains

\[
\begin{align*}
f &\to (p(x_1, \ldots, x_n) \equiv \langle f, g' \rangle) \\
\neg f &\to (p(x_1, \ldots, x_n) \equiv \Diamond p(x_1, \ldots, x_n)).
\end{align*}
\]

No other element is contained in \( \langle f, A \rangle \).

**Lemma 1.** Let \( S \models \langle f, A \rangle \), \( g_0 \in A \). For every subformula \( g \) occurring in \( g_0 \) and every assignment \( \pi \),

\[
S, w(i), \pi \models \langle f, g \rangle \quad \text{iff} \quad S \mid f, w(i), \pi \models g.
\]

**Proof.** Induction on construction of \( g \).

(1) If \( g \) is an atomic formula, it is trivial because \( \langle f, g \rangle = g \).

(2) \( g = \neg g' \):

\[
S \mid f, w(i), \pi \models \neg g' \quad \text{iff} \quad S \mid f, w(i), \pi \not\models g' \\
\quad \text{iff} \quad S, w(i), \pi \not\models \langle f, g' \rangle \\
\quad \text{iff} \quad S, w(i), \pi \not\models \neg \langle f, g' \rangle.
\]

(3) \( g = g' \lor g'' \).

(4) \( g = \exists x g' \). They are easily seen similarly to (2).
(5) \( g = \Box g'. \)

\((\Leftarrow)\) \( S \models f, w(i), \pi \models \Box g' \) iff \( S \models f, w(i), \pi \models g' \) for all \( k \geq j \). We will show that this implies \( S, w(k'), \pi \models f \rightarrow \langle f, g' \rangle \) for all \( k' \geq i_j \), i.e., \( S, w(i_j), \pi \models \Box (f \rightarrow \langle f, g' \rangle) \). Suppose it does not hold. There exists \( k'' \geq i_j \) such that \( S, w(k''), \pi \not\models f \rightarrow \langle f, g' \rangle \). Because \( f \) has no free variables, \( S, w(k'') \models f \) and \( S, w(k'') \not\models \langle f, g' \rangle \). By the definition of extraction and the hypothesis of induction, \( S \models f \) contains \( w(k'') \), and \( S \models f, w(k''), \pi \not\models g' \). This contradicts the hypothesis. Therefore \( S \models f, w(i_j), \pi \models \Box g' \) implies \( S, w(i_j), \pi \models \Box (f \rightarrow \langle f, g' \rangle) \).

\((\Rightarrow)\) It can be seen similarly to the proof of opposite direction that the negation of \( S \models f, w(i_j), \pi \not\models g' \) leads to contradiction.

(6) \( g = \bullet g' \). Let \( p(x_1, ..., x_n) \) be a status predicate for \( g \).

Case 1. \( j = 0 \). By the definition of \( \bullet \), \( S \models f, w(i_0) \not\models \bullet g' \). It is all right if \( i_0 = 0 \) because \( S, w(0) \not\models \bullet p(x_1, ..., x_n) \), too. Suppose \( i_0 > 0 \). By the definition of extraction, \( S, w(i') \not\models f \) for all \( i' < i_0 \). Since \( S \models \langle f, A \rangle \),

\( S, w(i') \models p(x_1, ..., x_n) \equiv \bullet p(x_1, ..., x_n) \) for all \( i' < i_0 \).

This and \( S, w(0) \not\models \bullet p(x_1, ..., x_n) \) lead to \( S, w(i_0) \not\models \bullet p(x_1, ..., x_n) \). Therefore \( S, w(i_0), \pi \not\models \bullet p(x_1, ..., x_n) \).

Case 2. \( j > 0 \). \( S \models f, w(i_j), \pi \models \bullet g \) iff \( S \models f, w(i_{j-1}), \pi \models g' \) iff \( S, w(i_{j-1}), \pi \models \langle f, g' \rangle \). Since \( S \models \langle f, A \rangle \),

\( S, w(i_{j-1}) \models p(x_1, ..., x_n) \equiv \langle f, g' \rangle \).

\( S, w(i') \models p(x_1, ..., x_n) \equiv \bullet p(x_1, ..., x_n) \) for all \( i' \) such that \( i_{j-1} < i' < i_j \).

Therefore \( S \models f, w(i_j), \pi \models \bullet g' \) iff \( S, w(i_j), \pi \models \bullet p(x_1, ..., x_n) \).

**Theorem 1.** Assume \( S \models \langle f, A \rangle \). Then \( S \models \langle f, A \rangle \) iff \( S \models f \models A \).

**Proof.** It is clear from Lemma 1.

**Example 1** (Description of an interphone). Let us try to describe a simple interphone which has a three-digit phone number. The events are represented by the following predicates:

- **on-hook** put down the receiver
- **off-hook** take up the receiver
- **dial(n)** dial a digit \( n \).

We assume that no more than one event happens at the same time. The action to call the interphone whose three-digit number is \( x \), is represented by a predicate \( \text{call}(x) \).
We can observe the following facts about the behavior of this interphone.

We repeat to "take up" and "put down" the receiver by turn. (FACT 1)

Take up the receiver, dial a digit three times, then a call takes place. (FACT 2)

If we use atnext and until, the facts above are formulated as:

FACT 1.

\[ \text{on-hook} \rightarrow \text{off-hook} \text{ atnext} (\text{on-hook} \lor \text{off-hook}) \]
\[ \text{off-hook} \rightarrow \text{on-hook} \text{ atnext} (\text{on-hook} \lor \text{off-hook}). \]

FACT 2.

\[ \text{off-hook} \rightarrow \text{state1} \text{ until} (\text{on-hook} \lor \exists n(\text{dial}(n))) \]
\[ \text{state1} \land \text{dial}(n_1) \rightarrow \text{state2}(n_1) \text{ until} (\text{on-hook} \lor \exists n(\text{dial}(n))) \]
\[ \text{state2}(x) \land \text{dial}(n_2) \rightarrow \text{state3}(x \ast 10 + n_2) \text{ until} (\text{on-hook} \lor \exists n(\text{dial}(n))) \]
\[ \text{state3}(x) \land \text{dial}(n_3) \rightarrow \text{call}(x \ast 10 + n_3). \]

In case of time-extraction:

FACT 1. The following formulas hold in the extracted time stream regarding on-hook \lor off-hook:

\[ \bullet \text{on-hook} \rightarrow \text{off-hook} \] (1.1)
\[ \bullet \text{off-hook} \rightarrow \text{on-hook}. \] (1.2)

FACT 2. The following formulas hold in the extracted time stream regarding on-hook \lor off-hook \lor \exists n(\text{dial}(n)):

\[ \bullet \bullet \bullet \text{off-hook} \land \bullet \bullet \text{dial}(n_1) \land \bullet \text{dial}(n_2) \land \text{dial}(n_3) \]
\[ \rightarrow \text{call}(100n_1 + 10n_2 + n_3) \] (1.3)
\[ \sim (\bullet \bullet \bullet \text{off-hook} \land \bullet \bullet \text{dial}(n_1) \land \bullet \text{dial}(n_2) \land \text{dial}(n_3)) \]
\[ \rightarrow \sim \text{call}(x). \] (1.4)

Comparing these two methods, the latter is considered to be a more direct and essential way of expression.

Suppose we put down the receiver before we dial the third digit, then (1.3) has nothing to do with the action of the interphone; namely, (1.3) is true regardless of the truth value of call(x). In this case, the events occur in the order of

\[ \text{off-hook, dial}(n_1), \text{dial}(n_2), \text{on-hook, dial}(n_3). \]

Therefore the left side of \( \rightarrow \) becomes false.
Now we will rewrite (1.1) and (1.2) using embeddings and anchors. First, we introduce status predicates $p_1$ and $p_2$: 

\begin{align*}
\langle \text{on-hook} \lor \text{off-hook}, \bullet \text{on-hook}\rangle &= \bullet p_1 \\
\langle \text{on-hook} \lor \text{off-hook}, \bullet \text{on-hook} \rightarrow \text{off-hook}\rangle &= \bullet p_1 \rightarrow \text{off-hook} \\
\langle \text{on-hook} \lor \text{off-hook}, \bullet \text{off-hook}\rangle &= \bullet p_2 \\
\langle \text{on-hook} \lor \text{off-hook}, \bullet \text{off-hook} \rightarrow \text{on-hook}\rangle &= \bullet p_2 \rightarrow \text{on-hook}.
\end{align*}

Then, let $A$ be \{(1.1), (1.2)\},

\begin{align*}
\langle \text{on-hook} \lor \text{off-hook}, A \rangle &= \{\text{on-hook} \lor \text{off-hook} \rightarrow \bullet p_1 \rightarrow \text{off-hook}, \\
&\text{on-hook} \lor \text{off-hook} \rightarrow \bullet p_2 \rightarrow \text{on-hook}\}; \tag{1.5}
\end{align*}

\begin{align*}
\langle \text{on-hook} \lor \text{off-hook}, A \rangle &= \{\text{on-hook} \lor \text{off-hook} \rightarrow (p_1 \equiv \text{on-hook}), \\
&\sim \text{on-hook} \land \sim \text{off-hook} \rightarrow (p_1 \equiv \bullet p_1), \\
&\text{on-hook} \lor \text{off-hook} \rightarrow (p_2 \equiv \text{on-hook}), \\
&\sim \text{on-hook} \land \sim \text{off-hook} \rightarrow (p_2 \equiv \bullet p_2)\}. \tag{1.6}
\end{align*}

Equations (1.5) and (1.6) form a sufficient condition for (1.1) and (1.2) to be valid in the extraction regarding $\text{on-hook} \lor \text{off-hook}$.

3. TEMPORAL PROLOG

In this section, we will apply the notion of time-extraction to the temporal prolog (TP for short) which is a logic programming language based on the temporal logic.

3.1. The Semantics of TP

Let us take a brief look at the model-theoretic semantics of TP. ([10] describes it in detail.) A program of TP is a set of formulas of first-order temporal logic with various temporal operators with some syntactic restrictions. These restrictions make it possible for every formula to be transformed to a normal formula like 

$$a_1 \land a_2 \land \cdots \land a_n \rightarrow b,$$

where $a_1, \ldots, a_n$ is an atomic formula or its negation, possibly preceded by some $\bullet$'s, and $b$ is an atomic formula. In the following, we assume that every formula has already been transformed to a normal formula.

A model of a program $A$ of TP is an infinite sequence of subsets of the Herbrand base $W(A)$ which is a set of all ground atomic formulas, where all terms belong to
the Herbrand universe. We define an order among models of $A$ as follows: A tuple $(W_0, \ldots, W_k)$ is a division of $W(A)$ iff $W_0, \ldots, W_k$ is disjoint and $\bigcup W_i = W(A)$. Let $(W_0, \ldots, W_k)$ be a division of $W(A)$, and $L = (v(0), v(1), \ldots)$, $M = (w(0), w(1), \ldots)$ be models of $A$:

$$L > M \quad \text{iff} \quad \begin{align*}
\text{there exist natural numbers } m \text{ and } n \text{ such that } \\
v(l) = w(l) & \quad \text{for all } l < m, \\
v(m) \cap W_i = w(m) \cap W_i & \quad \text{for all } i < n \\
w(m) \cap W_n & \text{ is properly included in } v(m) \cap W_n.
\end{align*}$$

The semantics of $A$ is defined as the least model of $A$, where the division is provided by the dependency relation among the predicates in $A$. After all, an execution of $A$ yields an infinite sequence of worlds which is the least model of $A$.

3.2. Time-Extraction for TP

Programming in TP is to prescribe a time stream by means of formulas which should be valid in it. Then, is it possible to make use of formulas which will be valid in an extracted time stream in the same way? For instance, suppose we have two programs $P_1$ and $P_2$ written in TP, only one of which can run at a time. Instead of rewriting $P_1$ and $P_2$ for the purpose of process switching, it will be desirable to treat $P_i$ as a program in the extracted time stream regarding process($i$), where the predicate process($i$) is supposed to be set or reset for $i = 1, 2$ by some scheduler. In order to accomplish this, we have to define a model of such programs. For a pair of a program $A$ and a key $f$, which we call a pseudo program, it is natural to define its model as an interpretation $S$ such that $S \models f$ is a model of $A$ in the sense of TP. We extend this definition to multiple pseudo programs. Let $f_i$ be a formula and $A_i$ be a program for $1 \leq i \leq m$. For a given $P = \{(f_i, A_i) \mid 1 \leq i \leq m\}$, $S$ is its model iff $S \models f_i \models A_i$ for all $i$. As described below, the order among the models of $P$ can be induced from the one provided by TP. Therefore we can define the semantics of $P$ by the least model in regard to that order.

However, it would be difficult to execute pseudo programs efficiently if the definition above was directly used. In the previous section, we defined the embeddings and the anchors which we could regard as substitutes that are easier to deal with. It can be expected that efficient executions will be available if pseudo programs are transformed to the equivalent programs in the same way as logical formulas. Nevertheless there exists a problem that $\langle f, A \rangle$ cannot be transformed to a normal formula because $\bullet$ occurs in the right side of $\rightarrow$. Hence we define a weak anchor $WA(f, A)$ instead of $\langle f, A \rangle$ so that we can manage to compose a program by embeddings and weak anchors. Let $p(x_1, \ldots, x_n)$ be a status predicate for $g'$, where $\bullet g'$ is a subformula in $A$. Then $WA(f, A)$ contains
\[ f \rightarrow ((f, g') \rightarrow p(x_1, ..., x_n)) \]
\[ \neg f \rightarrow (\neg p(x_1, ..., x_n) \rightarrow p(x_1, ..., x_n)). \]
No other element is contained in \( WA(f, A). \)

**Theorem 2.** If \( S \models WA(f, A) \) holds, \( S \models f \vdash A \) implies \( S \models \langle f, A \rangle. \)

**Proof.** It is easy to rewrite the proof of Theorem 1. \[ \] 

For interpretations \( S \) and \( S' \), \( S \simeq S' \) iff \( S \) and \( S' \) are identical except for part of the status predicates.

**Lemma 2.** \( S \models f_k \vdash A_k \) for \( 1 \leq k \leq m \) implies there exists \( S' \) such that \( S' \simeq S \) and
\( S' \models \langle f_k, A_k \rangle \cup WA(f_k, A_k) \) for all \( k. \)

**Proof.** We will make \( S' \) by changing the part of status predicates in \( S. \) In \( S' \), the truth values of predicates other than status predicates are the same as \( S. \) The interpretation of a status predicate \( p(x_1, ..., x_n) \) for \( g \) is defined as follows, where we assume that interpretations of status predicates that occurred in \( \langle f_k, g \rangle \) have already been defined. Let \( S' = (w(0), w(1), ...), S' \vdash f_k = (w(i_0), w(i_1), ...), \) and \( t_1, ..., t_n \) be ground terms:
\[ S', w(j) \not\models p(t_1, ..., t_n) \quad \text{if} \quad j < i_0 \]
\[ S', w(j) \models p(t_1, ..., t_n) \quad \text{iff} \quad S', w(i_n) \models \langle f_k, g(t_1, ..., t_n) \rangle \quad \text{if} \quad i_n \leq j < i_{n+1} \text{ for some } \ n. \]

It is easy to see that \( S' \models WA(f_k, A_k). \) Since the status predicates for the subformulas that occurred in \( A_k \) are different from each other, the operations above have no contradiction. \( S \models f_k \vdash A_k, \ S' \simeq S, \) and the fact that there is no status predicate in \( A_k \) lead to \( S' \models f_k \vdash A_k. \) By Theorem 2, \( S' \models \langle f_k, A_k \rangle. \)

**Lemma 3.** If a program \( \bigcup_{1 \leq k \leq m} (\langle f_k, A_k \rangle \cup WA(f_k, A_k)) \) has the least model \( M, \)
\[ M \models \langle f_k, A_k \rangle \quad \text{for all } \ k. \]

**Proof.** Suppose that there exists some \( k \) such that \( M \not\models \langle f_k, A_k \rangle \). Let \( M = (w(0), w(1), w(2), ...). \) There exists a status predicate \( p, \) a natural number \( i, \) and ground terms \( t_1, ..., t_n \) such that
\[ M, w(i) \not\models f_k \rightarrow (\langle f_k, g' \rangle \equiv p(t_1, ..., t_n)), \]
or
\[ M, w(i) \not\models \neg f_k \rightarrow (p(t_1, ..., t_n) \equiv p(t_1, ..., t_n)). \]
On the other hand, by $M, w(i) \models WA(f_k, A_k)$,

$$M, w(i) \models f_k \rightarrow (\langle f_k, g' \rangle \rightarrow p(t_1, ..., t_n))$$

$$M, w(i) \models \neg f_k \rightarrow (\bullet p(t_1, ..., t_n) \rightarrow p(t_1, ..., t_n)).$$

We find for $p(t_1, ..., t_n) \in w(i)$ in either case that $f_k$ is true or false. Suppose $M'$ is derived from $M$ by removing $p(t_1, ..., t_n)$ from $w(i)$. Clearly $M' < M$. Remembering the form of the normal formula, $\bullet$ occurs only in the left side of $\rightarrow$, so status predicates also occurred only in the left side in embeddings. This means that to remove $p(t_1, ..., t_n)$ from $w(i)$ does not falsify any formulas in $\langle f_k, A_k \rangle$ and $WA(f_k, A_k)$. Therefore $M' \models \langle f_k, A_k \rangle \cup WA(f_k, A_k)$, but this contradicts the hypothesis that $M$ is the least.

**Theorem 3.** If $\bigcup_{1 \leq i \leq m} (\langle f_i, A_i \rangle \cup WA(f_i, A_i))$ has the least model $M$ in the sense of TP, then $\{(f_i, A_i) \mid 1 \leq i \leq m\}$ also has the least model $S$ with the same order and $S \approx M$.

**Proof.** Let $E, F$ be a set of all models of $\{(f_i, A_i)\}$ and $\bigcup (\langle f_i, A_i \rangle \cup WA(f_i, A_i))$, respectively. Since $F$ has an order in the sense of TP and every predicate occurred in $E$ also occurred in $F$, $E$ has the same order. For $N \in E \cup F$, we define $[N]$ as an interpretation which is made from $N$ by changing all the status predicates to be false. We can obtain the following proposition easily:

$$N \geq [N]$$

$$N \leq N' \implies [N] \leq [N']$$

$$N \approx N' \implies [N] = [N']$$

$$S \in E \implies [S] \in E.$$  

By Lemma 3 and Theorem 1, $M \models f_i \models A_i$. Therefore $M \in E$. By (2.4), $[M] \in E$. Put $S = [M]$; then we have to show that $S$ is the least in $E$. For every $S' \in E$, there exists $M' \in F$ such that $S' \cap M'$ by Lemma 2. $M' \geq M$, (2.1), (2.2), and (2.3) lead to $S' \geq [S'] = [M'] = [M] = S$. Therefore $S$ is the least in $E$.

We define $S$ in Theorem 3 as the semantics of the pseudo programs. Since status predicates do not occur in the original program, we can consider that they have nothing to do with the semantics. Then Theorem 3 states that a set of pseudo programs can be transformed to the equivalent program.

**Example 2 (Controlling a switchboard).** Now we will try to control a switchboard for the interphones which we described in the Example 1. It is connected to a large number of interphones on which various events happen asynchronously. The simplest way to deal with such events is to make a process for each interphone to watch the dial or the receiver, and a process for each pair of interphones to control a connection between them.
Let us conceive of a group of pseudo programs carrying out the same job for different objects. Such a group can be represented by a syntactically single component whose key includes parameters, and each pseudo program will be obtained by instantiation of the parameters. This means that they are isolated from each other on different time streams so that unnecessary interference can be avoided. In this example, the program which deals with an abstract interphone will work in extracted time streams regarding keys for individual interphones. In the following, let $a$ and $b$ be parameters occurring in the key formulas, which represent the phone numbers. Note that this situation is similar to creating several processes from a single program in the usual time-sharing environments.

We assume that the events which happen on the interphones such as on-hook, off-hook, and dial do not happen at the same time even if they happen on different interphones. Predicates defined in Example 1 are modified to include one's own phone number.

A process for watching the dial of an interphone with a parametrized key of \(\text{on-hook}(a) \lor \text{off-hook}(a) \lor \text{dial}(a, 0) \lor \cdots \lor \text{dial}(a, 9)\):

\[
\begin{align*}
\bullet \bullet \bullet \text{off-hook}(x) & \land \bullet \bullet \text{dial}(x, n_1) \land \bullet \text{dial}(x, n_2) \land \text{dial}(x, n_3) \\
\rightarrow \text{call}(x, 100n_1 + 10n_2 + n_3).
\end{align*}
\]

We may regard the following two pseudo programs, which deal with connection between interphones, as either a single process consisting of two modules, or two processes which work in cooperation.

Regarding \(\text{call}(a, b) \lor \text{on-hook}(b) \lor \text{off-hook}(b)\):

\[
\begin{align*}
\bullet \text{on-hook}(y) \land \text{call}(x, y) & \rightarrow \text{ring}(y) \land \text{calling-tone}(x) \\
\bullet \text{off-hook}(y) \land \text{call}(x, y) & \rightarrow \text{busy-tone}(x) \\
\text{call}(x, y) \land \text{call}(x, y) & \rightarrow (\bullet \text{busy-tone}(x) \rightarrow \text{busy-tone}(x)) \\
& \land (\bullet \text{ring}(y) \rightarrow \text{ring}(y)) \\
& \land (\bullet \text{calling-tone}(x) \rightarrow \text{calling-tone}(x)).
\end{align*}
\]

Regarding \((\text{call}(a, b) \land \text{ring}(b)) \lor \text{on-hook}(a) \lor \text{off-hook}(b)\):

\[
\begin{align*}
\bullet \text{call}(x, y) \land \text{on-hook}(x) & \rightarrow \text{quiet}(y) \\
\bullet \text{call}(x, y) \land \text{off-hook}(y) & \rightarrow \text{connect}(x) \land \text{connect}(y).
\end{align*}
\]

It is easy to convince oneself of the validity of the pseudo programs above since they are related to only one or two interphones. However, it is not realistic to directly use them because it would yield an enormous number of processes which could not possibly run on a real machine. Therefore we must transform them to a program so that they can run as a single process. Since status predicates are
different for each value of parameters, they play the role of an array which keeps the internal states.

Now the program contains many formulas instead of many processes, still being difficult to run if the number of interphones is very large. However, as in this example, if a parameter varies all over the data domain, we can reduce a number of the formulas by replacing the parameter with a variable, because free variables are universally quantified. For instance,

\[ WA(\text{call}(a, b) \lor \text{on-hook}(b) \lor \text{off-hook}(b), \]
\[ \{ \text{on-hook}(y) \land \text{call}(x, y) \rightarrow \text{ring}(y) \land \text{calling-tone}(x) \} \]

consists of the following two formulas for every \( a \) and \( b \).

\[ \text{call}(a, b) \lor \text{on-hook}(b) \lor \text{off-hook}(b) \rightarrow \text{on-hook}(y) \rightarrow p-a-b(y) \]
\[ \sim (\text{call}(a, b) \lor \text{on-hook}(b) \lor \text{off-hook}(b)) \rightarrow \text{on-hook}(y) \rightarrow p-a-b(y). \]

Replace each occurrence of \( p-a-b(x) \) by \( p(a, b, x) \):

\[ \text{call}(a, b) \lor \text{on-hook}(b) \lor \text{off-hook}(b) \rightarrow \text{on-hook}(y) \rightarrow p(a, b, y) \]
\[ \sim (\text{call}(a, b) \lor \text{on-hook}(b) \lor \text{off-hook}(b)) \rightarrow \text{on-hook}(y) \rightarrow p(a, b, y). \]

There exist two formulas above for every pair of \( a \) and \( b \). Now we can replace all of such formulas by two formulas:

\[ \text{call}(v, w) \lor \text{on-hook}(w) \lor \text{off-hook}(w) \rightarrow \text{on-hook}(y) \rightarrow p(v, w, y) \]
\[ \sim (\text{call}(v, w) \lor \text{on-hook}(w) \lor \text{off-hook}(w)) \rightarrow \text{on-hook}(y) \rightarrow p(v, w, y). \]

In addition, we can improve the program using information from other parts of the program. By the hypothesis that \( \text{dial}, \text{on-hook}, \) and \( \text{off-hook} \) do not happen at the same time, and the fact that \( \text{call} \) is true only if \( \text{dial} \) is true, we find that \( \text{call}, \text{on-hook}, \) and \( \text{off-hook} \) do not happen at the same time. Therefore the formulas above are transformed to

\[ \text{on-hook}(w) \rightarrow p(v, w, w) \]
\[ \sim (\text{call}(v, w) \lor \text{on-hook}(w) \lor \text{off-hook}(w)) \rightarrow \text{on-hook}(y) \rightarrow p(v, w, y) \]

in which \( v \) and \( y \) are redundant. The final version is

\[ \text{on-hook}(w) \rightarrow p(w) \]
\[ \sim (\text{call}(v, w) \lor \text{on-hook}(w) \lor \text{off-hook}(w)) \rightarrow \text{on-hook}(y) \rightarrow p(w). \]

Let us see the comparison between the program derived from the pseudo program above and the one written with \( \text{atnext} \) and \( \text{until} \). First, the formula

\[ \text{on-hook}(y) \land \text{call}(x, y) \rightarrow \text{ring}(y) \land \text{calling-tone}(x) \]
in the pseudo program is transformed to
\[
\text{on-hook}(w) \to p(w)
\]
\[
\sim (\text{call}(v,w) \lor \text{on-hook}(w) \lor \text{off-hook}(w)) \to \bullet p(w) \to p(w).
\]

On the other hand, we can write the same part of the program with the \textit{until} operator:
\[
\text{on-hook}(y) \to \text{is-on}(y) \text{ until } \text{off-hook}(y)
\]
\[
\text{is-on}(y) \land \text{call}(x,y) \to \text{ring}(y) \land \text{calling-tone}(x).
\]

The former one, which is derived from the pseudo program, can be obtained by rewriting the latter one using the \(\bullet\) operator instead of the \textit{until} operator. It means that almost the same efficiency as \textit{until} is available after the pseudo programs are transformed into a single process.

4. \textbf{Comparison with Other Methods}

There are several extensions of the classical temporal logic such as binary temporal operators and the multiprocess network logic. In this section, we compare time-extraction with those approaches from the point of applying them to programming.

Binary temporal operators, including \textit{atnext}, are powerful methods to describe event sequences that spread over several time points. However, unlimited use of them often causes very complex relationships between formulas, even incomprehensible ones in the worst case. As to extraction, a sort of structured programming is achieved since a time stream is separated from others. Besides, we have already seen that extraction can provide more essential expressions than \textit{atnext} can in the Example 1 in Section 2. We must also consider expressiveness, but the \(\bullet\) operator has enough expressiveness for programming because a legal program of the temporal prolog can be rewritten without any temporal operators but \(\bullet\) operators.

The multiprocess network logic [8] is similar to extraction since both of them deal with the notion of process. However, they are quite different in the way of introducing it to temporal logic. The multiprocess network logic considers a process as a sequence of worlds connected by temporal modalities, where spatial modalities are independent from temporal ones and communication between processes is performed through them. Such a separate treatment of time and process may be easier to understand, but asynchronous process must communicate carefully, synchronizing each other's state of processing, because time passes invariantly even if a process must wait for an event from another process. On the other hand, extraction can provide a local process time. Synchronization is implicitly performed by the truth value of the key formula. Moreover, in the multiprocess network logic,
configuration of processes is determined by a program because it is explicitly written as spatial modalities, whereas extraction does not give such an explicit distinction between communication and internal processing, and therefore we can merge two processes into a single process as we saw in the previous section.

5. Generalization and Application

5.1. Other Modal Logics

In order to extend the application of time-extraction to other modal logics, we generalize it to the notion which provides extra models that are made from the original one, using a given key formula. Worlds in which the key formula holds are collected, and accessibility relations among them are induced in the following way: Let \( w_i \) be a world for every \( i \) and let \( F \) be a set of pairs of worlds associated to a modality. Then a new relation \( G \) induced by \( F \) consists of \( (w_1, w_n) \) such that \( (w_1, w_2), (w_2, w_3), ..., (w_{n-1}, w_n) \in F \), and the key formula holds at \( w_1 \) and \( w_n \), and does not hold at \( w_2, ..., w_{n-1} \).

For example, in the case of \( S4 \), it is almost trivial to see that

\[ \square g \text{ holds in the extracted model regarding a key } f \]

is equivalent to

\[ \square (f \rightarrow g) \text{ holds in the original model.} \]

5.2. Application for a Distributed System

Time-extraction can be helpful for designing and programming distributed systems. The most important decision in the design of distributed systems is how to divide a job into processes on which communication and synchronization depend. (It is also important in the design of single processor systems, but the cost of process communications on a single processor is not as expensive as multi-processor communications.) In the usual design method, this decision takes place at the earliest stage; therefore it is very expensive to fix mistakes that are found in later stages, or to adapt the program to future changes of the hardware definition.

Using extraction, design and programming will be like the following: First we write very small pieces of the program in appropriate time streams, which build up a model of the programmer's view and have nothing to do with the configuration of the hardware. Then these fragments are combined and executed in a debugging environment to ensure their correctness. Next we assign the fragments to one of the processors and transform them to a single process. Finally it is optimized and compiled to a certain machine language. The decision of dividing a job takes place after a program is written at which stage detailed information about the modularity is available. Furthermore, changes of the hardware definition require only re-assignment to the processors and compilation.
6. Conclusion

We have introduced the notion of time-extraction for the temporal logic in order to simplify descriptions of sequences of events, especially when at most one event happens at a time. Another aim of extraction is to make an extra time stream which can be regarded as the local process time. It can allow a more natural representation of the notion of process than a model which has a single time stream. For given formulas with extraction, it is possible to give counterpart formulas without extraction and conditions which guarantee the equivalence between them.

We have also applied time-extraction to the temporal prolog. We defined the semantics of programs with time-extraction. Programs with time-extraction can be transformed to a single program without time-extraction, which preserves the equivalence of semantics. This enables us to write a program in its own time stream.

One may think that it is not practical to make the assumption in both examples that no more than one event happens at the same time. In real machines, however, one processor can receive one interrupt signal at a time. (Nesting of interrupts is another problem.) In the case of multi-processor systems, it is more understandable to deal with one event at a time because we can regard the system as a single processor, and moreover, only atomic execution takes place at each point in time. One solution to have both describability and simplicity is to use a "beginning" and an "ending" of the event instead of the event itself, although it needs further consideration.

Acknowledgments

The author expresses his deep gratitude to Professor Reiji Nakajima for his appropriate advice. The author also thanks Mr. Takashi Sakuragawa and Mr. Naoyuki Niide for their useful comments on an earlier draft of this paper.

References


