数理解析研究所講究録
第655巻 1988年 167-185

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

京大数理研 服部隆志 (Takashi Hattori)

Abstract: Temporal logic is useful to describe 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 time stream. In distributed environments, it can allow simple descriptions of processes to deal each process as if it had its own proper time stream where a proper time stream, called 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.

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, the temporal logic is more useful than the ordinal logic because the former can describe the notion of time explicitly. There are still some difficulties, however, when we use the 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 the temporal logic is discrete, in which each period is not necessary 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 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 process has not yet been formalized in the temporal 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 synchronize each other at some points 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 about 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 operator in logic, we may write

$$\bullet p \wedge q \rightarrow r.$$

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

$$p \rightarrow a$$
$$\bullet a \wedge \sim q \rightarrow a$$
$$\bullet a \wedge q \rightarrow r$$

One way to prevent such redundancy is to introduce a new temporal operator such as *atnext*, *until* to specify complex temporal relations. 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 *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 relationship between formulas in 'virtual' and 'real' time streams. Next we apply time-extraction to Temporal Prolog [9] and give an algorithm which transforms a program using extraction to an equivalent and possibly more efficient program without extraction. 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 in the point that it has some modal operators in addition to logical ones. Its model consists of a set of possibility worlds (worlds for short) and a set of visibility relations. Each world is an interpretation of predicate and function symbols, which is exactly same as the model of the first order logic. The truth value of a formula can be different in each world. 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 the 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 *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.

$\Box p$ p will be true forever from now

$\blacksquare p$ p has been true until now

$\Diamond p$	p will become true at some time in the future
$\blacklozenge p$	p was true at some point in the past
$\bigcirc p$	p is true at the next point in time
$\bullet p$	p was true at the previous point in time

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

$\Box f$ is true at $w(i)$ in S with π iff

f is true at $w(j)$ in S with π for all $j \geq i$

$\bullet f$ is true at $w(i)$ in S with π iff

$i>0$ and f is true at $w(i\text{-}1)$ in S with π

We write

$S,w(i),\pi \models f$	if f is true at $w(i)$ in S with π
$S,w(i) \models f$	if $S,w(i),\pi \models f$ for all π, 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 $S \models f$.

An interpretation S' is called a *substream* of S when

$S' = (w(i_0),w(i_1),\ldots)$ where $j<j'$ 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 $S,w(i_j) \models f$. We call f as a *key* of the extraction. $S\backslash f$ denotes an extraction of S regarding to 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 $\ll f,A\gg$ which guarantees equivalence between A and $\langle f,A\rangle$. Note that $\ll f,A\gg$ depends on both f and A. We call $\langle f,A\rangle$ and $\ll f,A\gg$ as an *embedding* and an *anchor* of A regarding to f.

To prepare for the definition of $\langle f,A\rangle$ and $\ll f,A\gg$, we define $\langle f,g\rangle$ for formulas f and g recursively as follows:

1. $\langle f,g\rangle = g$	if g is an atomic formula
2. $\langle f,g\rangle = \sim\langle f,g'\rangle$	if $g = \sim g'$
3. $\langle f,g\rangle = \langle f,g'\rangle\vee\langle f,g''\rangle$	if $g = g'\vee g''$
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 free variables in g'.

We call the predicate p introduced in 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 $\ll f,A\gg$.

$$\langle f,A\rangle = \bigcup_{g\in A} \{f\rightarrow\langle f,g\rangle\}$$

Let p be a status predicate for g' where $\bullet g'$ is a subformula occurred in A. Then $\ll f,A \gg$ contains

$$f \rightarrow (p(x_1,\ldots,x_n) \equiv \langle f,g' \rangle)$$
$$\sim f \rightarrow (p(x_1,\ldots,x_n) \equiv \bullet p(x_1,\ldots,x_n))$$

No other element is contained in $\ll f,A \gg$.

Lemma 1 Let $S \vDash \ll f,A \gg$, $g_0 \in A$. For all subformula g occurred in g_0 and all assignment π,

$$S,w(i_j),\pi \vDash \langle f,g \rangle \text{ iff } S|f,w(i_j),\pi \vDash g$$

Proof) Induction on construction of g.

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

(2) $g = \sim g'$.

$S|f,w(i_j),\pi \vDash \sim g'$
iff $S|f,w(i_j),\pi \nvDash g'$
iff $S,w(i_j),\pi \nvDash \langle f,g' \rangle$
iff $S,w(i_j),\pi \vDash \sim \langle f,g' \rangle$.

(3) $g = g' | g''$, (4) $g = \exists x g'$. They are easily seen similarly to (2).

(5) $g = \Box g'$.

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

(=>) It can be seen similarly to the proof of opposite direction that the negation of $S|f,w(i_j),\pi \vDash \Box g'$ 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 $\bullet$, $S|f,w(i_0) \not\models \bullet g'$. 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 $S \models \ll f,A \gg$,

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

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

$$S,w(i_{j-1}) \models p(x_1,...,x_n) \equiv \langle f,g' \rangle \text{ and}$$
$$S,w(i') \models p(x_1,...,x_n) \equiv \bullet p(x_1,...,x_n) \text{ for all } i' \text{ such that } i_{j-1}<i'<i_j.$$

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

Theorem 1 Assume $S \models \ll f,A \gg$. Then $S \models \langle f,A \rangle$ iff $S|f \models A$.

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 more than one events never happen in the same time. The action to call the interphone whose three-digit number is x, is represented by a predicate *call*(x).

* We repeat to "take up" and "put down" the receiver by turn.

The following formulas hold in the extracted time stream regarding to *on-hook* ∨ *off-hook*.

$$\bullet on\text{-}hook \rightarrow off\text{-}hook \tag{1.1}$$

$$\bullet off\text{-}hook \rightarrow on\text{-}hook \tag{1.2}$$

* Take up the receiver, dial a digit three times, then a *call* takes place.

The following formulas hold in the extracted time stream regarding to *on-hook* ∨ *off-hook* ∨ $\exists n(dial(n))$.

$$\bullet\bullet\bullet off\text{-}hook \wedge \bullet\bullet dial(n_1) \wedge \bullet dial(n_2) \wedge dial(n_3) \rightarrow call(100n_1+10n_2+n_3) \tag{1.3}$$

$$\sim(\bullet\bullet\bullet off\text{-}hook \wedge \bullet\bullet dial(n_1) \wedge \bullet dial(n_2) \wedge dial(n_3)) \rightarrow \sim call(x) \tag{1.4}$$

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

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

Therefore the left side of → 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.

$$\langle on\text{-}hook \vee off\text{-}hook,\ \bullet on\text{-}hook\rangle = \bullet p_1$$
$$\langle on\text{-}hook \vee off\text{-}hook,\ \bullet on\text{-}hook \rightarrow off\text{-}hook\rangle = \bullet p_1 \rightarrow off\text{-}hook$$

$$\langle on\text{-}hook \vee off\text{-}hook,\ \bullet off\text{-}hook\rangle = \bullet p_2$$
$$\langle on\text{-}hook \vee off\text{-}hook,\ \bullet off\text{-}hook \rightarrow on\text{-}hook\rangle = \bullet p_2 \rightarrow on\text{-}hook$$

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

$$\langle on\text{-}hook \vee off\text{-}hook,\ A\rangle =$$
$$\{on\text{-}hook \vee off\text{-}hook \rightarrow \bullet p_1 \rightarrow off\text{-}hook,$$
$$on\text{-}hook \vee off\text{-}hook \rightarrow \bullet p_2 \rightarrow on\text{-}hook\} \qquad (1.5)$$

$$\ll on\text{-}hook \vee off\text{-}hook,\ A\gg =$$
$$\{on\text{-}hook \vee off\text{-}hook \rightarrow (p_1 \equiv on\text{-}hook),$$
$$\sim on\text{-}hook \wedge \sim off\text{-}hook \rightarrow (p_1 \equiv \bullet p_1),$$
$$on\text{-}hook \vee off\text{-}hook \rightarrow (p_2 \equiv on\text{-}hook),$$
$$\sim on\text{-}hook \wedge \sim off\text{-}hook \rightarrow (p_2 \equiv \bullet p_2)\} \qquad (1.6)$$

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

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 details.) A program of TP is a set of formulas of the first order temporal logic with various temporal operators with some syntactic restrictions. These restrictions make it possible to every formula to be transformed to a normal formula which is like the following:

$$a_1 \wedge a_2 \wedge \ldots \wedge a_n \rightarrow b$$

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 $(W_0,...,W_k)$ is a division of $W(A)$ iff $W_0,...,W_k$ is disjoint and $\cup W_i = W(A)$. Let $(W_0,...,W_k)$ be a division of $W(A)$, and $L=(v(0),v(1),...)$, $M=(w(0),w(1),...)$ be models of A.

$L>M$ iff
there exist natural numbers m and n such that
$v(l)=w(l)$ for all $l<m$,
$v(m) \cap W_i = w(m) \cap W_i$, and
$w(m) \cap W_n$ is properly included in $v(m) \cap W_n$.

The semantics of A is defined as the least model of A where the division is provided by the dependency relation among 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 to *process*(i), where the predicate *process*(i) is supposed to 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 *pseudo program*, it is natural to define its model as an interpretation S such that $S|f$ 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 given $P=\{(f_i,A_i) \mid 1 \leq i \leq m\}$, S is its model iff $S|f_i \models A_i$ for all i. As described below, the order among 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 which 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 $\ll f,A \gg$ 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 $\ll f,A \gg$ so that we can manage to compose a program by embeddings and weak anchors. Let $p(x_1,...,x_n)$ be a status predicate for g' where $\bullet g'$ is a subformula in A. Then $WA(f,A)$ contains

$$f \rightarrow (\langle f,g' \rangle \rightarrow p(x_1,...,x_n))$$
$$\sim f \rightarrow (\bullet p(x_1,...,x_n) \rightarrow p(x_1,...,x_n)).$$

No other element is contained in $WA(f,A)$.

Theorem 2 If $S \models WA(f,A)$ holds, $S|f \models A$ 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 the part of status predicates.

Lemma 2 $S|f_k \models A_k$ 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 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 occurred in $\langle f_k,g \rangle$ have already been defined. Let $S'=(w(0),w(1),....)$, $S'|f_k=(w(i_0),w(i_1),....)$, and $t_1,...,t_n$ be ground terms.

$S',w(j) \not\models p(t_1,...,t_n)$ if $j<i_0$

$$S',w(j) \models p(t_1,...,t_n) \text{ iff } S',w(i_n) \models \langle f_k, g(t_1,...,t_n)\rangle \qquad \text{if } i_n \leqq j < i_{n+1} \text{ for some } n$$

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

Lemma 3 If a program $\bigcup_{1\leqq k\leqq m} (\langle f_k,A_k\rangle \cup WA(f_k,A_k))$ has the least model M,

$M \models \ll f_k,A_k \gg$ for all k.

Proof) Suppose there exits some k such that $M \not\models \ll f_k,A_k \gg$. 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

$$M,w(i) \not\models f_k \rightarrow (\langle f_k,g'\rangle \equiv p(t_1,...,t_n)), \text{ or}$$
$$M,w(i) \not\models \sim f_k \rightarrow (\bullet p(t_1,...,t_n) \equiv p(t_1,...,t_n)).$$

On the other hand, by $M,w(i) \models WA(f_k,A_k)$,

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

We find $p(t_1,...,t_n) \in w(i)$ in either case that f_k is true or false. Suppose M' is made 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 $\bigcup_{1\leqq i\leqq m} (\langle f_i,A_i\rangle \cup WA(f_i,A_i))$ has the least model M in the sense of TP,

then $\{(f_i,A_i) \mid 1 \leqq i \leqq m\}$ also has the least model S with the same order and $S \simeq M$.

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 status predicates to be false. We can get the following proposition easily.

$$N \geq [N] \tag{2.1}$$

$$N \leq N' \text{ implies } [N] \leq [N'] \tag{2.2}$$

$$N \simeq N' \text{ implies } [N] = [N'] \tag{2.3}$$

$$S \in E \text{ implies } [S] \in E \tag{2.4}$$

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

We define S in the theorem 3 as the semantics of the pseudo programs. Since status predicates do not occur in the original program, we can consider they have nothing to do with the semantics. Then the 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 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 to keys about individual interphones. In the following, let a and b be parameters occurring in the key formulas, which represent 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 interphones such as *on-hook*, *off-hook*, and *dial* do not happen at the same time even if they happen on different interphones.

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

$$\bullet\bullet\bullet off\text{-}hook(x) \wedge \bullet\bullet dial(x,n_1) \wedge \bullet dial(x,n_2) \wedge dial(x,n_3)$$
$$\rightarrow call(x,100n_1+10n_2+n_3)$$

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

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

$$\bullet on\text{-}hook(y) \wedge call(x,y) \rightarrow ring(y) \wedge calling\text{-}tone(x)$$
$$\bullet off\text{-}hook(y) \wedge call(x,y) \rightarrow busy\text{-}tone(x)$$
$$\bullet call(x,y) \wedge call(x,y) \rightarrow$$
$$(\bullet busy\text{-}tone(x) \rightarrow busy\text{-}tone(x)) \wedge$$
$$(\bullet ring(y) \rightarrow ring(y)) \wedge$$
$$(\bullet calling\text{-}tone(x) \rightarrow calling\text{-}tone(x))$$

Regarding to $(call(a,b) \wedge \text{ring}(b)) \vee on\text{-}hook(a) \vee off\text{-}hook(b)$:

$$\bullet call(x,y) \wedge on\text{-}hook(x) \rightarrow quiet(y)$$
$$\bullet call(x,y) \wedge off\text{-}hook(y) \rightarrow connect(x) \wedge connect(y)$$

It is easily convinced 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 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 act the role of an array which keep internal states.

Now the program contains many formulas instead of many processes, being still difficult to run if the number of interphones is very large. However, as is 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,

$$WA(call(a,b) \vee on\text{-}hook(b) \vee off\text{-}hook(b),$$
$$\{\bullet on\text{-}hook(y) \wedge call(x,y) \rightarrow ring(y) \wedge calling\text{-}tone(x)\})$$

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

$$call(a,b) \vee on\text{-}hook(b) \vee off\text{-}hook(b) \rightarrow on\text{-}hook(y) \rightarrow p\text{-}a\text{-}b(y)$$
$$\sim(call(a,b) \vee on\text{-}hook(b) \vee off\text{-}hook(b)) \rightarrow \bullet p\text{-}a\text{-}b(y) \rightarrow p\text{-}a\text{-}b(y)$$

Replace each occurrence of $p\text{-}a\text{-}b(x)$ by $p(a,b,x)$.

$$call(a,b) \vee on\text{-}hook(b) \vee off\text{-}hook(b) \rightarrow on\text{-}hook(y) \rightarrow p(a,b,y)$$
$$\sim(call(a,b) \vee on\text{-}hook(b) \vee off\text{-}hook(b)) \rightarrow \bullet p(a,b,y) \rightarrow p(a,b,y)$$

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

$$call(v,w) \vee on\text{-}hook(w) \vee off\text{-}hook(w) \rightarrow on\text{-}hook(y) \rightarrow p(v,w,y)$$
$$\sim(call(v,w) \vee on\text{-}hook(w) \vee off\text{-}hook(w)) \rightarrow \bullet p(v,w,y) \rightarrow p(v,w,y)$$

In addition, we can improve the program using information from other parts of the program. By the hypothesis that *dial*, *on-hook*, and *off-hook* do not happen at the same

time, and the fact that *call* is true only if *dial* is true, we find that *call*, *on-hook*, and *off-hook* do not happen at the same time. Therefore the formulas above are transformed to

$$on\text{-}hook(w) \rightarrow p(v,w,w)$$
$$\sim(call(v,w) \vee on\text{-}hook(w) \vee off\text{-}hook(w)) \rightarrow \bullet p(v,w,y) \rightarrow p(v,w,y)$$

in which v and y have no meaning. The final version is

$$on\text{-}hook(w) \rightarrow p(w)$$
$$\sim(call(v,w) \vee on\text{-}hook(w) \vee off\text{-}hook(w)) \rightarrow \bullet p(w) \rightarrow p(w).$$

4. GENERALIZATION AND APPLICATION

4.1 Other Modal Logics

In order to extend 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 visibility relations among them are induced in the following way: Let w_i be a world for every i, and F be a set of pairs of worlds associated to a modality. Then new relation G induced by F consists of (w_1,w_n) such that $(w_1,w_2),(w_2,w_3),\ldots,(w_{n-1},w_n) \in F$, and the key formula holds at w_1 and w_n, and does not hold at $w_2,\ldots,w_{n-1}$.

For example, in case of S4, it is almost trivial to see that

"$\Box g$ holds in the extracted model regarding to a key f"

is equivalent to

"$\Box(f \rightarrow g)$ holds in the original model."

On the other hand, in case of the temporal and spatial logic [8] the generalization above does not work well because it is necessary to deal with combinations of temporal and

spatial modalities, which should be another research problem.

4.2 Application for Distributed System

Time-extraction can be helpful for designing and programming of 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 so expensive as multi-processor communications.) In the usual design method, this decision takes place at the earliest stage, therefore it costs very expensive to fix mistakes found in later stages, or to adapt the program to future changes of the hardware definition.

Using extraction, design and programming will be like following: First we write very small pieces of the program in appropriate time streams, which build up a model of programmer's view and have nothing to do with 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 detail information about modularity is available. Furthermore changes of the hardware definition require only re-assignment to the processors and compilation.

5. 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 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 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 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 more than one events never happen at the same time. In real machines, however, one processor can get one interrupt signal at a time. (Nesting of interrupts is another problem.) In 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, though it needs further considerations.

Acknowledgement

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

References

[1] M.Ben-Ari, Z.Manna and A.Pnueli,
The Temporal Logic of Branching time, 8th Ann. ACM Symp. on Principles of Programming Language (1981) pp.164-176

[2] F.Cröger,
A Generalized Nexttime Operator in Temporal Logic,
Journal of Computer and System Sciences 29 (1984) pp.80-98

[3] J.DeTreville,
Phoan: An Intelligent System For Distributed Control Synthesis,
ACM SIGPLAN Notices Vol.19,No.5 (1984) pp.96-103

[4] M.H.Van Emden and R.A.Kowalski,
The Semantics of Predicate Logic as a Programming Language,
JACM Vol.23,No.4 (1976) pp.733-742

[5] E.A.Emerson, Alternative Semantics for Temporal Logics,
Theoretical Computer Science 26 (1983) pp.121-130

[6] T.Hattori, R.Nakajima, T.Sakuragawa, K.Takenaka and N.Nide,
RACCO: A Modal Logic Programming Language for Writing Models of
Real-time Control System
Technical Report in RIMS 558, Kyoto University, Japan

[7] T.Hattori, R.Nakajima, T.Sakuragawa, K.Takenaka and N.Nide,
A Work Out Example of Tube Mill in RACCO
Technical Report in RIMS 561, Kyoto University, Japan

[8] J.Reif and A.P.Sistla,
A Multiprocess Network Logic with Temporal and Spatial Modalities,
Journal of Computer and System Science 30 (1985) pp.41-53

[9] T.Sakuragawa, Temporal Prolog,
Computer Software (To appear)

[10] T.Sakuragawa, A Model of Temporal Prolog,
(In preparation)

[11] P.Zave, An Operational Approach to Requirements Specification for
Embedded Systems, IEEE Trans. Software Engr. SE-8 (1982) pp.250-269

[12] P.Zave, The Operational versus, The Conventional Approach to
Software Development, CACM Vol.27 No.2 (1984) pp.104-118