Semantics of Framed Temporal Logic Programs*

Zhenhua Duan[1], Xiaoxiao Yang[1], and Maciej Koutny[2]

[1] School of Computer Science and Engineering, Xidian University,
Xi'an 710071, P.R. China
{zhhduan, xxyang}@mail.xidian.edu.cn
[2] Department of Computing Science, University of Newcastle,
Newcastle upon Tyne NE1 7RU, UK
maciej.koutny@ncl.ac.uk

Abstract. This paper investigates semantics of framed temporal logic programs. To this end, a projection temporal logic and its executable subset are presented. Based on this language, a framing technique is introduced. The semantics of a non-framed program is well interpreted by the canonical model. However, since introducing a framing operator destroys monotonicity, a canonical model may no longer capture the intended meaning of a program. Hence, a minimal model theory is developed. Within this model, negation by default is used to manipulate frame operator. Further, the temporal semantics of framed programs is captured by means of the minimal models. The existence of a minimal model for a given framed program is also proved. An example is given to illustrate how the semantics of framed programs can be captured.

Keywords: Temporal logic programming, framing, minimal model, monotonicity, semantics.

1 Introduction

Framing [6,3] is concerned with how the value of a variable from one state can be carried to the next. Temporal logic offers no solution in this respect; no value from a previous state is assumed to be carried along. Framing techniques have been employed by conventional imperative languages for many years. However, framing in conventional languages has been taken for granted and there is no conscious effort to consider it explicitly. However, within a temporal logic programming language such as Tempura [8,3], XYZ/E [11] a program is executed over a sequence of states and the values of variables are not inherited automatically. Thus, for improving the efficiency of a program and synchronizing communication for parallel processes, we have to consider the framing techniques carefully in temporal logic programming. To synchronize communication between parallel processes in a concurrent program with the shared variable

* This research is supported by the NSFC Grant No. 60373103 and 60433010, the SRFDP Grant 20030701015, and Grant SYSKF0407 from Lab. Computer Science, ISCAS.

M. Gabbrielli and G. Gupta (Eds.): ICLP 2005, LNCS 3668, pp. 356–370, 2005.

model, a synchronization construct, *await*(c) is required, similarly as in many concurrent programming languages [10]. Defining *await*(c) is difficult without some kind of framing construct since the values of variables are not inherited automatically from one state to another. But one requires some kind of indefinite stability, since it cannot be known at the point of use how long the waiting will last. At the same time one must also allow variables to change, so that an external process can modify the boolean parameter and it can eventually become true.

To capture the temporal semantics of non-framed programs in Tempura, the canonical model has been introduced to interpret programs [3]. Within this model, the semantics of a non-framed program is well captured. However, since introducing a framing operator destroys monotonicity, a canonical model may no longer capture the intended meaning of a program. A program, therefore, can have different meanings under different models. To interpret a framed program faithfully, minimal models will be employed in this paper. Within this model, negation by default is used to manipulate the frame operator. Furthermore, the existence of a minimal model for a satisfiable program is proved by means of fix-point theory.

This paper is organized as follows. In the following section, a Projection Temporal Logic (PTL) is briefly introduced. Based on this logic, an executable temporal logic programming language called Tempura is formalized in Section 3. Section 4 formalizes a framing technique. Section 5 presents the temporal semantics of framed programs by means of minimal models. Finally, in Section 6, an example is given to illustrate how the minimal model can be used to capture the meaning of a framed program. Conclusions are drawn in Section 7.

2 Projection Temporal Logic

Our underlying logic PTL is the first order temporal logic [7,10] with projection [2,3,5]. It is an extension of ITL [8].

2.1 Syntax

Let Π be a countable set of *propositions*, and V be a countable set of typed static and dynamic *variables*. The *terms* e and *formulas* p of the logic are given by the following grammar:

$$e ::= x \mid u \mid \bigcirc e \mid \ominus e \mid \mathit{beg}(e) \mid \mathit{end}(e) \mid f(e_1, \ldots, e_n)$$

$$p ::= \pi \mid e_1 = e_2 \mid P(e_1, \ldots, e_n) \mid \neg p \mid p_1 \wedge p_2 \mid \exists x : p \mid \bigcirc p \mid \ominus p \mid (p_1, \ldots, p_m)\ \mathit{prj}\ p$$

where π is a proposition, x is a dynamic variable and u is a static variable. In $f(e_1, \ldots, e_n)$ and $P(e_1, \ldots, e_n)$, where f is a function and P is a predicate. It is assumed that the types of the terms are compatible with those of the arguments of f and P. A formula (term) is called a state formula (term) if it does not contain any temporal operators (i.e. $\bigcirc$, $\ominus$, *beg*(.), *end*(.) and *prj*); otherwise it is a temporal formula (term).

2.2 Semantics

A *state* s is a pair of assignments (I_v, I_p) which for each variable $v \in V$ defines $s[v] = I_v[v]$, and for each proposition $\pi \in \Pi$ defines $s[\pi] = I_p[\pi]$. $I_v[v]$ is a value of the appropriate type or *nil* (undefined), whereas $I_p[\pi] \in \{true, false\}$. An *interval* $\sigma = \langle s_0, s_1, \ldots \rangle$ is a non-empty (possibly infinite) sequence of states. The length of σ, denoted by $|\sigma|$, is defined as ω if σ is infinite; otherwise it is the number of states in σ minus one. To have a uniform notation for both finite and infinite intervals, we will use *extended integers* as indices. That is, we consider the set N_0 of non-negative integers and ω, $N_\omega = N_0 \cup \{\omega\}$ and extend the comparison operators, $=, <, \leq$, to N_ω by considering $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Moreover, we define $\preceq$ as $\leq - \{(\omega, \omega)\}$. For $0 \leq i, j \leq |\sigma|$ we will use $\sigma_{(i..j)}$ to denote the subinterval $\langle s_i, s_{i+1}, \ldots, s_j \rangle$.[1] It is assumed that each static variable is assigned the same value in all the states in σ. To define the semantics of the projection operator we need an auxiliary operator for intervals.

Let $\sigma = \langle s_0, s_1, \ldots \rangle$ be an interval and $r_1, \ldots, r_h$ be integers ($h \geq 1$) such that $0 \leq r_1 \leq r_2 \leq \ldots \leq r_h \leq |\sigma|$. The *projection* of σ onto $r_1, \ldots, r_h$ is the interval, $\sigma{\downarrow}(r_1, \ldots, r_h) = \langle s_{t_1}, s_{t_2}, \ldots, s_{t_l} \rangle$, where $t_1, \ldots, t_l$ is obtained from $r_1, \ldots, r_h$ by deleting all duplicates. For example, $\langle s_0, s_1, s_2, s_3, s_4 \rangle{\downarrow}(0, 0, 2, 2, 2, 3) = \langle s_0, s_2, s_3 \rangle$.

An *interpretation* for a PTL term or formula is a tuple $\mathcal{I} = (\sigma, i, k, j)$, where $\sigma = \langle s_0, s_1, \ldots \rangle$ is an interval, i and k are non-negative integers, and j is an integer or ω, such that $i \leq k \preceq j \leq |\sigma|$. We use (σ, i, k, j) to mean that a term or formula is interpreted over a subinterval $\sigma_{(i..j)}$ with the current state being s_k. For every term e, the evaluation of e relative to interpretation $\mathcal{I} = (\sigma, i, k, j)$ is defined as $\mathcal{I}[e]$, by induction on the structure of a term, as shown in Fig. 1, where v is a variable and $e_1, \ldots, e_m$ are terms. The satisfaction relation for formulas $\models$ is defined as the least relation satisfying the following.

$$
\begin{aligned}
\mathcal{I}[a] &= s_k[a] \;=\; I_v^k[a] \;=\; I_v^i[a] \quad \text{if } a \text{ is a static variable.} \\
\mathcal{I}[x] &= s_k[x] \;=\; I_v^k[x] \text{ if } x \text{ is a dynamic variable.} \\
\mathcal{I}[f(e_1, \ldots, e_m)] &= \begin{cases} f(\mathcal{I}[e_1], \ldots, \mathcal{I}[e_m]) & \text{if } \mathcal{I}[e_h] \neq nil \text{ for all } h \\ nil & \text{otherwise} \end{cases} \\
\mathcal{I}[\bigcirc e] &= \begin{cases} (\sigma, i, k+1, j)[e] & \text{if } k < j \\ nil & \text{otherwise} \end{cases} \\
\mathcal{I}[\ominus e] &= \begin{cases} (\sigma, i, k-1, j)[e] & \text{if } i < k \\ nil & \text{otherwise} \end{cases} \\
\mathcal{I}[beg(e)] &= (\sigma, i, i, j)[e] \\
\mathcal{I}[end(e)] &= \begin{cases} (\sigma, i, j, j)[e] & \text{if } j \neq \omega \\ nil & \text{otherwise} \end{cases}
\end{aligned}
$$

Fig. 1. Interpretation of PTL terms

[1] When $i > j$, $\sigma_{(i..j)}$ is the empty string, and if $j = \omega$ then $\sigma_{(i..j)} = \langle s_i, s_{i+1}, \ldots \rangle$.

1. $\mathcal{I} \models \pi$ if $s_k[\pi] = I_p^k[\pi] = true$.
2. $\mathcal{I} \models P(e_1, \ldots, e_m)$ if $P(\mathcal{I}[e_1], \ldots, \mathcal{I}[e_m]) = true$ and $\mathcal{I}[e_h] \neq nil$, for all h.
3. $\mathcal{I} \models e = e'$ if $\mathcal{I}[e] = \mathcal{I}[e']$.
4. $\mathcal{I} \models \neg p$ if $\mathcal{I} \not\models p$.
5. $\mathcal{I} \models p \wedge q$ if $\mathcal{I} \models p$ and $\mathcal{I} \models q$.
6. $\mathcal{I} \models \bigcirc p$ if $k < j$ and $(\sigma, i, k+1, j) \models p$.
7. $\mathcal{I} \models \ominus p$ if $i < k$ and $(\sigma, i, k-1, j) \models p$.
8. $\mathcal{I} \models \exists x : p$ if for some interval σ' which has the same length as σ, $(\sigma', i, k, j) \models p$ and the only difference between σ and σ' can be in the values assigned to variable x.
9. $\mathcal{I} \models (p_1, \ldots, p_m)\ prj\ q$ if there exist integers $k = r_0 \leq r_1 \leq \ldots \leq r_m \preceq j$ such that $(\sigma, i, r_0, r_1) \models p_1$, $(\sigma, r_{l-1}, r_{l-1}, r_l) \models p_l$ (for $1 < l \leq m$), and $(\sigma', 0, 0, |\sigma'|) \models q$ for one of the following σ':
   (a) $r_m < j$ and $\sigma' = \sigma{\downarrow}(r_0, \ldots, r_m) \cdot \sigma_{(r_m+1..j)}$
   (b) $r_m = j$ and $\sigma' = \sigma{\downarrow}(r_0, \ldots, r_h)$ for some $0 \leq h \leq m$.

A formula p is said to be:

- *satisfied* by an interval σ, denoted $\sigma \models p$, if $(\sigma, 0, 0, |\sigma|) \models p$.
- *satisfiable* if $\sigma \models p$ for some σ.
- *valid*, denoted $\models p$, if $\sigma \models p$ for all σ.
- *left end closed* (lec-formula) if $(\sigma, k, k, j) \models p \Leftrightarrow (\sigma, i, k, j) \models p$ for any interpretation (σ, i, k, j).
- *equivalent* to another formula q, denoted $p \equiv q$, if $\models \Box(p \leftrightarrow q)$.

Projection. To ensure smooth synchronization between $p_1, \ldots, p_m$ and q, the previous operator is not allowed within q appearing in $(p_1, \ldots, p_m)\ prj\ q$. The projection construct is executable, and to interpret $(p_1, \ldots, p_m)\ prj\ q$ we need *two* sequences of clocks (states) running on different time scales: one is a local state sequence, over which $p_1, \ldots, p_m$ are executed, while the other is a global state sequence over which q is executed in parallel with the sequence of processes $p_1, \ldots, p_m$. The execution proceeds as follows: First, q and p_1 start at the first global state and p_1 is executed over a sequence of local states until its termination. Then (the remaining part of) q and p_2 are executed at the second global state. Subsequently, p_2 is continuously executed over a sequence of local states until its termination, and so on. Although q and p_1 start at the same time, $p_1, \ldots, p_m$ and q may terminate at different time points. E.g., if q terminates before some p_{h+1}, then, subsequently, $p_{h+1}, \ldots, p_m$ are executed sequentially.

2.3 Other Formulas

The derived connectives, $\vee$, $\rightarrow$ and $\leftrightarrow$, as well as the logic formulas, *true* and *false*, are defined as usual. We also use the following derived formulas:

Simple Temporal Formulas

$$Prj\ (p_1,\ldots,p_m) \stackrel{\text{def}}{=} (p_1,\ldots,p_m)\ prj\ empty \qquad empty \stackrel{\text{def}}{=} \neg \bigcirc true$$

$$\odot p \stackrel{\text{def}}{=} \neg\ominus\neg p \qquad more \stackrel{\text{def}}{=} \neg empty$$

$$first \stackrel{\text{def}}{=} \neg\ominus true \qquad skip \stackrel{\text{def}}{=} len(1)$$

$$\bigodot p \stackrel{\text{def}}{=} empty \vee \bigcirc p \qquad \Diamond p \stackrel{\text{def}}{=} Prj\ (true, p)$$

$$len(n) \stackrel{\text{def}}{=} \begin{cases} empty & n = 0 \\ \bigcirc len(n-1) & n > 1 \end{cases} \qquad \Box p \stackrel{\text{def}}{=} \neg\Diamond\neg p$$

The chop operator (;), which is a central operator in ITL [8], can be expressed by the projection operator of PTL, as follows: $p; q \stackrel{\text{def}}{=} Prj\ (p, q)$. The chop star operator ($*$) of [9] can also be defined.

Theorem 1. Let p, q, w be formulas, and e, e_1, e_2 terms, then the following formulas hold:

$FCH1$	$\bigcirc p; q \equiv \bigcirc (p; q)$	$FDU1$	$\neg\bigodot p \equiv \bigcirc\neg p$
$FCH2$	$w \wedge (p; q) \equiv (w \wedge p; q)$	$FDU2$	$\neg\bigcirc p \equiv \bigodot\neg p$
$FD3$	$\bigcirc(p \wedge q) \equiv \bigcirc p \wedge \bigcirc q$	$FE1$	$\Diamond p \equiv p \vee \bigcirc\Diamond p$
$FD4$	$\bigcirc(p \vee q) \equiv \bigcirc p \vee \bigcirc q$	$FE2$	$\Box p \equiv p \wedge \bigodot\Box p$
$FD9$	$(w; p \vee q) \equiv (w; p) \vee (w; q)$	NFE	$\neg first \wedge more \supset (\bigcirc\ominus p \leftrightarrow \ominus\bigcirc p)$
$FD10$	$(p \vee q; w) \equiv (p; w) \vee (q; w)$	$FQT1$	$\bigcirc(\exists x : p) \equiv \exists x : \bigcirc p$
$FW2$	$\bigodot p \equiv \neg\bigcirc\neg p$	$EQ3$	$\neg first \wedge more \supset (\bigcirc\ominus e = \ominus\bigcirc e)$
$FW1$	$\bigcirc p \equiv \bigodot p \wedge more$	$EQ1$	$more \supset (\bigcirc e_1 = \bigcirc e_2 \leftrightarrow \bigcirc(e_1 = e_2))$
$FUN3$	$\bigcirc e_1 + \bigcirc e_2 = \bigcirc(e_1 + e_2)$	$FST1$	$p^* \equiv empty \vee (p; p^*) \vee p \wedge \Box more$

These logic laws are useful in the reduction of programs and the proofs of them can be found in [3].

3 Temporal Logic Programming Language

The programming language we use is an executable subset of PTL. It is an extension of Tempura [8,6]. We augment Tempura with frame, new projection, and await operators [2,3,4,5]. In addition, variables within a program can also refer to their previous values. In the following, we first introduce the basic constructs of Tempura. Later, we formalize the frame and await constructs.

3.1 Syntax

The basic statements of Tempura are as follows.

- **Assignment**: $x = e$
- **Conjunction**: $p \wedge q$
- **Conditional statement**: $if\ b\ then\ p\ else\ q \stackrel{\text{def}}{=} (b \rightarrow p) \wedge (\neg b \rightarrow q)$
- **Local variable**: $\exists x : p$
- **Next statement**: $\bigcirc p$

- **Always statement**: $\Box p$
- **Sequential statement**: $p; q$
- **While statement**: $while\ b\ do\ p \stackrel{\text{def}}{=} (p \wedge b)^* \wedge \Box(empty \rightarrow \neg b)$
- **Projection statement**: $(p_1, ..., p_m)\ prj\ q$
- **Parallel statement**: $p||q \stackrel{\text{def}}{=} p \wedge (q; true) \vee q \wedge (p; true)$
- **Termination**: $empty$

where b is a state boolean expression consisting of propositions, variables, and boolean connectives.

The following formulas are derived from PTL and can be used in programs.

1. **Assignment Operators**
   Let x be a variable, u a static variable, and e an expression (term).
   1) Next assignment: $x\ o{=}\ e \stackrel{\text{def}}{=} \bigcirc x = e$
   2) Unit assignment: $x\ :=\ e \stackrel{\text{def}}{=} skip \wedge x\ o{=}\ e$
   The next assignment specifies the value of x to be e at the next state, while the unit assignment assigns value e to x at the next state, the same function as the next assignment, but, in the meantime, it specifies the length of the interval over which the assignment takes place to be 1.

2. **Termination and the Final State**
   1) $fin(p) \stackrel{\text{def}}{=} \Box(empty \rightarrow p)$
   2) $keep(p) \stackrel{\text{def}}{=} \Box(\neg empty \rightarrow p)$
   3) $halt(p) \stackrel{\text{def}}{=} \Box(empty \leftrightarrow p)$
   $fin(p)$ holds over an interval as long as p holds at the final state, whereas $keep(p)$ holds over an interval if p holds at every state, ignoring the final one. $halt(p)$ holds over an interval if and only if p holds at the final state.

3.2 Semantics of Programs

An expression e can be treated as a term and a program P can be viewed as a formula in PTL. Therefore, the evaluation of e and the interpretation of P can be done as in PTL. However, since the programming language is a subset of the underlying logic, a program may have its own characteristics and may be interpreted in a simple and manageable way.

In order to interpret temporal logic programs, we assume that a program P contains a finite set S of variables and a finite set Φ of propositions. We interpret propositions over B and variables over $D' = D \cup \{nil\}$, where nil is undefined and D denotes all data needed by us including integers, lists, sets etc. For a program P, there are three ways to interpret propositions contained in P, namely canonical, complete, and partial interpretations as defined for the semantics of logic programming language [1]. Here, we use the canonical interpretation only on propositions. That is, in a model $\sigma =< (I_v^0, I_p^0), ... >$, I_v^k is used as in the logic but I_p^k is changed to the canonical interpretation.

A canonical interpretation on propositions is a subset $I_p \subseteq \Phi$. Implicitly, propositions not in I_p are false. Note that I_p^k in the interpretation of the logic

framework is an assignment of a truth value in B to each proposition $\pi \in \Pi$ at state s_k; whereas in a canonical interpretation, I_p^k is a set of propositions, each of them has truth value *true* in B at s_k. Clearly, the two definitions are equivalent except that they refer to different sets of variables and propositions. Using canonical interpretation is necessary for easy manipulation of minimal models. Let $\sigma =< (I_v^0, I_p^0), ... >$ be a model. We denote the sequence of interpretation on propositions of σ by $\sigma_p =< I_p^0, ... >$. σ_p is said to be canonical if each $I_p^i (i \geq 0)$ is a canonical interpretation on propositions.

If there exists a model σ with σ_p being a canonical interpretation sequence on propositions and $\sigma \models P$ as in the logic, then program P is said to be satisfiable under the canonical interpretation on propositions, denoted by $\sigma \models_c P$; and σ_p is said to be a canonical interpretation sequence (on propositions) of program P. If for all σ with σ_p being a canonical interpretation sequence, $\sigma \models P$, then program P is said to be valid under the canonical interpretation on propositions, denoted by $\models_c P$.

Note that the definition of the canonical interpretation of program P is independent of its syntax in the sense that the definition does not refer to the structure of the program. So the definition can be extended so that it can be applied to non-deterministic programs and temporal formulas.

Example 1. For the propositional formula, P_1: $\neg A \leftrightarrow \bigcirc B$, which can be treated as a non-deterministic program, we have $\Phi = \{A, B\}$, and P_1 has the following canonical interpretation sequences of length 2, $< \phi, \{B\} >$, $< \phi, \{A, B\} >$, $< \{B\}, \{B\} >$, $< \{B\}, \{A, B\} >$, $< \{A\}, \phi >$, $< \{A\}, \{A\} >$, $< \{A, B\}, \phi >$, and $< \{A, B\}, \{A\} >$.

P_1 is satisfiable but not valid under the canonical interpretation on propositions because a canonical interpretation sequence, $< \phi, \phi >$, does not satisfy it.

Note that a program P can be satisfied by several different canonical models on propositions so program P has, possibly, different meanings under different models. Therefore, it is important to choose a model which satisfies the intended meaning of a program P, and this is the topic of Section 5.

Since the canonical model is basically equivalent to the basic model except that the latter acts on the fixed set V of variables and the fixed set Π of propositions, whereas the former acts on the set of variables and the set of propositions within a concrete program. $\exists x : p(x)$ can be renamed as a formula $p(y)$ (or $p[y/x]$) with a free variable y by renaming x as y.

Lemma 2. Let $p(y)$ be a renamed formula of $\exists x : p(x)$. Then, $\exists x : p(x)$ is satisfiable if and only if $p(y)$ is satisfiable. Furthermore, any model of $p(y)$ is a model of $\exists x : p(x)$.

4 Framing

In this section, we first define some new assignments which are required by framing, then we define frame operators; and finally, we present a minimal model-based approach for framing.

Suppose $S = \{x_1, ..., x_n\}(S \subset V)$ is a set of state variables within a program P. Note that variables bound by quantifiers can always be given distinct names by renaming them as necessary.

Definition 1. *(new assignments)*

(1) $x_i \Leftarrow e \stackrel{\text{def}}{=} x_i = e \wedge p_i \quad (0 \leq i \leq n, e \neq nil)$
(2) $x_i \ o{=}^+ \ e \stackrel{\text{def}}{=} \bigcirc x_i = e \wedge \bigcirc p_i$
(3) $x_i :=^+ e \stackrel{\text{def}}{=} x_i \ o{=}^+ e \wedge skip$

where p_i is an atomic proposition associated with state variable x_i $(0 \leq i \leq n)$ and cannot be used for other purposes.

The meanings of these assignment operators are similar to those presented in Section 3, but they render some propositions *true* besides assigning some values to variables in the same unit of time. It is now time to define the assignment flag

$$af(x_i) \stackrel{\text{def}}{=} p_i$$

where proposition p_i associated with variable x_i is the same as in Definition 1, and cannot be used for other purposes. As expected, whenever $x_i \Leftarrow b$ is encountered, p_i is set to be *true*, hence $af(x_i)$ is *true* whereas if no assignment to x_i takes place, p_i is unspecified. In this case, we will use a minimal model to force it to be $false$.

Armed with the assignment flag, we can define state frame and interval frame operators. Intuitively, when a variable is framed at a state, its value remains unchanged if no assignment is encountered at that state. A variable is framed over an interval if it is framed at every state over the interval.

Definition 2. *(looking back framing)*

(1) $lbf(x_k) \stackrel{\text{def}}{=} \neg af(x_k) \rightarrow \exists b : (\ominus x_k = b \wedge x_k = b)$
(2) $frame(x_k) \stackrel{\text{def}}{=} \Box(more \rightarrow \bigcirc lbf(x_k))$
(3) $frame(x_1, ..., x_n) \stackrel{\text{def}}{=} frame(x_1) \wedge ... \wedge frame(x_n)$

We interpret programs using minimal models. Let $\sigma =< s_0, ... >$ be an interval, and $s_i = (I_v^i, I_p^i)$; I_v^i is defined as in Section 2.2 and I_p^i is the canonical interpretation defined as in Section 3.2 but the sequence of interpretations on propositions of σ, $\sigma_p =< I_p^0, ..., >$, is required to be a minimal canonical sequence, as defined in the next section. Armed with framing operator, the synchronized communication construct $await(c)$ can be defined as follows:

Definition 3. $await(c) \stackrel{\text{def}}{=} frame(V_c) \wedge halt(c)$ whereV_c represents all dynamic variables contained in c.

5 Minimal Model

5.1 The Minimal Satisfaction Relation

In this section, we discuss semantics of framed programs. As before, let V denote the set of all variables. A dynamic variable $x \in V$ is said to be framed in a

program p if $frame(x)$ or $lbf(x)$ is contained in p. A program p is said to be framed if p contains at least one framed variable. In general, a framed program is non-deterministic under the canonical model. Consequently, a framed program can inductively be defined, as follows

- For any variable $x \in V$ and any well-formed expression e, $x = e$, $x \Leftarrow e$, and $empty$ are framed programs.
- $lbf(x)$, and $frame(x)$ are framed programs.
- If $p, q, p_1, ..., p_m$ are framed programs, then so are the followings:

  $\bigcirc p$, $\Box p$, $p \wedge q$, $p; q$, $if\ b\ then\ p\ else\ q$, $while\ b\ do\ p$, $p \| q$, $(p_1, ..., p_m)\ prj\ q$, and $\exists x : p$.

Fact 1.

$$\begin{array}{rl} EQFR & x_i = e_i \equiv p_{x_i} \wedge x_i = e_i \vee \neg p_{x_i} \wedge x_i = e_i \\ LBF & lbf(x_i) \equiv p_{x_i} \vee \neg p_{x_i} \wedge x_i = \ominus x_i \end{array}$$

Proof:

EQFR is obviously true. We only prove LBF.

$$\begin{array}{rll} lbf(x_i) & \equiv \neg af x_i \rightarrow \exists b : \ominus x_i = b \wedge x_i = b & \\ & \equiv \neg p_{x_i} \rightarrow \exists b : \ominus x_i = b \wedge x_i = b & \\ & \equiv \neg p_{x_i} \rightarrow \ominus x_i = a \wedge x_i = a & \text{Lemma 2} \\ & \equiv \neg p_{x_i} \rightarrow x_i = \ominus x_i\ (\neq nil) & \\ & \equiv p_{x_i} \vee \neg p_{x_i} \wedge x_i = \ominus x_i & \end{array}$$

By EQFR and LBF, when we reduce a framed program p, whenever $x_i = e_i$ occurs in p, it is replaced by $p_{x_i} \wedge x_i = e_i \vee \neg p_{x_i} \wedge x_i = e_i$; whereas whenever $lbf(x_i)$ occurs in p, it is replaced by $p_{x_i} \vee \neg p_{x_i} \wedge x_i = \ominus x_i$. Then we can reduce p under the canonical model as usual.

A framed program p can be a non-deterministic program. There may be several models satisfying the program under the canonical models.

Example 2.

$$\begin{array}{l} \quad frame(x) \wedge x = 1 \wedge len(1) \\ \equiv \Box(more \rightarrow \bigcirc lbf(x)) \wedge x = 1 \wedge \bigcirc(empty) \\ \equiv (more \rightarrow \bigcirc lbf(x)) \wedge \odot \Box(more \rightarrow \bigcirc lbf(x)) \wedge x = 1 \wedge more \wedge \bigcirc(empty) \\ \equiv \bigcirc lbf(x) \wedge \bigcirc \Box(more \rightarrow \bigcirc lbf(x)) \wedge x = 1 \wedge \bigcirc(empty) \\ \equiv (p_x \wedge x = 1 \vee \neg p_x \wedge x = 1) \wedge \bigcirc(lbf(x) \wedge empty) \end{array}$$

Thus,

$$p_c \equiv p_x \wedge x = 1 \vee \neg p_x \wedge x = 1$$

$$\begin{array}{rl} p_f & \equiv lbf(x) \wedge empty \\ & \equiv (p_x \vee \neg p_x \wedge x = 1) \wedge empty \\ & \equiv p_x \wedge empty \vee \neg p_x \wedge x = 1 \wedge empty \end{array}$$

Hence, four models given below can satisfy the program.

$$\sigma_1 =< (\{p_x\}, \{x:1\}), (\{p_x\}, \phi) >, \sigma_2 =< (\{p_x\}, \{x:1\}), (\phi, \{x:1\}) >$$

$$\sigma_3 =< (\phi, \{x:1\}), (\{p_x\}, \phi) >, \sigma_4 =< (\phi, \{x:1\}), (\phi, \{x:1\}) >$$

As seen, a framed program can have a number of canonical models. Thus, a problem we have to face is how to choose a model to satisfy the intended meaning of a program. We interpret framed programs using minimal models.

Definition 4. Let p be a framed program, and $\Sigma_p = \{\sigma | \sigma \models_c p\}$. Let $\sigma_p =< I_p^0, I_p^1, ... >$, $\sigma_1, \sigma_2 \in \Sigma_p$. We define

- $\sigma_{1p} \sqsubseteq \sigma_{2p}$ iff $I_{1p}^i \subseteq I_{2p}^i$ and $|\sigma_1| = |\sigma_2|$ for all $i, 0 \leq i \leq |\sigma_1|$
- $\sigma_1 \sqsubseteq \sigma_2$ iff $\sigma_{1p} \sqsubseteq \sigma_{2p}$
- $\sigma_1 \sqsubset \sigma_2$ iff $\sigma_1 \sqsubseteq \sigma_2$ and $\sigma_2 \not\sqsubseteq \sigma_1$

Example 3.

$$< (\phi, \{x:1\}) >=< (\phi, \{x:1\}) >, < (\{p_x\}, \phi) > \sqsupset < (\phi, \{x:1\}) >$$

Definition 5. (the minimal satisfaction relation)
Let p be a program, and (σ, i, k, j) be an interpretation. Then the minimal satisfaction relation $\models_m$ is defined as

$(\sigma, i, k, j) \models_m p$ iff $(\sigma, i, k, j) \models_c p$ and there is no σ' such that $\sigma' \sqsubset \sigma$ and $(\sigma', i, k, j) \models_c p$.

A program p is satisfied by a model σ under relation $\models_m$, denoted by $\sigma \models_m p$, if $(\sigma, 0, 0, |\sigma|) \models_m p$. A model σ is a minimal model of program p if $\sigma \models_m p$.

The relations $\equiv_m$ and $\approx_m$ can be defined similarly to the relations $\equiv$ and $\approx$. $p \equiv_m q$ iff for all σ, all k, $0 \leq k \preceq |\sigma|$, $(\sigma, 0, k, |\sigma|) \models_m p \Leftrightarrow (\sigma, 0, k, |\sigma|) \models_m q$. $p \approx_m q$ iff for all σ, $\sigma \models_m p \Leftrightarrow \sigma \models_m q$. The relations $\equiv_m$ and $\approx_m$ are also equivalence relations over the set of programs. That is, they are reflexive, symmetric and transitive.

Note that the definition of the minimal model of a program p is also independent of its syntax in the sense that the definition does not refer to the structure of the program, and can be applied to temporal formulas.

Example 4. The program p in Example 2 has only one minimal model $\sigma_4 = (< \phi, \{x:1\}), (\phi, \{x:1\}) >$. The formula P_1 in Example 1 has only two minimal models, namely, $< \phi, \{B\} >$ and $< \{A\}, \phi >$.

The intended meaning of a program p is captured by its minimal model. For instance, if p is $x_1 \Leftarrow 1 \wedge frame(x_1) \wedge len(1)$ then under the minimal model, $x_1 = 1$ defined at both state s_0 and s_1, this is the intended meaning of p. However, within only the canonical model, p_{x_1} is unspecified at state s_1, so it could be *true* at s_1. This causes x_1 to be unspecified at state s_1. Therefore, x_1 could be any value from its domain.

5.2 Normal Form

Definition 6. A framed program q is in normal form if

$$q \stackrel{\text{def}}{=} \bigvee_{i=1}^{k} q_{ei} \wedge empty \vee \bigvee_{j=1}^{h} q_{cj} \wedge \bigcirc q_{fj} \tag{5.1}$$

where $k, h \geq 0$ $(k + h \geq 1)$ and

- for all $1 \leq j \leq h$, $\bigcirc q_{fj}$ are lec-formulas and q_{fj} are programs.
- q_{cj} $(j \leq h)$ and q_{ei} $(i \leq k)$ are *true* or all state formulas of the form:

$$(x_1 = e_1) \wedge ... \wedge (x_l = e_l) \wedge \dot{p}_{x_1} \wedge ... \wedge \dot{p}_{x_m}$$

  where $e_i \in D$ $(1 \leq i \leq l)$ and $\dot{p}_x$ denotes p_x or $\neg p_x$ and $l \geq 0$ and $m \geq 0$ and $k + h \geq 1$. Notice that, $q_{cj_1} \not\equiv q_{cj_2}$ if $j_1 \neq j_2$, otherwise they can be merged into one by taking the common factor.

In some circumstances, we simply write $q_e \wedge empty$ instead of $\bigvee_{i=1}^{k} q_{ei} \wedge empty$. Also, we call conjuncts, $q_{ei} \wedge empty$, $q_{cj} \wedge \bigcirc q_{fj}$, basic products; the former is called terminal product whereas the latter is called future products. Further, we call q_{ei}, q_{cj} present components, $\bigcirc q_{fj}$ future components of basic products.

Theorem 3. If p is a framed program, then there is a program q as defined in (5.1) such that

$$p \equiv q$$

Theorem 4. Let $q \equiv \bigvee_{i=1}^{k} q_{ei} \wedge empty \vee \bigvee_{j=1}^{h} q_{cj} \wedge \bigcirc q_{fj}$ be the normal form of a framed program q. If p_x and $x = e'$, where $e' \neq e$ $(e', e \in D)$, are not contained in q_{ei} $(1 \leq i \leq k)$ and q_{cj} $(1 \leq j \leq h)$, then

$$(p_x \vee \neg p_x \wedge x = e) \wedge q \equiv_m \neg p_x \wedge x = e \wedge q$$

The proofs of the above two theorems can be found in [3]. Armed with the normal form, a program q can be decomposed to a so called Normal Form Graph(NFG) as follows:

Initially, the root (denoted by a small double circle) of the Graph is labelled by program q, each basic product in the normal form of q becomes a son of q. With the terminal product, the edge labelled by present component q_e and a terminal vertex (a small black dot) labelled by ε without appearing of empty; and with the future product, the edge labelled by q_{cj} and the next vertex (a small circle) labelled by next component q_{fj}. Then, q_{fj} can further be reduced to a sub-graph of q and so on. If two vertices are identical, we merge them into one. It is clear that if q has only finite models, its NFG is also finite. A normal form graph is shown in Fig.2(a). Note that a sup-script of a present component denotes the reduction level on a path in NFG.

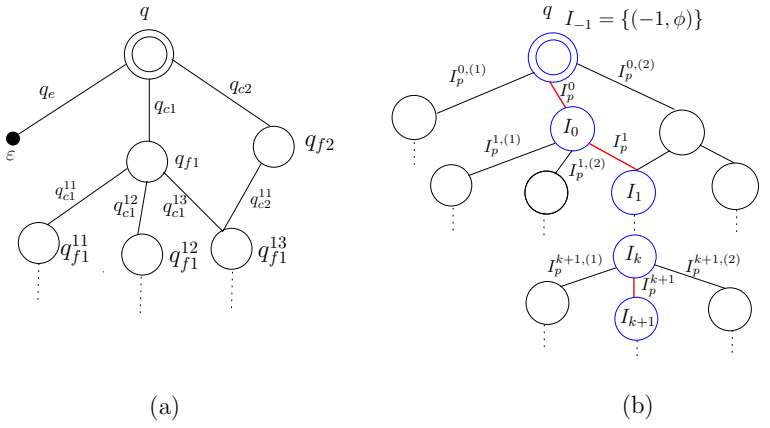

Fig. 2. NFG

5.3 Existence Theorems of Minimal Models

In this section, we investigate the existence of minimal models of a satisfiable framed program. Two theorems are proved. The first is the one with sufficiency conditions whereas the second is the one with necessity and sufficiency conditions.

Theorem 5. Let p be a satisfiable framed program (which may be non-terminating, and/or non-deterministic). If, (1) p has at least one finite model or (2) p has finitely many models, then p has at least one minimal model on propositions.

The proof of the theorem can be found in [3] and is omitted here.

Theorem 6. Let p be a satisfiable framed program (which can be non-terminating, and/or non-deterministic), then p has at least one minimal model on propositions.

Proof:
In order to distinguish operations between sequences and sets, we denote a finite canonical interpretation sequence $(I_p^0, ..., I_p^k)$ by $\overline{I}_k$, and its corresponding coded set $\{(0, I_p^0), ..., (k, I_p^k)\}$ by I_k. Thus, for an arbitrary canonical interpretation sequence $\overline{I} = (I_p^0, ..., I_p^h)$ $(h \in N_0)$, its corresponding coded set is $I = \{(0, I_p^0), ..., (h, I_p^h)\}$.

For convenience, we need to add extra information to the NFG of a program p. First, the label of each edge, i.e, present component in the normal form of p, e.g. p_{ej} or p_{ci} is changed to corresponding canonical interpretation on propositions I_p^h for some $h \geq 0$ ($(h-1)^{th}$ edge from root on a path) ignoring program variables. For instance, if $p_{ci} \equiv p_1 \wedge x_1 = 1 \wedge p_2 \wedge p_3 \wedge x_3 = 2$, then $I_p^h = \{p_1, p_2, p_3\}$. Second, a node is given an extra label I_k. The initial node is labelled by $I_{-1} = \{(-1, \phi)\}$. With a node I_k ($(k-2)^{th}$ node from root), to find out the next edge with minimal canonical interpretation, we define a function $e - min$ as follows.

$e-min(I_k) = min\{I_p^{k+1,(1)}, ..., I_p^{k+1,(h)}\} = I_p^{k+1,(i)}$ (I_p^{k+1} for short), if $I_p^{k+1,(i)} \subset I_p^{k+1,(j)}$ or $I_p^{k+1,(i)}$ is not comparable with $I_p^{k+1,(j)}$, for $\forall j, i \neq j, 0 \leq i, j \leq h$

where $I_p^{k+1,(1)}, ..., I_p^{k+1,(h)}$ are all canonical interpretations associated with edges departing from node I_k.

By Theorem 3, a framed program p can be reduced to its normal form. Since p is satisfiable, so p has at least one canonical model. Thus, we can construct its NFG as shown in Fig.2(b). Based on NFG, we can construct a calculus T on a canonical interpretation coded set I, $T(I)$ is defined as:

$$T(I) = \{(n, I_p^n) | \exists I_{n-1} \subseteq I, I_p^n = e - min(I_{n-1}), n \geq 0\}$$

$e - min(I_{n-1})$ is a function returning the minimal interpretation among all canonical interpretations associated with edges departing from node I_{n-1}.

Initially I_{-1}=$\{(-1, \emptyset)\}$, then we repeatedly apply calculus T to sets $I_{-1}, I_0, ...$ Thus, we got,
$I_0 = T(I_{-1}) = \{(0, I_p^0)\}$, $I_1 = T(I_0) = \{(0, I_p^0), (1, I_p^1)\}$, $I_2 = T(I_1) = T^2(I_0) = \{(0, I_p^0), (1, I_p^1), (2, I_p^2)\}$,..., $I_n = T(I_{n-1}) = T^n(I_0) = \{(0, I_p^0), (1, I_p^1), ..., (n, I_p^n)\}$. Thus, $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots \subseteq I_n \subseteq \ldots$. It is readily to see that T is monotonic.
Let $I = \bigcup_{n=0}^{\infty} I_n$, $\overline{I}_n$ stands for the prefix of minimal interpretation sequence $\overline{I}$.
We now prove the following conclusions.
1. $\overline{I}$ is a canonical interpretation sequence of p.
We first prove $T(I) = I$.
(1) $T(I) \subseteq I$. For $\forall (n+1, I_p^{n+1}) \in T(I)$, by definition, $\exists I_n, I_n \subseteq I, I_p^{n+1} = e - min(I_n)$, so $(n+1, I_p^{n+1}) \in T(I_n)$, i.e. $(n+1, I_p^{n+1}) \in I_{n+1}$. Since $I_{n+1} \subseteq I$, we have $(n+1, I_p^{n+1}) \in I$. Hence $T(I) \subseteq I$.

(2) $I \subseteq T(I)$. That is to prove $\bigcup_{n=0}^{\infty} I_n \subseteq T(I)$. First, we prove $\forall n, n \in N, I_n \subseteq T(I)$. Suppose $(n, I_p^n) \in I_n$, then $(n, I_p^n) \in T(I_{n-1})$. Since $I_{n-1} \subseteq I$, by definition, $I_p^n = e - min(I_{n-1})$, so $(n, I_p^n) \in T(I)$, hence $I_n \subseteq T(I)$. In addition, for all $i \in N$, if $(i, I_p^i) \in \bigcup_{n=0}^{\infty} I_n$, then $\exists n \in N$ such that $(i, I_p^i) \in I_n$. Since $I_n \subseteq T(I)$, so $(i, I_p^i) \in T(I)$, $\bigcup_{n=0}^{\infty} I_n \subseteq T(I)$, therefore $I \subseteq T(I)$.
By the above, we have $T(I) = I$. Thus, I is a fix-point of T and $\overline{I}$ is a canonical interpretation sequence of program p.

2. Let $M = \{\sigma | \sigma \models_c p\}$ and $\sigma \in M$, $\sigma_p = \overline{I}$. Then σ is a minimal model of p.
Suppose $\exists \sigma' \in M, \sigma' \sqsubseteq \sigma$. We prove $\sigma = \sigma'$ by induction on $I_p^n = I_p^{'n}$.
(1) Since $\sigma' \sqsubseteq \sigma$, so $\sigma_p^{'} \sqsubseteq \sigma_p$, i.e. $I_p^{'i} \subseteq I_p^i$ for all i, $0 \leq i \leq |\sigma|$. Thus $I_p^{'0} \subseteq I_p^0$, by definition of T, $I_p^0 \subseteq I_p^{'0}$, therefore $I_p^0 = I_p^{'0}$.
(2) Suppose for $n \leq k$ $(0 \leq k \leq |\sigma|)$, $I_p^h = I_p^{'h}$ $(0 \leq h \leq k)$. Let $n = k+1$. Since $I_p^{'k+1} \subseteq I_p^{k+1}$, on the other hand, by definition of T, $I_p^{k+1} \subseteq I_p^{'k+1}$, so $I_p^{k+1} = I_p^{'k+1}$. Therefore $\sigma = \sigma'$.
In conclusion, σ is a minimal model of program p.

6 Example

In this section, an example is given to show how to apply the minimal model to interpret a framed program. Let $p \equiv frame(x) \wedge x = 1 \wedge if\ p_x\ then\ y \circ= 1\ else\ y \circ=^+ 2 \wedge len(1)$. The following is a complete reduction process of program p. The sup-scripts of components denote the reduction levels and positions.

$$
\begin{aligned}
p &\equiv frame(x) \wedge x = 1 \wedge if\ p_x\ then\ y \circ= 1\ else\ y \circ=^+ 2 \wedge len(1) \\
&\equiv \Box(more \rightarrow \bigcirc lbf(x)) \wedge x = 1 \wedge (p_x \wedge y \circ= 1 \ \vee\ \neg p_x \wedge y \circ=^+ 2) \wedge \\
&\quad \bigcirc(empty) && \text{definition 2} \\
&\equiv (more \rightarrow \bigcirc lbf(x)) \wedge \odot \Box(more \rightarrow \bigcirc lbf(x)) \wedge x = 1 \wedge \\
&\quad (p_x \wedge y \circ= 1 \ \vee\ \neg p_x \wedge y \circ=^+ 2 \wedge \bigcirc(empty) \wedge more && \text{FE2} \\
&\equiv \bigcirc lbf(x) \wedge \bigcirc \Box(more \rightarrow \bigcirc lbf(x)) \wedge x = 1 \wedge \\
&\quad (p_x \wedge y \circ= 1 \ \vee\ \neg p_x \wedge y \circ=^+ 2) \wedge \bigcirc(empty) && \text{FW1} \\
&\equiv x = 1 \wedge \bigcirc(lbf(x) \wedge empty) \wedge (p_x \wedge y \circ= 1 \ \vee\ \neg p_x \wedge y \circ=^+ 2) \\
&\equiv x = 1 \wedge \bigcirc(lbf(x) \wedge empty) \wedge p_x \wedge y \circ= 1 \vee \\
&\quad x = 1 \wedge \bigcirc(lbf(x) \wedge empty) \wedge \neg p_x \wedge y \circ=^+ 2 \\
&\equiv x = 1 \wedge \bigcirc(lbf(x) \wedge empty) \wedge p_x \wedge \bigcirc y = 1 \vee \\
&\quad x = 1 \wedge \bigcirc(lbf(x) \wedge empty) \wedge \neg p_x \wedge \bigcirc y \Leftarrow 2 && \text{Next assignment,def1(3)} \\
&\equiv x = 1 \wedge p_x \wedge \bigcirc(lbf(x) \wedge y = 1 \wedge empty) \vee x = 1 \wedge \neg p_x \wedge \bigcirc(lbf(x) \wedge y \Leftarrow 2 \wedge empty)
\end{aligned}
$$

The last formula shows that p is in a well-reduced (i.e. normal) form $p_c^{0(1)} \wedge \bigcirc p_f^{0(1)} \vee p_c^{0(2)} \wedge \bigcirc p_f^{0(2)}$ at state s_0. Then $p_f^{0(1)}$ and $p_f^{0(2)}$ are continuously re-reduced as follows:

$$
\begin{aligned}
p_c^{0(1)} &\equiv x = 1 \wedge p_x \\
p_f^{0(1)} &\equiv lbf(x) \wedge y = 1 \wedge empty \\
&\equiv (p_x \vee \neg p_x \wedge x = 1) \wedge (p_y \wedge y = 1 \vee \neg p_y \wedge y = 1) \wedge empty && \text{fact1} \\
&\equiv_m \neg p_x \wedge x = 1 \wedge \neg p_y \wedge y = 1 \wedge empty && \text{theorem4} \\
p_c^{0(2)} &\equiv x = 1 \wedge \neg p_x \\
p_f^{0(2)} &\equiv lbf(x) \wedge y \Leftarrow 2 \wedge empty \\
&\equiv (p_x \vee \neg p_x \wedge x = 1) \wedge y \Leftarrow 2 \wedge empty && \text{fact 1,LBF} \\
&\equiv_m \neg p_x \wedge x = 1 \wedge p_y \wedge y = 2 \wedge empty && \text{theorem4, def1(1)}
\end{aligned}
$$

Finally, $p_f^{0(1)}$ is reduced to the form $p_{e(1)} \equiv p_c^{1(1)} \wedge empty$, $p_f^{0(2)}$ is reduced to the form $p_{e(2)} \equiv p_c^{1(2)} \wedge empty$,which indicate that the reduction process of p are successfully completed. Where

$$
\begin{aligned}
p_c^{1(1)} &\equiv \neg p_x \wedge x = 1 \wedge \neg p_y \wedge y = 1, \quad p_f^{1(1)} \equiv empty \\
p_c^{1(2)} &\equiv \neg p_x \wedge x = 1 \wedge p_y \wedge y = 2, \quad p_f^{1(2)} \equiv empty
\end{aligned}
$$

Hence, six models given below can satisfy the program.

$$\sigma_1 =< (\{p_x\}, \{x:1\}), (\{p_x\}, \{y:1\}) >, \sigma_2 =< (\{p_x\}, \{x:1\}), (\{p_x, p_y\}, \{y:1\}) >$$

$$\sigma_3 =< (\{p_x\}, \{x:1\}), (\{p_y\}, \{x:1, y:1\}) >, \sigma_4 =< (\phi, \{x:1\}), (\{p_x, p_y\}, \{y:2\}) >$$

$$\sigma_5 =< (\{p_x\}, \{x:1\}), (\phi, \{x:1, y:1\}) >, \sigma_6 =< (\phi, \{x:1\}), (\{p_y\}, \{x:1, y:2\}) >$$

However, only two minimal models, σ_5 and σ_6, can capture the meaning of program p. This example also shows us that a framed program might have more than one minimal models.

7 Conclusion

This paper presented a framing technique based on an explicit frame operator. A framed program is interpreted by a minimal model. An interpreter was also developed using SICSTUS Prolog for the Framed Tempura. The interpreter employed the framing technique we presented in this paper. It is a workable and useful technique. Because of the space limitation, we cannot introduce the interpreter and the reduction technique in this paper. It will be discussed elsewhere.

References

1. N.Bidoit: *Negation in Rule-based Data Base Languages: A Survey.* Theoretical Computer Science 78 (1991) 3-83, North-Holland.
2. Z.Duan, M.Koutny M., and C.Holt: *Projection in temporal logic programming.* In F. Pfenning, editor, Proceeding of Logic Programming and Automatic Reasoning, Lecture Notes in Artificial Intelligence, a subseries of LNAI, Vol. 822, pp333-344, Springer Verlag, July, 1994.
3. Z.Duan: *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming.* Ph.D Thesis (Technical Report No.556), University of Newcastle upon Tyne, May 1996.
4. Z.Duan, M.Holcombe, and A.Bell: *A Logic for Biological Systems.* BioSystems 55 (2000) 93-105, Elsevier, 2000.
5. Z.Duan, M.Koutny: *A Framed Temporal Logic Programming Language.* Journal of Computer Science and Technology, Vol.19, No.3, pp.341-351, May, 2004.
6. R.Hale: *Programming in Temporal Logic.* Ph.D. Thesis, 173, (1989) Trini College Computer Laboratory, Cambridge University, Cambridge, England, October, 1988
7. F.Kröger: *Temporal logic of programs.* Springer-Verlag (1987).
8. B.Moszkowski: *Executing temporal logic programs.* Cambridge University Press Cambridge (1986).
9. B. Moszkowski: *Some very compositional temporal properties.* Programming Concepts, Methods, and Calculi, 307–326. Elsevier Science B.V. (North-Holland), 1994.
10. Z.Manner and A.Pnueli: *The temporal logic of reactive and concurrent systems.* Springer-Verlag (1992).
11. H.xie, J.Gong, C.S.Tang: *A Structured Temporal Logic Language XYZ/SE.* J.of Comp.Sci.& Tech.,1991.1