Fair Transition System Specification: An Integrated Approach

Guoping Jia and *Guoliang Zheng*
Department of Computer Science, Nanjing University
Nanjing, Jiangsu, 210093, P.R. China
Tel: +86-25-6637551 ext. 3671
Fax: +86-25-3300710

Abstract

In this paper, we discuss the two approaches to the type of formalism used to express specifications: logic-based approach and model-based approach. Temporal logic and state machine, representatives of formalisms used in each approach, are compared. As a result of this comparison, we know that although temporal logics have many advantages, especially abstraction and flexibility of the specification process, they fail to directly characterize situations easily modeled by model-based formalisms, such as "local" properties of execution sequences. To copy with these drawbacks, we present a new kind of formalism: fair transition system specification (FTSS). This approach combines the best features of temporal logic and state machine methods and it is easy to understand and use. A nontrivial example is used to illustrate our approach and it shows that our FTSS approach is promising.

1. Introduction

There have been many approaches for specifying concurrent systems, such as temporal logic, CCS, state machine and Statecharts etc.[10,12,7,3]. According to the type of formalism used to express specifications, two main approaches to this problem can be distinguished: one is *logic-based* method. This approach specifies a program by stating what properties the program should have. In this case, the language of formulas F of some logic is used to specify the behavior of a program and the language P is used for the description of programs. A relation *sat*, subset of $P \times F$, is defined. An assertion p *sat* f means that the program p satisfies the property described by f. This approach corresponds to the *two-language* framework in [14] and the *axiomatic* approach in [5]. The representative of logic-based method is the temporal logic approach. An alternative approach is *model-based* method. This approach, typically, provides an abstract model of the system, that can fully tell how the system should behave. In this case, specifications and program can be considered, at some abstraction, as transition systems. These transition systems are defined from the operational semantics of the specification and programming languages and they provide a common framework for their comparison with programs. A crucial point in the application of this approach is the choice of the relation used to compare programs to specifications. This approach is qualified as the *single-language* framework in [14] and the *constructive* approach in [5]. The representative of model-based method is state machine.

In this paper, we discuss the two main approaches to the specification problem. Especially, temporal logic and state machine, representatives of formalisms used in each approach, are compared. As a result of comparison, we know that although temporal logics have many advantages, such as abstraction and flexibility of the specification process, they fail to directly characterize situations easily modeled by model-based formalisms, such as "local" properties of execution sequences. To copy with these drawbacks, we present a new kind of formalism: fair transition system specification (FTSS). This approach combines the best features of temporal logic and state machine methods and it is easy to understand and use. We further discuss each component of FTSS and we get conclusions that our approach is machine closed and our specification process is consistent and complete. An example of a lossy-transmission protocol is used to illustrate our approach and it shows that our FTSS approach is promising.

2. Logic-based Specifications vs. Model-based Specifications

In this section, we compare the two approaches, especially temporal logic and state machine methods. There are many criteria by which specification styles can be evaluated and compared. Here, we discuss shortly only few of them. The comparison is intended as a summary of the differences between the two and as a basis for the following further discussion of these differences.

· **Incremental Modification**

By incremental modification we mean how easy it is to change the specification, once we realize that one of the requirements is missing or needs to be modified.

Since a logic-based approach is typically to list a set of properties the system should satisfy, it is usually easy to locate the relevant conjunct in the case that an existing requirement is to be modified or to add a new conjunct in the case of a new requirement.

In comparison, such modification is often difficult in a model-based case which usually integrates all the requirements into an abstract machine, and is structured according to processes or modules rather than according to properties. Often, the required change can't be localized and permeates throughout the complete specification. Consider a system specified by a state machine, for which a new additional requirement is identified. Most often the whole specification has to be reconstructed, because of the interaction between the previous requirements and the new one.

· **Consistency**

By consistency we mean that the specification defines a nonempty set of potential implementations, i.e., that there exists at least one system that satisfies the specifications.

In the logic-based style, the danger of inconsistency is nontrivial, since requirements are formulated almost independently of one another. It is easy to inadvertently include two requirements that contradict one another, or a larger set of requirements that can't be jointly satisfied.

In the model-based style, there is a nonproblem. By definition, every abstract model has at least one execution and, therefore, a nonempty semantics. For example, every state machine has a meaning as a process. So, one need never worry about consistency of specification in this style.

· **Completeness**

By completeness we mean the realization that all important properties of the systems have been specified and none is missing. This is a notion that is difficult for the logic-based approach that progresses by adding requirements one at a time and needs a good criterion of when to stop.

In comparison, completeness is rarely an issue in the model-based approach. Consider a state machine system, the only type of incompleteness that can arise in this approach is that a complete state machine has been forgotten or that some transitions have been omitted, while expanding the set of possible transitions from a given state.

· **Validation**

Validation is the process of ensuring that the specification is consistent with the designer's intentions. Since we don't have a formal representation of the designer's innermost thoughts, validation can't be a formal process.

In the logic-based case, we often have some representation of the designer's intents through the informal requirement document. We can perform the validation process in a modular fashion, comparing the formal translation of each requirement to its original description. Even if we don't have a written requirement document, we can validate each requirement separately, testing whether the designer or customer agrees the full meaning of the formal requirement.

The situation is much more difficult in the case of model-based approach. We are faced with a relatively large abstract model and have to determine whether all the possible behaviors it can generate are consistent with our intuition of how the system should behave. In some sense, this task is very similar to the problem of program verification.

From above comparison, we know that formulas in logic-based specifications represent classes of behaviors. This type of formalism has a conjunctive character. As a consequence, logic-based specifications are easily modified and the specification process is flexible. Another obvious advantage of this approach is abstractness, i.e., independence with respect to implementation choices. On the other hand, the model-based approach has some features which make it very attractive. It uses two very primitive concepts: the concepts of state and transition. Its descriptions are concrete, easier to read and understand and it seems to be better adapted to the description of "local" properties of execution sequences.

3. Fair Transition System Specification

Motivated by above observations, we present a new kind of formalism of specification: fair transition system specification(FTSS). It combines the best features of temporal logic and state machine methods.

3.1 The Underlying Model

As a computational model, we present the model of fair transition system(FTS). The presentation and discussion of this model follows [10].

A fair transition system S is a six-tuple $<V, \Sigma, \Theta, T, WF, SF>$, where

- $V=\{u_1, u_2, \cdots, u_n\}$: A finite set of state variables.
- Σ : A set of states.
- Θ : initial condition. A state s satisfying Θ, i.e., $s|=\Theta$, is called an *initial state*.
- T: A finite set of transitions. Each transition $\tau \in T$ is a function τ: $\Sigma \rightarrow 2^{\Sigma}$.
- $WF \subseteq T$: A set of weakly fair transitions.
- $SF \subseteq T$: A set of strongly fair transitions.

A transition τ is *enabled* on s if $\tau(s) \neq \phi$, otherwise, τ is *disabled* on s. We include in T a transition τ_I, called the *idling transition*. It is an identity transition, i.e., $\tau_I(s)=\{s\}$ for every state s.

For each transition τ, we associate with an assertion $\rho_\tau(V, V')$, called the *transition relation*. It relates a state s $\in \Sigma$ to its τ-successor $s' \in \tau(s)$ by referring to both primed and unprimed versions of the state variables. A primed version of a state variable refer to its value in s', while an unprimed version of the same variable refers to its value in s.

Given an FTS S: $<V, \Sigma, \Theta, T, WF, SF>$, we define a **computation** of S to be an infinite sequence of states $\sigma: s_0, s_1, s_2, \cdots$ satisfying the following requirements:

- *Initiality*: The first state s_0 is initial, i.e., $s_0|=\Theta$.
- *Consecution*: For each state pair of consecutive states s_i, s_{i+1} in σ, $s_{i+1} \in \tau(s_i)$ for some $\tau \in T$.
- *Weak fairness*: For each $\tau \in WF$, it is not the case that τ is continually enabled beyond some position in σ but taken only finitely many times.
- *Strong fairness*: For each $\tau \in SF$, it is not the case that τ is enabled infinitely many times in σ but taken only finitely many times.

We denote by *Comp(S)* the set of all computations of the FTS S.

3.2 Temporal Logic

As a specification language, we take temporal logic.

The temporal logic we use is syntactically and semantically similar to other linear time temporal logics[14,12], containing the temporal operators O (*next operator*) and U (*until operator*) . We use the next operator in two different ways: as a temporal operator applied to formulas and as a temporal operator applied to terms. In the latter case, we denote $Ot \equiv t^+$, called the next value of t. Additional temporal operators can be introduced as abbreviation, e.g.,

$\Diamond p$ for $true\, U p$, $\Box p$ for $\sim\Diamond\sim p$, $p W q$ for $\Box p \vee (p U q)$ and $p \Rightarrow q$ for $\Box(p \rightarrow q)$.

3.3 The Temporal Semantics of a Fair Transition System

In this section, we construct a temporal formula *Sem(S)*, called the *temporal semantics* of S. It holds on a model σ iff σ is a computation of S. This construction was firstly presented by Pnueli in [13].

Given an FTS S: $<V, \Sigma, \Theta, T, WF, SF>$. Firstly, we introduce several formulas that express different properties of computations of S.

· $En(\tau)$: $(\exists V')\ \rho_{\tau}(V, V')$. It expresses that transition τ is enabled.

· $taken(\tau)$: $\rho_{\tau}(V, V^{+})$. It means that transition τ is taken.

· $wf(\tau)$: $\Diamond\Box\ En(\tau) \rightarrow \Box\Diamond taken(\tau)$. This formula expresses weak fairness, i.e., any sequence satisfying formula $wf(\tau)$ is weakly fair w.r.t. τ.

· $sf(\tau)$: $\Box\Diamond En(\tau) \rightarrow \Box\Diamond taken(\tau)$. This formula expresses strong fairness, i.e., any model satisfying formula $sf(\tau)$ is strongly fair w.r.t. τ.

For a given FTS S, we define the temporal semantics formula $Sem(S)$ by:

$$Sem(S):\ \Theta \wedge \Box \bigvee_{\tau \in T} taken\ (\tau) \wedge \bigwedge_{\tau \in WF} wf(\tau) \wedge \bigwedge_{\tau \in SF} sf(\tau) \qquad (*), \qquad \text{where}$$

· Θ ensures that the initial state satisfies the initial condition Θ.

· $\Box \bigvee_{\tau \in T} taken\ (\tau)$ ensures that every step is taken by the application of some transition $\tau \in T$.

· $\bigwedge_{\tau \in WF} wf(\tau)$ ensures that the sequence satisfies all the weakly fairness requirements.

· $\bigwedge_{\tau \in SF} sf(\tau)$ ensures that the sequence satisfies all the strongly fairness requirements.

The four clauses correspond to and ensure the four requirements a sequence has to satisfy in order to be a computation of S. Consequently, we can get the fact that the formula $Sem(S)$ precisely characterizes the computation of S.

4.4 Fair Transition System Specification

In this section, we firstly give some definitions, then we give out our fair transition system specification(FTSS).

Definition 1: A *transition module M* is a system M:$<V, U, \Sigma, \Theta, T, WF, SF>$, where S_M: $<V, \Sigma, \Theta, T, WF, SF>$ is a fair transition system and $U \subseteq V$ is a subset. We refer to S_M as the *body* of M and to U the *internal variables* of M.]]

Definition 2: A *run* of a transition module M with body S_M and internal variables U is any U-variant of a computation of S_M, i.e., any model that differs from a computation of S_M by at most the interpretation of the variables in U.]]

In the following, we propose our fair transition system specification style.

Definition 3: A temporal formula ϕ is said to be in fair transition system specification style, if it has form ϕ: $\exists U.Sem(S)$, where S is a fair transition system with state variables V, U is a subset of V, $Sem(S)$ is the temporal semantics of system S.]]

Consequently, for a given transition module M with body S_M: $<V, \Sigma, \Theta, T, WF, SF>$ and internal variables $U \subseteq V$, our FTSS has form:

$$\exists\ U.\ [\ \Theta \wedge \Box \bigvee_{\tau \in T} taken\ (\tau) \wedge \bigwedge_{\tau \in WF} wf(\tau) \wedge \bigwedge_{\tau \in SF} sf(\tau)\ ] \qquad (**) \qquad , \text{where}$$

· Θ is an assertion specifying the initial values of variables.

· $\Box \bigvee_{\tau \in T} taken\ (\tau)$ is next-state relation of the transition module M.

· $\bigwedge_{\tau \in WF} wf(\tau)$ and $\bigwedge_{\tau \in SF} sf(\tau)$ are conjunctions of formulas of the form $wf(\tau)$ and $sf(\tau)$. They respectively represent weak fairness assumption and strong fairness assumption.

· U is the internal variables of M.

Let $L = \bigwedge_{\tau \in WF} wf(\tau) \wedge \bigwedge_{\tau \in SF} sf(\tau)$. Our FTSS $(**)$ has the following interpretation:

There is some way of choosing values for U such that (a) Θ is true in the initial state, (b) every step of M is either $taken(\tau)$ for some transition $\tau \in (T \setminus \{\tau_I\})$or $taken(\tau_I)$ for transition τ_I, which leaves all variables unchanged, and (c) the entire behavior of M satisfies formula L, i.e., it satisfies all the weak fairness and strong fairness assumptions.

Obviously, our FTSS $(**)$ characterizes precisely the run of module M, i.e., a model σ satisfies $(**)$ iff it is a run of M. It can be stated as following:

Proposition : A model σ satisfies FTSS (**) iff σ is a run of transition module M.]]

In the following, we give some discussions about our FTSS.

A property is a set of behaviors. A finite behavior is a finite sequence of states. We say a finite behavior satisfies a property F iff it can be continued to an infinite behavior in F. Our FTSS specifies two basic types of properties of system: safety properties and liveness properties. A property S is a safety property iff the following condition holds: F contains a behavior iff it is satisfied by every finite prefix of the behavior[2]. A property L is a liveness property iff any finite behavior can be extended to a behavior in L [2]. So in our FTSS, $\Theta \wedge \Box \bigvee_{\tau\in T} taken(\tau)$ is a safety property. $wf(\tau)$ and $sf(\tau)$ are liveness properties. As indicated in [2], safety properties are closed set in a topology on the set of all behaviors and liveness properties are dense sets. In a topological space, every set can be written as the intersection of a closed set and a dense set. So any property P can be written as $S \wedge L$, where S is a safety property and L is a liveness property. Our FTSS approach adopts the *safety-liveness partition* paradigm, its main purpose is to help in achieving completeness. We can always checklist whether we have specified some requirements in each part. However, using arbitrary liveness properties to specifying liveness part is dangerous because it can add unexpected safety properties. It is a common source of errors in temporal logic specifications and a source of incompleteness for proof methods[1,6]. In our FTSS, we avoid such errors by expressing liveness in terms of fairness.

Definition 4: If F is a safety property and L an arbitrary property, then the pair (F, L) is *machine closed* iff every finite behavior satisfying F can be extended to an infinite behavior in $F \wedge L$. The conjunction $F \wedge L$ is called *machine closed specification.*]]

Intuitively, the pair (F, L) is machine closed iff conjoining F to L introduces no additional safety properties. So we can avoid accidentally adding safety properties by writing machine closed specification.

Theorem 1: If F is a safety property and L is the conjunction of a finite or countably infinite number of formulas of the form $wf(\tau)$ and/or $sf(\tau)$ such that each $taken(\tau)$ implies F, i.e., $taken(\tau) \rightarrow F$, then (F, L) is machine closed.

Proof: It is a special case of Proposition 4 of [1]. omitted.]

Theorem 2: If (F, L) is machine closed, U is a set of variables that do not occur free in L and $(\exists U.F)$ is a safety property, then $((\exists U.F), L)$ is machine closed.

Proof: It is similar to Proposition 2 of [1]. omitted.]

In our FTSS (**), the pair $([\Theta \wedge \Box \bigvee_{\tau\in T} taken(\tau)], L)$ is machine closed (by theorem 1), where $L = \bigwedge_{\tau\in WF} wf(\tau) \wedge \bigwedge_{\tau\in SF} sf(\tau)$. Since the variables in U do not occur free in L, so (**) can be rewritten as $(\exists U. [\Theta \wedge \Box \bigvee_{\tau\in T} taken(\tau)]) \wedge L$. By above definition, $\exists U. [\Theta \wedge \Box \bigvee_{\tau\in T} taken(\tau)]$ is a safety property, so (**) is machine closed (by theorem 2). Our FTSS can not introduce additional safety property and our specification process is consistent and complete.

Comparing the temporal semantics formula (*) with our FTSS (**), we know that the main distinction between them relies upon the existential quantification over internal variables. It is an effective mechanism to increase the expressive power of the temporal logic and decrease the complexity of the specifications. Having the existential quantification over internal variables guarantees complete freedom of any implementation bias. It is the same as hiding in programming languages. The precise meaning of existential quantification over internal variables is the essential of understanding our FTSS.

4. Assessment

Comparing with other specification methods, our presented approach has the following features:

· **Simplicity of the specification**

A formal specification is meant to be read by human beings, so it is natural to require it is easy for them to read and understand the specification. Temporal logic specifications are hard to understand. Our approach describes a method for writing formal specifications, it combines the best features of temporal logic and state machine methods, so it is simple and easy to understand.

· **Simplicity of the semantics**

Simplicity of the specification formalism can be deceiving. True simplicity of a formal specification requires that the formal semantics of the specification language be simple. The real complexity of a specification must take into account the difficulty in understanding its formal meaning. Our approach bases on the temporal logic framework of Manna-Pnueli, its temporal semantics is less complicated.

· **Completeness of the proof method**

Completeness of a formal system means that every semantically valid assertion is provable. As mentioned above, our approach bases on the framework of Manna-Pnueli , this temporal logic theory has been widely and thoroughly investigated. Many existing complete temporal proof systems can be reused in our method. Our proof method is complete.

· **Practicality**

While temporal logic specifications are less likely to overspecify the system, they are much more likely to underspecify it by omitting important constraints. In practice, temporal logic methods are hard to use because they don't tell one where to start or when to stop. In contrast, our FTSS method provides a well structured approach to writing specifications, it uses two very primitive concepts, i.e., state and transition. Its descriptions are concrete and the generated specifications are abstract programs. It is easier to read and understand by people not familiar with logics and more likely to be accepted by them.

5. An Example: A Lossy-Transmission Protocol

In this section, we present an example of a lossy-transmission protocol to illustrate our approach. The protocol communicates with its environment via two pairs of "wires", each pair consisting of a *val* wire that holds a messages and a Boolean-valued *bit* wire. A message *m* is sent over a pair of wires by setting the *val* wire to *m* and complementing the *bit* wire. The receiver detects the presence of a new message by observing that the *bit* wire has changed value. Input to the transmission arrives on the wire pair (*ival, ibit*), and output is sent on the wire pair (*oval, obit*). (Shown in Figure 1.)

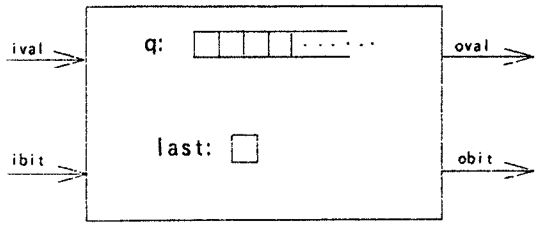

Figure 1. A Lossy-trasmission Protocol

There is no acknowledgment protocol, so inputs are lost if they arrive faster than the transmission processes them. The property guaranteed by this lossy transmission protocol is that the sequence of output messages is a subsequence of the sequence of input messages.

This protocol has been described in [4] by transition-axiom method and in [1] by *TLA*-based approach respectively. Taken it as illustration and comparison, we give its FTSS formalism.

The FTSS of the lossy-transmission protocol is a temporal formula, which mentions the four variables *ibit, obit, ival* and *oval*, as well as two internal variables: *q*, which equals the sequence of messages received but not yet output, and *last*, which equals the value of *ibit* for the last received message. The variable *last* is just used to prevent the same message from being received twice. These six variables are *flexible* variables, their values can change during a behavior. For specification purpose, we also can introduce a *rigid* variable *Mes* denoting the set of possible messages. It has the same value throughout a behavior.

We firstly introduce the following notations. ( ) denotes the empty sequence; (m) denotes the singleton sequence having m as its one element; $\cdot$ denotes concatenation; $Head(\sigma)$ denotes the first element of the sequence σ and $Tail(\sigma)$ denotes the sequence obtained by removing the first element of σ.

We define a transition module M_{FTSS}=$<V, U, \Sigma, \Theta, T, WF, SF>$, where

V: $\{ ival, ibit, oval, obit, q, last \}$, U: $\{ q, last \}$, Θ : $q=(\,) \wedge (ival, oval \in Mes)$,

T: $\{ \tau_I, \tau_1, \tau_2, \tau_3 \}$, whose transition relations are given by

ρ_{τ_I} : $(ival', ibit', oval', obit', q', last') = (ival, ibit, oval, obit, q, last)$, i.e., τ_I is identity transition.

ρ_{τ_1} : $(ibit' = \sim ibit) \wedge (ival' \in Mes) \wedge (obit', oval', q', last') = (obit, oval, q, last)$.

ρ_{τ_2} : $(last \neq ibit) \wedge q' = q \cdot (ival) \wedge last' = ibit \wedge (ibit', obit', ival', oval') = (ibit, obit, ival, oval)$.

ρ_{τ_3} : $q \neq (\,) \wedge oval' = Head(q) \wedge q' = Tail(q) \wedge obit' = \sim obit \wedge (ibit', ival', last') = (ibit, ival, last)$.

WF : $\{ \tau_3 \}$, SF : $\{ \tau_2 \}$.

The FTSS of the lossy-transmission protocol is given as following:

$$\exists\, q, last: [\, \Theta \wedge \Box \bigvee_{\tau \in T} taken\ (\tau) \wedge \bigwedge_{\tau \in WF} wf(\tau) \wedge \bigwedge_{\tau \in SF} sf(\tau) \,] \qquad (*)$$

Let $S = \exists\, q, last$: $[\, \Theta \wedge \Box \bigvee_{\tau \in T} taken\ (\tau) \,]$, $L = \bigwedge_{\tau \in WF} wf(\tau) \wedge \bigwedge_{\tau \in SF} sf(\tau)$, so (*) equals to $S \wedge L$, where S is the safety part and L is the fairness part. We will discuss in the following that L is also the liveness part.

Formula $[\Theta \wedge \Box \bigvee_{\tau \in T} taken\ (\tau)]$ is the internal specification of the safety part S. It specifies all sequences of values that may be assumed by the protocol's six variables, including the internal variables q and *last*. Its first conjunct asserts that Θ is true in the initial state. Its second conjunct $\Box \bigvee_{\tau \in T} taken\ (\tau)$ asserts that every step is either a transition $\tau \in (T \setminus \{\tau_I\})$ or else a transition τ_I, which leaves all six variables unchanged.

Formula S is the actual safety part of our FTSS, in which the internal variables q and *last* have been hidden. A behavior satisfies S iff there is some way to assign sequences of values to q and *last* such that formula $[\Theta \wedge \Box \bigvee_{\tau \in T} taken\ (\tau)]$ is satisfied. The free variables of S are *ibit*, *obit*, *ival* and *oval*, so S specifies what sequences of values these four variables can assume.

Formulas $wf(\tau_3)$ and $sf(\tau_2)$ are fairness properties. Property $wf(\tau_2)$ asserts that if transition τ_3 is enabled forever, then infinitely many *taken* (τ_3) steps must occur. This property implies that every message reaching the q is eventually output. Property $sf(\tau_2)$ asserts that if transition τ_2 is enabled infinitely often, then infinitely many *taken* (τ_2) steps must occur. It implies that if infinitely many inputs are sent, then the q must receive infinitely many of them. So formula $L = \bigwedge_{\tau \in WF} wf(\tau) \wedge \bigwedge_{\tau \in SF} sf(\tau)$ implies the liveness property that an infinite number of inputs produce an infinite number of outputs. It ensures the protocol satisfies the property that the sequence of output messages is subsequence of the sequence of input messages.

It is most important to realize that, although our FTSS uses a list-valued variable q to express the desired property, this carries no implication that a similar structure must be present in the implementation. This is just a device for a simpler presentation of the specification. The device makes our FTSS easy to understand. What makes our FTSS abstract and free of any implementation bias is the fact that the variables q and *last* are quantified over, which is equivalent to hiding in programming languages.

6. Concluding Remarks

In this paper, we present a new kind of formalism of specification: fair transition system specification(FTSS). Our work mainly bases on the works in [13,14,15,4,6]. In [13], Pnueli firstly gave a temporal semantics of a concurrent programs, which is a temporal formula and precisely characterizes the computations of a concurrent programs. In consideration of the actual implementation, Pnueli indicated in [15] the temporal run semantics of a concurrent programs, i.e., the temporal semantics with some hidden variables. It provides a base for further applications of temporal logic. Based on these works and works in [4,6], we present complete temporal run semantics formula as our actual specification formalism. We further discuss each component of our specification formalism and arrive the conclusions that our specification formalism is machine-closed, and our specification process is consistent and complete.

In [4], Lamport presented a transition-axiom method, which also integrated temporal logic method with state machine method. However, the semantics of its *allowed changes*-operator introduced in his method appears to be rather complicated. Recently, Lamport presented a temporal logic of actions (*TLA*), which combines a logic of actions with a standard temporal logic[6]. Based on the transition-axiom method, he gave out his *TLA* specification style. It is simple and easy to understand.

Our FTSS bases on the temporal logic framework of Manna-Pnueli[10]. Its temporal semantics is less complicated, it doesn't deal with state variables explicitly, and therefore, is more convenient to work with. Note that the semantics of *next* operator in our framework is much simpler than that of actions in *TLA*, because an action is similar to a non-deterministic program in dynamic logic which represents sets of pairs of states. Another important advantage of our framework is that temporal logic theory has been very widely and thoroughly investigated. There have been many comprehensive proof systems developed for proving temporal properties of concurrent systems, such as [8,9,10,11]. Our FTSS allows us to make full use of these proof systems. These features show that our FTSS is very promising.

References

[1] M.Abadi and L.Lamport, An Old-Fashioned Recipe for Real Time, *ACM Trans. on Prog. Lang. and Sys.*, 16(5):1543-1572, 1994.

[2] B.Alpern and F.B.Schneider, Defining Liveness, *Inf. Process. Lett.*, 21(4): 181-185, 1985.

[3] D.Harel, Statecharts: A Visual Formalism for Complex System, *Sci. Comp. Prog.*, 8: 231-274, 1987.

[4] L.Lamport, Specifying Concurrent Program Modules, *ACM Trans. on Prog. Lang. and Sys.*, 5: 190-222, 1983.

[5] L.Lamport, What Good is Temporal Logic ?, *Information Processing 83*, R.E.A.Mason (eds.), North-Holland, 1983: 657-68.

[6] L.Lamport, The Temporal Logic of Actions, *ACM Trans. on Prog. Lang. and Sys.*, 16(3):872-923, 1994.

[7] N.Lynch and M.Tuttle, An Introduction to Input/Output Automata, *CWI-Quarterly*, 2(3): 219-246, 1989.

[8] Z.Manna and A.Pnueli, Verification of Concurrent Programs: a Temporal Proof Systems, *Foundations of Computer Science IV,Distributed System: Part 2*, 163-255, 1983.

[9] Z.Manna and A.Pnueli, Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Systems, *Sci. Comput. Prog.*, 32: 257-289, 1984.

[10] Z.Manna and A.Pnueli, The Temporal Logic of Reactive and Concurrent System: Specification, Springer-Verlag, New York, 1991.

[11] Z.Manna and A.Pnueli, Completing the Temporal Picture, *Theor. Comp. Sci.*, 83(1): 97-130, 1991.

[12] R.Milner, A Calculus of Communicating System, *Lec. Notes in Comp. Sci. 94*, Springer-Verlag, 1980.

[13] A.Pnueli, The Temporal Semantics of Concurrent Programs, *Theor. Comp. Sci.*, 13: 1-20, 1981.

[14] A.Pnueli, Specification and Development of Reactive Systems, In: *Information Processing 86*, IFIP, North-Holland, 1986: 845-858.

[15] A.Pnueli, System Specification and Refinement in Temporal Logic, In: R.Shyamasundar (ed.), *Soft. Tech. and Theor. Comp. Sci.* , Lect. Note. in Comp. Sci. Vol.652, Springer-Verlag, 1992.