

Formal Analysis of Real-Time Systems with SAM ^{*}

Huiqun Yu^{1,2}, Xudong He¹, Yi Deng¹, Lian Mo¹

¹School of Computer Science
Florida International University
Miami, FL 33199, USA
{yhq|hex|deng|lmo01}@cs.fiu.edu

²Department of Computer Science and Engineering
East China University of Science and Technology
Shanghai 200237, China

Abstract. The Software Architecture Model (SAM) is a general software architecture model based on a dual formalism combining Petri nets and temporal logic. This paper proposes a formal method for modeling and analyzing real-time systems with SAM. A high level Petri net and a linear time temporal logic are used as the theoretical basis for SAM. Behaviors of real-time systems are modeled by Petri nets, while their properties are specified by temporal logic. By translating Petri nets into clocked transition systems, we can apply the Stanford Temporal Prover to automating the analysis of real-time systems. A case study of interactive multimedia documents demonstrates our approach to modeling and analyzing real-time systems with SAM.

Keywords: Real-time system, SAM, Petri net, temporal logic, model, analysis

1 Introduction

SAM is a general software architecture model for developing and analyzing software architecture specifications. The theoretical basis of SAM is a combination of two complementary formal notions: Petri nets [Mur89] and temporal logic [Pnu77], with the choice of Petri nets and temporal logic open. In [WHD99], Time Petri Nets (TPNs) and Real-Time Computation Tree Logic (RTCTL) are used, while in [HD02], Predicate Transition Nets (PrTNs) and a first order linear time temporal logic (LTL) are used. Other kind of Petri nets [Tro95] and/or temporal logic [Eme90] can also be used. In the SAM framework, Petri nets are used to define the behavior models of systems and temporal logic is used to specify system properties (or constraints). The sound mathematical foundation

^{*} Supported in part by the NSF under grants HDR-9707076 and CCR-0098120, and by NASA under grant NAG 2-1440.

of SAM can help to detect and eliminate design errors early in the development cycle, avoid costly fixes at the implementation stage.

Several earlier works dealt with real-time system modeling and analysis based on Petri nets [MF76,CR83,LS87], but few built tools to support the analysis process. This paper shows how to model and analyze real-time systems in the SAM framework. Our work differs from [BD91] in that we use high-level Petri nets and a first-order temporal logic as our formal foundations. In addition, we formalize and automate analysis process, which is believed to be vital for practical use of formal methods [CW96].

The rest of the paper is organized as follows: Section 2 gives a brief introduction to SAM and its application in modeling and specification of real-time systems. Section 3 proposes our analysis method. Section 4 presents a detailed case study which shows how to apply our method to formally modeling interactive multimedia documents and analyzing their consistency. Section 5 is the conclusion.

2 Real-Time Modeling in SAM

2.1 The SAM Specification Structure

In SAM, a software architecture is defined by a hierarchical set of compositions in which each composition consists of a set of components, a set of connectors and a set of constraints to be satisfied by the interacting components. Basically, behaviors of components and connectors are modeled by Petri nets, while their properties (or constraints) are specified by temporal logic formulas. The interfaces of components and connectors are *ports*. Figure 1 shows a graphical view of a SAM architecture model, where the higher level component *A3* is decomposed into a lower level sub-architecture.

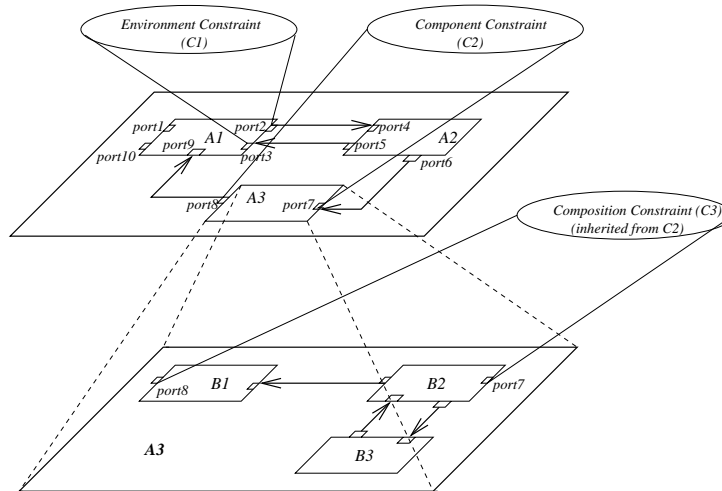


Figure 1. A SAM architecture model

Several correctness criteria of SAM models are identified in [HD02]. One basic criterion is *element correctness* which requires the property specification S to hold in the corresponding behavior model B , i.e. $B \models S$.

2.2 PrTNs and Behavior Modeling

A *PrTN* is defined with an underlying first-order language \mathcal{L}_P with typed domains. The logic contains ordinary symbols for variables, constants, functions, predicates, logic connectives, and quantifiers. The set of variables V is partitioned into two disjoint sets: D (the set of discrete variables) and C (the set of clock variables). One of the clock variables T is designated as the *master clock*. The sets of terms and well-formed formulas of \mathcal{L}_P are denoted by *Term* and *Wff*. Let M_s denote multi-set of terms of the special forms $\{k_1 a_1, \dots, k_n a_n\}$, where $k_i \in \text{Nat}$ and a_i is a ground term.

A PrTN in \mathcal{L}_P is a tuple $\pi = (P, Tr, F, L, R, m_0)$, where





- P : a finite set of places;
- Tr : a finite set of transitions ;
- $F \subseteq P \times Tr \cup Tr \times P$: the set of arcs;
- $L : F \rightarrow Ms$: labels;
- $R : Tr \rightarrow Wff$: transition constraints. We assume that R contain both functional constraint R_F and real-time constraint R_T . In particular, a real-time constraint for transition t is a relation of the form $c_t \in [\ell, u]$, where c_t is a local clock associated with transition t , ℓ is called the *earliest firing time* (EFT) for t , and u the *latest firing time* (LFT).
- m_0 : the initial marking.

Remarks:

- As far as real-time specification is concerned, a PrTN is similar to a TPN. A real-time constraint R_T in a PrTN is the counterpart of *static interval marking* (SIM) in TPN [BD91]. Consequently, they share similar real-time semantics.
- PrTNs allow for complex data structures. In particular, we require that each token in a PrTN contain a time stamp (global time).

Graphical Notations: A real-time constraint for transition t is generally represented by a time interval $[\ell, u]$. For simplicity, we use several graphical notations as defined in Table 1.

Table 1. Several graphical notations

<i>Description</i>	Instantaneous	Acurate time	Lower bounded	Upper bounded
<i>Notation</i>				
<i>Meaning</i>	$l = u = 0$	$l = u = a$	$l = a, u = \infty$	$l = 0, u = a$

A general form for a state S of a PrTN can be defined as a pair $S = (m, I)$ consisting of:

- a marking m ;
- a firing interval set I which is a vector of possible firing times. The number of entries in this vector is given by the number of transitions enabled by marking m .

Given a marking m , a transition $t \in Tr$ is *enabled* if the following condition is satisfied : $\forall p \in P. (L(p, t) \subseteq m(p)) \wedge R_F(t)$.

A transition t is *firable* from state $S = (m, I)$ at time $\tau + \theta$ if both of the following conditions hold:

- t is enabled by marking at time τ : $\forall p \in P. (L(p, t) \subseteq m(p)) \wedge R_F(t)$
- the relative firing time θ is not smaller than EFT of t and not greater than the smallest of the LFT's of all the transitions enabled by the marking M .

Let us assume transition t be firable at time $c + \theta$ from state $S = (m, I)$. Then the state $S' = (m', I')$ *reached from* S by firing t at the relative time θ can be computed as follows.

- m' defined by $m'(p) = m(p) - L(p, t) + L(t, p)$, for all $p \in P$. We call m' the t -successor of m , denoted by mtm' .¹
- I' is computed in three steps:
 - Remove from the expression of I the intervals that are related to the transitions disabled when t is fired.
 - Shift of the value θ towards the origin of times all remaining firing intervals, i.e., the intervals that remain enabled and so remain in I , and truncate them, when necessary, to nonnegative values.
 - Introduce in the domain the time interval of new transitions enabled.

The behavior “transition t is firable from state S at time c and its firing leads to state S' ” is denoted as:

$$S \xrightarrow{(t, c)} S'$$

A *firing schedule* is a sequence of pairs

$$(t_1, c_1) \bullet (t_2, c_2) \bullet \dots \bullet (t_n, c_n) \dots$$

in which $t_1, t_2, \dots, t_n, \dots$ are transitions and $c_1, c_2, \dots, c_n, \dots$ are times. This firing schedule is *feasible* from a state (m_0, I_0) iff there exist states $(m_1, I_1), (m_2, I_2), \dots, (m_n, I_n), \dots$ such that:

$$(m_0, I_0) \xrightarrow{(t_1, c_1)} (m_1, I_1) \xrightarrow{(t_2, c_2)} (m_2, I_2) \dots \xrightarrow{(t_n, c_n)} (m_n, I_n) \dots$$

A *run* of a PrTN is a sequence of observations $(m_0, c_0), (m_1, c_1), (m_2, c_2), \dots$ in the above firing sequence. We define the *computation* of π , denoted by $Comp(\pi)$, to be the set of all possible runs.

¹ Note that multiple non-conflicting transitions are capable of being fired simultaneously. In this case, m' equals to the final marking by arranging transitions to be fired one by one.

2.3 LTL and Property Specification

The requirement specification language is LTL. A *temporal formula* in LTL is constructed out of state formulas to which we apply the boolean connectives and temporal operators. Future temporal operators include \Box (always), \mathcal{W} (wait-for), \Diamond (eventually), \bigcirc (next), and \mathcal{U} (until). Past operators include \ominus (previously), \boxplus (so-far), \diamondleftarrow (once), and \mathcal{S} (since). We write $p \Rightarrow q$ as an abbreviation of $\Box(p \rightarrow q)$. A *past formula* is one that contains no future temporal operators.

A *model* for a temporal formula p is an infinite sequence of states $\sigma : s_0, s_1, \dots$, where each state s_j provides an interpretation for the variables mentioned in p . The semantics of a temporal formula p in a given model σ and position j is denoted by $(\sigma, j) \models p$. We define:

- For a state formula p , $(\sigma, j) \models p \Leftrightarrow s_j \models p$
- $(\sigma, j) \models \neg p \Leftrightarrow (\sigma, j) \not\models p$
- $(\sigma, j) \models p \vee q \Leftrightarrow (\sigma, j) \models p$ or $(\sigma, j) \models q$
- $(\sigma, j) \models \Box p \Leftrightarrow (\sigma, i) \models p$ for all $i \geq j$
- $(\sigma, j) \models \Diamond p \Leftrightarrow (\sigma, i) \models p$ for some $i \geq j$

These are all the LTL operators we will use in this paper. The interested reader is referred to [MP92,MP95] for detailed presentation of temporal logic.

Two important classes of properties are: *safety* and *response*.

- Safety properties are those that can be expressed by a formula $\Box\psi$, for some past formula ψ .
- Response properties are those that can be expressed by a formula $p \Rightarrow \Diamond q$, for some past formula p and q .

Since clocks are explicit variables in PrTNS, temporal formulas may freely refer to them. This includes, in particular, the master clock T , allowing the natural expression of time-dependent properties. For example, $\Box(\ell \leq T \wedge T \leq u \rightarrow p)$ states that p holds over the time interval $[\ell, u]$.

One interesting point is that many useful specifications which are progress properties in their untimed version correspond to safety properties when a time bound is added [MP96]. For example, showing that a system reaches and maintains p within time δ is expressed as the invariance $\Box(T \geq \delta \rightarrow p)$.

A model $\sigma : s_0, s_1, \dots$ satisfies a temporal formula if $(\sigma, 0) \models p$. A temporal formula p that is *valid* over a program P specifies a property of P , i.e., states a condition that is satisfied by all computations of P .

Since LTL is an extension of first-order logic, the deductive system for LTL includes axioms and rules from first-order logic, as well as those axioms and rules specific to temporal operators. Typical rules include:

- Modus ponens: $p, p \rightarrow q \vdash q$
- Chain: $p \rightarrow \Diamond q, q \rightarrow \Diamond r \vdash p \rightarrow \Diamond r$

3 Formal Analysis of SAM Models

3.1 A Formal Analysis Method

Our analysis method is illustrated in Figure 2, which is based on the Stanford Temporal Prover (STeP) [MAB⁺94]. PrTNs are systematically translated into CTSs. The latter becomes one input to STeP tools. The correctness of LTL formulas against CTSs is verified either by deductive proof, or by model checking.

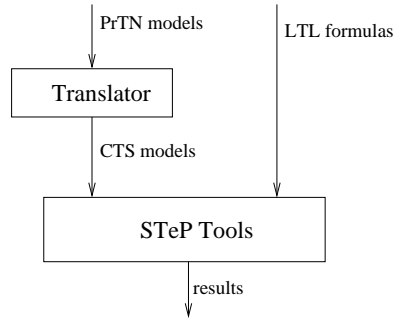


Figure 2. A formal analysis method

The basic verification rule for safety properties is B-INV:

$$\frac{\Theta \rightarrow p, \{p\}\mathcal{T}\{p\}}{\Box p}$$

A verification session in STeP begins by loading a program or transition system that describes the system of interest and entering a temporal-logic formula that expresses one or more properties to be proved. The formula becomes the root goal of a proof tree. Verification can be performed by the model checker or by deductive means. The most common route to proceed proof is to apply a verification rule, which generates the first level of subgoals of the proof tree. If all subgoals are established automatically by the Simplifier, the proof is finished. For those verification conditions that are not proved automatically, an interactive Gentzen-style theorem prover is usually invoked.

3.2 Clocked Transition Systems

A CTS is an extension of transitions systems to account for continuous real-time. The basic idea is to add explicit real-valued clock variables to the system, which measure the passing of time. Formally, a CTS $\mathcal{S} = (V, \Theta, \mathcal{T}, \Pi)$ consists of:

- V : A finite set of *system variables*, partitioned into a set D of discrete variables and a set C of real-valued clock variables. We define a *state* s to be a type-consistent interpretation of V . The set of all state is denoted by Σ .
- Θ : The *initial condition*, a satisfiable assertion characterizing the possible initial states. We require that Θ implies $T = 0$.

- \mathcal{T} : A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is a function

$$\tau : \Sigma \rightarrow 2^\Sigma,$$

mapping each state $s \in \Sigma$ into a (possibly empty) set of states $\tau(s) \subseteq \Sigma$. Each state in $\tau(s)$ is called a τ - *successor* of s .

We say that the transition τ is *enabled* on the state s if $\tau(s) \neq \emptyset$. Otherwise, we say that τ is *disabled* on s .

- Π : The *time-progress condition*. This is an assertion over V , used to specify a global restriction on the progress of time.

To account for the passage of time, a *tick* transition is added to the set of transitions. The transition relation for *tick* is given by

$$\rho_{tick} : \exists \Delta. \left(\begin{array}{c} \Delta > 0 \wedge \forall t \in [0, \Delta]. \Pi(D, C + t) \\ \wedge \\ D' = D \wedge C' = C + \Delta \end{array} \right)$$

where $C' = C + \Delta$ stands for $\bigwedge_{c \in C} (c' = c + \Delta)$. Thus, *tick* preserves the values of all discrete variables and uniformly increments all clocks by an amount Δ that satisfies the global time-progress condition Π . We denote the set $\mathcal{T} \cup \{tick\}$ by \mathcal{T}_T .

A *run* of a CTS $\mathcal{S} : (V, \Theta, \mathcal{T}, \Pi)$ is an infinite sequence of states $\sigma : s_0, s_1, \dots$ such that (1) $s_0 \models \Theta$ and (2) for each $j \geq 0$ there is some $\tau \in \mathcal{T}_T$ such that $s_{j+1} \in \tau(s_j)$. In this case we say that τ is taken at s_j . A state s is *reachable* if it appears in some run of \mathcal{T} . A run is a *computation* if it satisfies *time divergence*, that is, the value of T increases beyond any bound. $Comp(\mathcal{S})$ is the set of all the computations of \mathcal{S} .

3.3 From PrTNs to CTSs

We adopt the idea from [MP96] to associate lower and upper time bounds with transitions. A lower bound ℓ on transition τ is enforced by associating a clock c_τ and adding the condition $c_\tau \geq \ell$ to τ 's enabling condition. When the enabling condition for τ first becomes true, c_τ is reset to 0. Upper bounds appear as constraints on Π : if u is the upper bound on τ , then $c_\tau \leq u$ appears as a conjunct in Π .

Given a PrTN $\pi = (P, Tr, F, L, R, m_0)$ as defined in section 2.2, we define a corresponding CTS by $\mathcal{S} = (V, \Theta, \mathcal{T}, \Pi)$, where

- The set of local variables: $V = \{v_p \mid p \in P\} \cup \{c_t \mid t \in Tr\} \cup \{T\}$.
- Initial conditions:

$$\Theta = \bigwedge_{p \in P} (v_p = m_0(p)) \bigwedge_{t \in Tr} (c_t = 0) \wedge (T = 0)$$

- For every transition $t \in Tr$, there is a corresponding transition τ_t , whose transition relation is defined as $\rho_{\tau_t} \stackrel{\text{def}}{=} en(\tau_t) \rightarrow f(\tau_t)$, where

$$en(\tau_t) = R_F(t) \wedge \bigwedge_{p \in P} (v_p \supseteq L(p, t))$$

$$f(\tau_t) = \bigwedge_{p \in P} (v'_p = v_p - L(p, t) + L(t, p))$$

The set of all possible transitions $\mathcal{T} = \{\tau_t \mid t \in Tr\}$

- For the special transition *tick*: the transition relation is the same as in section 3.2.
- The time-progress condition Π is : $\bigwedge_{t \in Tr} (c_t \leq u_t)$

Theorem 1 (Soundness of the transformation). *Given a PrTN π , suppose $S_\pi = (V, \Theta, \mathcal{T}, \Pi)$ be the CTS obtained from π using the above transformation rules.*

- For any run $\sigma = (m_0, c_0), (m_1, c_1), (m_2, c_2), \dots \in Comp(\pi)$, $\forall i \in Nat$, there exists a mapping s_i from V to the correspondence domain such that $\forall v_p \in V. s_i(v_p) = m_i(p)$, $s_i(T) = c_i$, and $\sigma' = s_0, s_1, s_2, \dots \in Comp(S_\pi)$.
- For any state sequence $\sigma = s_0, s_1, s_2, \dots \in Comp(S_\pi)$, $\forall p \in P, i \in Nat$, let $m_i(p) = s_i(v_p)$, $c_i = s_i(T)$, then $\sigma' = (m_0, c_0), (m_1, c_1), (m_2, c_2), \dots \in Comp(\pi)$.

Proof:

(1) Suppose $\sigma = (m_0, c_0), (m_1, c_1), (m_2, c_2), \dots \in Comp(\pi)$. Let the corresponding transition sequence be $t_0 t_1 t_2 \dots$. By definition, the following predicate holds:

$$\forall i \in Nat, \forall p \in P. (m_i(p) \supseteq L(p, t_i) \wedge R_F(t_i)) \wedge (m_{i+1} = m_i(p) - L(p, t_i) + L(t_i, p))$$

$\forall v_p \in D, c \in C$, let $s_0(v_p) = m_0(p)$, $s_0(c) = 0$, we get $s_0 \models \Theta$. For each $i \in Nat$, $(s_i, s_{i+1}) \models \tau_{t_i}$ holds. Consequently, $s_0, s_1, s_2 \dots$ is a run of S_π . Therefore $\sigma' = s_0, s_1, s_2, \dots \in Comp(S_\pi)$.

(2) Suppose $\sigma = s_0, s_1, s_2, \dots \in Comp(S_\pi)$. Let the corresponding transition sequence be $\tau_{t_0} \tau_{t_1} \tau_{t_2}, \dots$. So $(m_i, c_i) \in [(m_0, c_0) >^2]$ follows from the definition of m_i, c_i , and τ_{t_i} . Consequently, $\sigma' = (m_0, c_0), (m_1, c_1), (m_2, c_2), \dots \in Comp(\pi)$.

□

4 An Example: Consistency of SMIL Documents

SMIL is the W3C format for multimedia synchronization on the web [Rec98]. It uses the XML to define a set of markup tags to associate timing and positioning relationships between multimedia objects, such as audio, video, image and text.

² The notation $(m_i, c_i) \in [(m_0, c_0) >^2]$ denotes that (m_i, c_i) is reachable from the initial observation (m_0, c_0) .

One important issue about specification of temporal constraints of an interactive multimedia document is how to meet the user's QoS requirements. In the case of SMIL, the tag *switch* is applied to express the user's preferences concerning bit-rate transmission capabilities, preferred language, size of screen, alternative media presentation, etc.. However, the satisfaction of user's QoS requirements may lead to unexpected bad timing constraints. By defining formal semantics for SMIL, we can verify the correctness of a SMIL document against its specification.

Consider the following SMIL document.

```

< id='par01' >
  < seq id='seq01' >
    < Img1 dur = '5s'.../ >
    < switch >
      < Audio dur = '20s'.../ >
      < Txt dur = '3s'.../ >
    < /switch >
  < /seq >

  < seq id='seq02' end='id(seq01)(end)' >
    < Video dur = '10s'.../ >
    < Img2 .../ >
  < /seq >
< /par >

```

It consists of a sequence of a video clip (*Video*) followed by an image (*Img2*). This sequence (*Seq02*) must be presented simultaneously with another sequence (*Seq01*) which consists of an image (*Img1*) followed by some related information. This information corresponds to an element of the SMIL operator *switch*, such as an audio segment (*Audio*) or a text (*Txt*). The end of presentation of *Img2* is determined by the end of the sequential presentation of *Img1* and the element chosen in the *switch* operator.

The above SMIL document can be formally modeled by a PrTN in Figure 3.

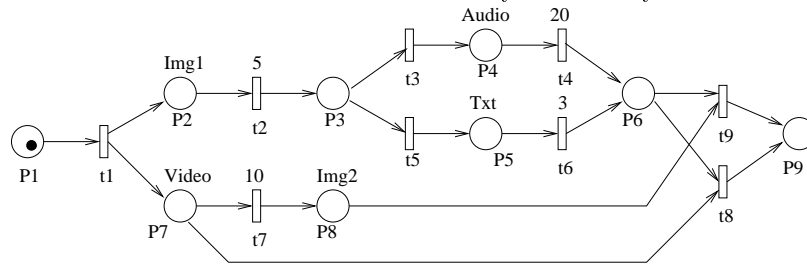


Figure 3. A PrTN for the SMIL document

By applying the translation rules, we can obtain a corresponding CTS $\mathcal{S} = (V, \Theta, \mathcal{T}, \Pi)$ from the PrTN in Figure 3, where

$$- V = \{v_{p_1}, \dots, v_{p_9}, c_{t_1}, \dots, c_{t_9}, T\}$$

- $\Theta = ((v_{p_1} = 1) \wedge_{x \neq v_{p_1}} (x = 0))$
- $T = \{t_1, t_2, \dots, t_9\}$
- $\Pi = \bigwedge_{t \in Tr} (c_t \leq u_t)$

The CTS in STeP script is as follows.

```

Clocked Transition System % A CTS for the SMIL document
out p:array [1..9] of int
  where (p[1]=1) /\ Forall i:[2..9]. (p[i]=0)
clock c1,c2,c3,c4,c5,c6,c7,c8,c9
  where c1=0 /\ c2 =0 /\ c3=0 /\ c4 =0 /\ c5=0 /\ c6=0
        /\ c7=0 /\ c8 =0 /\ c9=0
Progress
  (p[2]=1 --> c2 <= 5) /\ (p[4]=1 --> c4 <= 20)
  /\ (p[5]=1 --> c6 <= 3) /\ (p[7]=1 --> c7 <= 10)
Transition t1: enable p[1]=1
  assign (p[1],p[2],p[7],c2,c7) := (0,1,1,0,0)
Transition t2: enable p[2]=1 /\c2>=5
  assign (p[2],p[3]) := (0,1)
Transition t3: enable p[3]=1
  assign (p[3],p[4],c4) := (0,1,0)
Transition t4: enable p[4]=1 /\c4>=20
  assign (p[4],p[6]) := (0,1)
Transition t5: enable p[3]=1
  assign (p[3],p[5],c6) := (0,1,0)
Transition t6: enable p[5]=1 /\c6>=3
  assign (p[5],p[6]) := (0,1)
Transition t7: enable p[7]=1 /\c7>=10
  assign (p[7],p[8]) := (0,1)
Transition t8: p[6]=1 /\ p[7]=1
  assign (p[6],p[7],p[9]) := (0,0,1)
Transition t9: enable p[6]=1 /\ p[8]=1
  assign (p[6],p[8],p[9]) := (0,0,1)

```

One real-time requirement of the SMIL document is consistency between the two media sequences, i.e.,

Once the two media sequences start to play, they will coincide within 25 time units.

This property can be formally expressed by the following LTL formula:

$$[] (T \geq 25 \rightarrow p[9]=1).$$

This property is inductive, that is, it is preserved by all transitions. It is proved automatically using rule B-INV and automatic simplification procedures, which is shown in Figure 4.

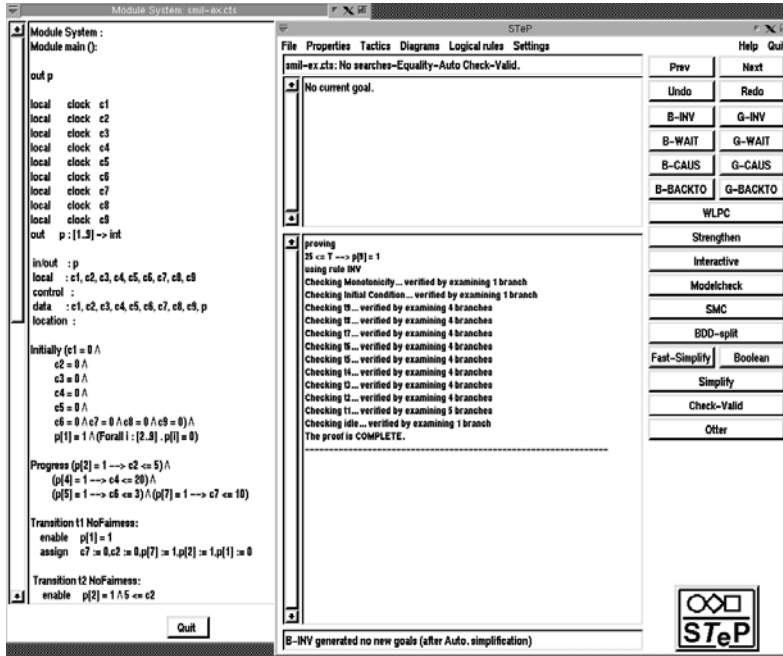


Figure 4. A snapshot of the verification session

5 Conclusion

This paper proposes a formal method for modeling and analyzing real-time systems with SAM. PrTN and LTL are used as the theoretical basis for SAM. Behaviors of real-time systems are modeled by PrTNs, while their properties are specified by LTL. PrTNs have expressive power to model real-time behaviors and complex data structures. By using explicit clock variables in PrTNs, ordinary temporal logic like LTL is capable of specifying real-time properties, which eliminates the need of other specification languages. PrTNs are systematically translated into CTSS. Consequently, we can resort to STeP tools for automating the analysis process.

Our method differs from that in [SC00] for analysis of interactive multimedia documents. In [SC00], interactive multimedia documents are translated into RT-LOTOS specifications [CSLO00], and reachability analysis is performed on the transformed specifications. Instead, we use high-level Petri nets and first-order temporal logic as our formal foundations, and base our method on STeP, where deductive proof or model checking or the combination allow for the verification of a broader class of systems.

Interesting research topics include compositional verification methods and tools for real-time systems with SAM, application of SAM to modeling and design of industrial scale real-time systems.

References

- [BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using Time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.
- [CR83] J.E. Coolahan, Jr. and N. Roussopoulos. Timing requirements for time-driven systems using augmented Petri nets. *IEEE Transactions on Software Engineering*, 9(5):603–616, 1983.
- [CSLO00] J.P. Courtiat, C.A.S. Santos, C. Lohr, and B. Outtaj. Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. *Computer Communications*, 23(12):1104–1123, 2000.
- [CW96] E.M. Clarke and J.M. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.
- [Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 16. Elsevier Science Publisher B.V., 1990.
- [HD02] X. He and Y. Deng. A framework for developing and analyzing software architecture specifications in SAM. *The Computer Journal*, 45(1):111–128, 2002.
- [LS87] N.G. Leveson and J.L. Stolzy. Safety analysis using Petri nets. *IEEE Transactions on Software Engineering*, 13(3):386–397, 1987.
- [MAB⁺94] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. de Alaro, H. Devarajan, H. Sipma, and M. Uribe. STeP: the Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Department of Computer Science, Stanford University, June 1994.
- [MF76] P. Merlin and D.J. Faber. Recoverability of communication protocols. *IEEE Transactions on Communications*, COM-24(9):1036–1043, 1976.
- [MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
- [MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.
- [MP96] Z. Manna and A. Pnueli. Clocked transition systems. Technical Report STAN-CS-TR-96-1566, Computer Science Department, Stanford University, 1996.
- [Mur89] T. Murata. Petri nets properties, analysis and applications. *Proceedings of the IEEE*, 77(4):27–60, 1989.
- [Pnu77] A. Pnueli. The temporal logic of programs. In *18th Ann. IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.
- [Rec98] W3C Recommendation. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*. <http://www.w3.org/TR/REC-smil>, 1998.
- [SC00] P.N.M. Sampaio and J.P. Courtiat. A formal approach for the presentation of interactive multimedia documents. In *Proceedings Proceedings of the 7th ACM International Conference on Multimedia (Part 1)*, pages 435–438, 2000.
- [Tro95] M. Trompedeller. *A Classification of Petri Nets*. <http://www.daimi.dk/PetriNets/classification>, 1995.
- [WHD99] J. Wang, X. He, and Y. Deng. Introducing software architecture specification and analysis in SAM through an example. *Information and Software Technology*, 41:451–467, 1999.