Research Article

A Formal Specification Framework for Designing and Verifying Reliable and Dependable Software for CNC Systems

Yunan Cao

The School of Mechanical, Electronic and Control Engineering, Beijing Jiaotong University, Beijing 100044, China

Correspondence should be addressed to Yunan Cao; ync@bjtu.edu.cn

Received 21 February 2014; Accepted 11 April 2014; Published 22 June 2014

Academic Editor: Yong Tao

Copyright © 2014 Yunan Cao. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As a distributed computing system, a CNC system needs to be operated reliably, dependably, and safely. How to design reliable and dependable software and perform effective verification for CNC systems becomes an important research problem. In this paper, we propose a new modeling method called TTM/ATRTTL (timed transition models/all-time real-time temporal logics) for specifying CNC systems. TTM/ATRTTL provides full supports for specifying hard real time and feedback that are needed for modeling CNC systems. We also propose a verification framework with verification rules and theorems and implement it with STeP and SF2STeP. The proposed verification framework can check reliability, dependability, and safety of systems specified by our TTM/ATRTTL method. We apply our modeling and verification techniques on an open architecture CNC (OAC) system and conduct comprehensive studies on modeling and verifying a system controller that is the key part of OAC. The results show that our method can effectively model and verify CNC systems and generate CNC software that can satisfy system requirements in reliability, dependability, and safety.

1. Introduction

Reliability, dependability, and safety are the three most important factors in software design for computerized numerical control (CNC) systems. As a distributed computing system, a CNC system consists of a computer controller, a set of microcontrollers, and cutting tools, in which the controller can execute machine tool program and generate instructions for microcontrollers to drive the cutting tools to fabricate workpieces by selectively removing materials. Software design is extremely important to make CNC systems operate reliably, dependably, and safely, which directly influences manufacturing quality of workpieces and even the safety of operators. Therefore, how to design reliable and dependable software and perform effective verification for CNC systems becomes an important research problem.

To design such safety-critical systems, testing is not sufficient to guarantee the reliability, dependability, safety, and correctness of CNC software systems. Integrating formal specification and verification methods into software design cycle is an essential solution for quality control in CNC systems [1–3]. Hard real time and feedback, two special properties of applications in CNC systems, need to be considered when formal methods are designed. In terms of hard real time, as CNC systems are used to control machine cutting tools, almost every operation of a CNC system must be finished within a given time constraint. In terms of feedback, in order to guide cutting tools to perform directed interpolation, a lot of results generated previously are needed for generating current result. In this paper, we focus on developing a formal specification and verification frameworks by integrating hard real time and feedback with reliability, dependability, and safety for designing and verifying reliable and dependable software for CNC systems.

A lot of methods have been proposed in modeling formalisms such as finite state machines [4], statecharts [5], message sequence charts [6], and Petri nets [7]. However, the above methods cannot model real-time property efficiently. To model systems with temporal behaviors, temporal logics are the best choice which can specify system behaviors with logical formulas including temporal constraints, events, and relationships between the two [8–10]. Many different forms
of temporal logics have been proposed such as PTL (propositional temporal logic) [11–15], BTTL (branching time temporal logic) [16], ITL (interval temporal logic) [17], CTL (computational tree logic) [18], RTL (real-time logic) [19, 20], LRTL (linear real-time Logic) [1–3], and RTTL (real-time temporal logic) [21]. Among them, LRTL and RTTL are most suitable to specify and verify hard real-time systems.

RTL, which is based on a first-order logic, was introduced in [19] to capture the timing requirements of real-time systems. It provides a uniform way for the specification of relative and uniform timing of events. However, the satisfiability problem for RTL, as well as for other first-order logics, is undecidable. LRTL is the extension form of RTL and can express any linear timing constraint with an arbitrary number of events variables. The way that LRTL uses to overcome the limitation of expressing the relationship between three or more events is discarding the graph form instead from a new matrices form [1], while RTTL, which is also derived from RTL, still uses the graph form, which is called the reachability graph, to express the dependencies in multievents condition [21]. However, RTTL is not good at specifying systems with feedback because there is no specification to express the past [22]. Therefore, we need new modeling formalisms to model and verify CNC systems.

In this paper, we propose a new modeling method called TTM/ATRTTL (extended timed transition models/all-time real-time temporal logics) for specifying CNC systems based on TTM/RTTL in [21, 23–30]. TTM/ATRTTL provides full support for specifying hard real time and feedback that are needed for modeling CNC systems. We also propose a verification framework with verification rules and theorems and implement it with STrP [14, 15] and SF2STrP [31, 32]. The proposed verification framework can check reliability, dependability, and safety of systems specified by our TTM/ATRTTL method. We apply our modeling and verification techniques on an open architecture CNC (OAC) system [33] and conduct comprehensive studies on modeling and verifying a system controller that is the key part of OAC. The results show that our method can effectively model and verify CNC systems and generate CNC software that can satisfy system requirements in reliability, dependability, and safety.

The rest of the paper is organized as follows. In Section 2, we propose our specification method. The formal verification framework is proposed in Section 3. In Section 4, we apply our specification and verification techniques on an open architecture CNC system. The conclusion and the future work are given in Section 5.

2. TTM/ATRTTL: A Formal Specification Method

In this section, we propose our formal specification method, TTM/ATRTTL, that can handle both hard real time and feedback. We first give background knowledge for TTM that is used to describe reactive/real-time systems [12]. Then we propose a new declarative specification language called ATRTTL that is used to describe requirements that a TTM should satisfy.

2.1. Timed Transition Models (TTMs). TTM is a state transition system that can be used to represent a variety of reactive, concurrent, distributed, and real-time systems. It is an extension of the fair transition systems [12] by adding time metric. The time metric with lower and upper time bounds is introduced into transitions. So it can model hard real-time behaviors of systems. Many kinds of modeling methods can be directly translated into TTM such as real-time programs, nondeterministic timed Petri nets, and other constructs with timing properties.

Definition 1. A TTM M is defined as a seven-tuple \( M = (V, R, S, I, T, J, C) \), in which \( V \) is the variable set used to describe processes of \( M \), \( R \) is the union of all variable types of \( V \), \( S \) is the set of states of \( M \) and every state \( s \) in \( S \) is a mapping between variable \( v \in V \) and a corresponding value \( s(v) \), and \( I \) is a Boolean expression of system variables to represent the initial condition of \( M \), \( T \) is the set of transitions, \( J \) is the weakly fair transitions set, and \( C \) is the strongly fair transitions set.

In variable set \( V \), there are four different kinds of variables: activity variables, data variables, clock variables, and next-transition variables. Activity variable is used to represent current status of processes. Data variable can express any linear timing constraint with an arbitrary number of events variables: activity variables, data variables, clock variables, and next-transition variables. Activity variable is used to represent current status of processes. Data variable can be integers, rationales, sets, or other forms of data. Clock variable is a nonnegative integer used to represent global time. Next-transition variable refers to events or transitions.

Definition 2. Transition set \( T \) is the set of all transitions of \( M \) and is defined as a four-tuple \( \tau = (e_\tau, f_\tau, l_\tau, u_\tau) \).

Here, \( e_\tau \) is the enable condition of transition \( \tau \). It is often expressed as \( \text{enb}(\tau) \). It is a Boolean formula defined over transition variables of \( V \). A transition \( \tau \) can only be taken when the value of \( \text{enb}(\tau) \) is true.

\( f_\tau \) is the transformation function of transition \( \tau \). It is often expressed as \( \text{upd}(\tau) \). It is a partial function to cause state transition when \( s(e_\tau) = \text{true} \) is satisfied.

\( l_\tau \) and \( u_\tau \) are constants representing the lower and upper time bounds of transition \( \tau \). They represent that transition \( \tau \) can only be taken during the interval starting from \( l_\tau \) to \( u_\tau \).

There are three particular transitions:

1. **tick** transition: \( \text{tick} = (\text{true}, [t : t + 1], -,-,-) \);
2. **initial** transition: \( \text{initial} = (\text{true}, [], 0,0) \);
3. **spontaneous** transition: \( \text{spont} = (e_\tau, f_\tau, 0,\infty) \).

If there is a transition \( \tau \) from state \( s \) to state \( s' \), then \( s \) is called the prestate of \( \tau \) and \( s' \) is called the poststate of \( \tau \), denoted as \( s' \in \tau(s) \).

Definition 3. Transition relation is a first-order formula \( \rho_\tau(V,V') \) which relates the prestate and poststate of a transition \( \tau \). The formula is denoted as follows:

\[
\rho_\tau(V,V') = \text{enb}(\tau) \land \text{upd}(\tau).
\]
Definition 4. State-assignment is the restriction of states to the domain \((V - \{n\})\) in which \(V\) is the variable set and \(n\) is next-transition variable set. The state-assignment set is denoted as \(S_n\). State-map is the restriction of state \(s\) to the domain \((V - \{t, n\})\) in which \(t\) is time variable set. The state-map set is denoted as \(S_{st}^n\).

Definition 5. A trajectory \(\sigma\) is any infinite sequence \(s_0s_1s_2 \ldots\) of states, where \(s_i \in S\) (the set of all states). Let \(q_1\) be the state-assignment corresponding to a state \(s_i\) in trajectory \(\sigma\); then

\[
\sigma = s_0s_1s_2 \ldots = q_0 \xrightarrow{\tau_0} q_1 \xrightarrow{\tau_1} q_2 \xrightarrow{\tau_2} \ldots
\]

(2)

Not all trajectories are actual behaviors of \(M\). Legal trajectories are those which reflect actual behaviors of a system.

Definition 6. If there are two TTM's, \(M_1\) and \(M_2\), let \(M_1 = (V_1, R_1, S_1, I_1, T_1, J_1, C_1)\) and \(M_2 = (V_2, R_2, S_2, I_2, T_2, J_2, C_2)\); then

\[
M = M_1 \parallel M_2
\]

(3)

\[= (V_1 \cup V_2, R_1 \cup R_2, S_1 \cup S_2, I_1 \wedge I_2, T_1 \parallel T_2, J_1 \parallel J_2, C_1 \parallel C_2)\]

and \(M\) is still a TTM, in which one calls \(M_1\) and \(M_2\) as a pair of parallel TTM's.

2.2. All-Time Real-Time Temporal Logic (ATRTTL). RTTL is used to describe requirements and operations of systems modeled by TTM. In RTTL, the temporal structure is linear and discrete. Time is defined with both a sequence of states and a sequence of temporal instants. The state-based model makes RTTL particularly suitable for model-checking techniques. A natural number is associated with each time instant, so RTTL is based on an explicit model of time. The clock in TTM/RTTL is periodically incremented and is accessible to define real-time formulas. However, there is no representation for past time in RTTL. In CNC systems, a lot of decisions are made based on the past; therefore, we propose a new temporal logic method called ATRTTL (all-time RTTL) by adding new formulas to represent past time.

To define new formulas for ATRTTL, the following operators in linear time temporal logic are used including \(\langle\\rangle\) (next), \(\lozenge\) (henceforth), \(\Diamond\) (eventually), \(\Diamond\) (until), \(\Box\) (preceding), and \(\Diamond\) (waiting-for). Using these temporal operators, RTTL can give succinct, abstract formulas which support the expression of not only qualitative properties such as safety, deadlock avoidance, mutual exclusion, and liveness but also quantitative time bounds on events. Based on these operators, combining future and past temporal logics, ATRTTL (all-time real-time temporal logic), is shown in Tables 1 and 2. We call the formulas in ATRTTL as ATRTTF (all-time real-time temporal formula) that can be used to define reliability, dependability, liveness, and safety.

<table>
<thead>
<tr>
<th>ATRTTF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\sigma, j) = \neg p)</td>
<td>((\sigma, j) \neq p)</td>
</tr>
<tr>
<td>((\sigma, j) = p \lor q)</td>
<td>((\sigma, j) \lor q)</td>
</tr>
<tr>
<td>((\sigma, j) = 0_{\langle})</td>
<td>((\sigma, j + 1) = p)</td>
</tr>
<tr>
<td>((\sigma, j) = \langle)</td>
<td>(\langle p\rangle)</td>
</tr>
</tbody>
</table>

For some \(k, j \leq k, (\sigma, k) = p\) and \(q\)

\[\Box_{\langle, \rangle} p\]

true \((\Box_{\langle, \rangle} p)\) (eventually)

\[\Box_\langle p\rangle\]

\(\neg p\)

\(\forall (\forall\langle\) (entailment)

\[\Box_\langle q\rangle\]

\(\Box_\langle p \leftrightarrow q\rangle\) (congruence)

\[\Box_\langle p \lor q\rangle\]

\(\Box_\langle p \lor q\rangle\) (waiting-for)

\[\Box_\langle \neg p\rangle\]

\(\neg (\neg p)\) (back-to)

Table 1: Satisfaction expressions of ATRTTL.

| Table 2: Past operator expressions of ATRTTL. |
|---|---|
| ATRTTF | Meaning |
| \((\sigma, j) = \Box_{\langle, \rangle} p\) | \(j \geq 1\) and \((\sigma, j - 1) = p\) |
| \((\sigma, j) = pS_{\langle, \rangle} q\) | \((\sigma, j - 1) = p\) |
| \((\sigma, j) = \Box_{\langle, \rangle} p\) | \(\forall (\forall\langle\) (sometime in the past)

The "satisfaction" in Tables 1 and 2 is defined as follows.

Definition 7. For any trajectory \(\sigma = s_0s_1s_2 \ldots\), let \(p\) and \(q\) represent the given ATRTTFs, and let \((\sigma, j) = p\) represent that \(p\) holds at position \(j\) in \(\sigma\); the satisfaction is given in Tables 1 and 2.

The concept of past operator expressions of ATRTTL is corresponding to the concept of normal expressions of RTTL. We introduce the past time factor, so that the whole time axis can be depicted.

3. A Formal Verification Framework with TTM/ATRTTL

In this section, we propose a formal verification framework to verify systems modeled by TTM/ATRTTL. We first propose a set of basic verification rules which can be used to simplify temporal formulas. Then we derive a set of theorems based on the verification rules which can make verification simpler.

3.1. Formal Verification Rules. First, we define verification condition as follows.

Definition 8. Verification condition is \((\varphi)\) \(\tau(\psi)\), which means that whenever \(\tau\) is taken from a state satisfying \(\varphi\), the resulting
state must satisfy \( \psi \). If \( \varphi \) itself is the only resulting state, then the verification condition will be given by \( \{\varphi\} \tau \{\varphi\} \).

Based on Definition 8, we give a set of verification rules which are the basis of the theorems in Section 3.2. The most important feature of verification rules is that it can convert a given temporal logic formula to simple first-order formulas that can be easily verified.

Giving an initial condition \( I \), a transition set \( T \), and ATRTTFs \( \varphi \) and \( \psi \), the verification rules are as follows.

**Rule 1 (INV). INV for ATRTTF \( \varphi \):**

\[
\begin{align*}
I1: & \quad I \rightarrow \varphi \\
I2: & \quad \{\varphi\} T \{\varphi\} \\
\Downarrow & \quad \varphi
\end{align*}
\]  

Premise \( I1 \) means that the initial condition implies formula \( \varphi \). Premise \( I2 \) means the verification condition \( \{\varphi\} \tau \{\varphi\} \) holds. The conclusion is that formula \( \varphi \) is henceforth satisfied.

**Rule 2 (MON). MON for ATRTTF \( \varphi \) and \( \psi \):**

\[
\Downarrow \varphi, \varphi \rightarrow \psi \quad \Downarrow \varphi, \psi
\]  

Rule MON means if we can prove the henceforth property of formula \( \varphi \) by INV and we can imply the \( \psi \) from \( \varphi \), then the conclusion is that formula \( \psi \) is henceforth valid. By applying Rule 2, we can convert a complicated assertion to a simpler one that can be proved first.

**Rule 3 (CON). CON for ATRTTF \( \varphi \) and \( \psi \):**

\[
\Downarrow \varphi, \varphi \psi \quad \Downarrow \varphi, (\varphi \land \psi)
\]  

Rule CON means that, by proving the henceforth property of relative simple ATRTTFs, we can prove a more complicated one.

**Rule 4 (INC). INC For ATRTTF \( \varphi \), \( \psi \), and \( \chi \):**

\[
\Downarrow \chi, \varphi \rightarrow \psi \quad \Downarrow \varphi, (\varphi \land \psi)
\]  

The INC rule means if we can prove the henceforth validity of ATRTTF \( \chi \), and an augmented implication \( \varphi \land (\varphi \rightarrow \psi) \), then we can get the henceforth property of implication \( \varphi \rightarrow \psi \).

**Rule 5 (GEN). GEN for ATRTTF \( \varphi \), \( \chi \), and \( \psi_1, \ldots, \psi_k \):**

\[
\begin{align*}
G0: & \quad \Downarrow \psi_1, \ldots, \psi_k \\
G1: & \quad \Downarrow \left( \bigwedge_{i=1}^{k} \psi_i \right) \land \chi \rightarrow \varphi \\
G2: & \quad I \rightarrow \chi \\
G3: & \quad \Downarrow \left( \bigwedge_{i=1}^{k} \psi_i \right) \land \chi \quad \Downarrow \varphi
\end{align*}
\]  

The GEN rule is a generalized form of RULE INC. It means, if we can prove a series of ATRTTF \( \psi_1, \ldots, \psi_k \) and the augmented implication \( \left( \bigwedge_{i=1}^{k} \psi_i \right) \land \chi \rightarrow \varphi \), the initial condition implies formula \( \chi \), and the verification condition \( \left( \bigwedge_{i=1}^{k} \psi_i \right) \tau \chi \) holds; then we can get the validity of henceforth property of ATRTTF \( \varphi \).

**Rule 6 (WAIT). WAIT for ATRTTF \( \varphi, \psi \):**

\[
\begin{align*}
W1: & \quad I \rightarrow \varphi \lor \psi \\
W2: & \quad \{\varphi\} \tau \{\varphi \lor \psi\}
\end{align*}
\]  

The WAIT rule means if any state satisfying the initial condition implies ATRTTFs \( \varphi \) and \( \psi \) and the verification condition \( \{\varphi\} \tau \{\varphi \lor \psi\} \) is also satisfied, then we can get the waiting-for formula \( \varphi \lor \psi \).

**Rule 7 (NEST-WAIT). NEST for ATRTTF \( \varphi_n, \ldots, \varphi_0 \):**

\[
\begin{align*}
W1: & \quad I \rightarrow \bigvee_{j=0}^{n} \varphi_j \\
W2: & \quad \{\varphi_i\} T \left\{ \bigvee_{j=0}^{n} \varphi_j \right\} \quad \text{for } i = 1, \ldots, n
\end{align*}
\]  

The NEST-WAIT rule is a generalized form of RULE WAIT. It means if any state satisfying the initial condition implies a series of \( \varphi_i \) and every \( \varphi_i \) is followed by a \( \varphi_j \) \((i \geq j)\), then we can get the nested waiting-for ATRTTF \( \varphi_n \lor \varphi_{n-1} \cdots \lor \varphi_1 \lor \varphi_0 \).

**Rule 8 (STRONG-NEST-WAIT). STRONG for ATRTTF \( \varphi_n, \ldots, \varphi_0 \):**

\[
\begin{align*}
S1: & \quad I \rightarrow \bigvee_{j=0}^{n} \varphi_j \\
S2: & \quad \{\varphi_i\} T \left\{ \bigvee_{j=0}^{n} \varphi_j \right\} \quad \text{for } i = 1, \ldots, n \\
S3: & \quad \varphi_i \rightarrow \psi_j \quad \text{for } i = 0, \ldots, n
\end{align*}
\]  

The STRONG-NEST-WAIT rule is a more generalized form of RULE NEST-WAIT. It adds the assertion \( S3 \) to NEST-WAIT in such a way that we can get a complicated assertion waiting-for ATRTTF \( \varphi_n \lor \varphi_{n-1} \cdots \lor \varphi_1 \lor \varphi_0 \) by a relative simple one \( \varphi_n \lor \varphi_{n-1} \cdots \lor \varphi_1 \lor \varphi_0 \).

**Rule 9 (CHAIN). CHAIN for ATRTTF \( \varphi \):**

\[
\begin{align*}
C1: & \quad I \rightarrow \varphi \\
C2: & \quad \{\varphi\} T \{\varphi\}
\end{align*}
\]  

The CHAIN rule means if we can prove \( \varphi \) from the initial condition and the verification condition \( \{\varphi\} T \{\varphi\} \) is valid, then we can get the eventuality ATRTTF \( \diamond \varphi \).
Rule 10 (NEST-CHAIN). NEST-CHAIN for ATRTTF $\psi_n, \ldots, \psi_1$ and transitions $\tau_n, \ldots, \tau_1$:

\[
C1: I \rightarrow q \vee \bigvee_{j=1}^{n} \psi_j \\
C2: \{\psi_i\} T\left\{ q \vee \bigvee_{j=1}^{n} \psi_j \right\} \text{ for } i = 1, \ldots, n \\
C3: \{\psi_i\} \tau_i \left\{ q \vee \bigvee_{j=1}^{n} \psi_j \right\} \text{ for } i = 1, \ldots, n \\
C4: \psi_i \rightarrow \text{Enb}(\tau_i) \text{ for } i = 1, \ldots, n \\
\phi q 
\]

The NEST-CHAIN ATRTTF is a more generalized form of RULE CHAIN. Premise C1 means initial condition $I$ implies $q$ or one of the ATRTTFs $\psi_i$. C2 means the verification condition $[\psi_i] T\{ q \vee \bigvee_{j=1}^{n} \psi_j \}$ is valid for every $\psi_i$. C3 means that the verification condition satisfies the “just” property. And C4 means that, at each $\psi_i$, the just transition $\tau_i$ is enabled. With these four premises, we can get the more generalized eventuality ATRTTF $\phi q$.

3.2. Theorems Based on Verification Rules. Based on the verification rules above, we derive a set of theorems that can simplify complicated systems or specifications for easy verification based on theorems in [31]. We first define closed-loop TTM and then present four theorems.

Definition 9. A closed-loop TTM is a three-tuple $CT = (O, S, C)$, in which $O$ represents the TTM for the controlled object of a system, $S$ represents the required specifications of a system in ATRTL, and $C$ represents the TTM of the system controller. $CT$ means that $O$ can work under $C$ in the manner of what $S$ requires.

Based on the above definition, we give two $CT$s. One is simple and the other is complicated. The set of theorems below are used to construct a complicated $CT$ based on a simple $CT$ with easy transferring procedures. The theorems are separated into three categories: invariance set, eventuality set, and sequence set. We first present a premise that is used for all theorems.

Premise. Suppose we have two closed-loop TTMs $CT_1 = (O_1, S_1, C_1)$ and $CT_2 = (O_2, S_2, C_2)$ where $CT_1$ and $CT_2$ have underlying TTM representations $M_1 = [O_1 \parallel C_1] = (V_1, R_1, S_1, I_1, T_1, C_1)$ and $M_2 = [O_2 \parallel C_2] = (V_2, R_2, S_2, I_2, T_2, C_2)$. Let $\bar{T} = T_2 - T_1$, just transition, $\bar{\tau} = \tau_2 - \tau_1$, $V_1 \subseteq V_2$, $R_1 \equiv R_2$, $S_1 \subseteq S_2$, $I_2 \rightarrow I_1$, $T_1 \subseteq T_2$, $I_1 \subseteq I_2$, and $C_1 \subseteq C_2$.

Theorem 10 (invariance 1). $CT_2$ satisfies ATRTTF $M_2 \models \phi \varphi$ if $CT_1$ satisfies $M_1 \models \phi \varphi$ and the proof obligations $\Delta_1$ and $\Delta_2$ hold, where

$$\Delta_1: \mathcal{O} ((\neg I_2 \land I_1 \land \neg \varphi) \lor (I_2 \land \neg I_1 \land \varphi))$$

$$\Delta_2: \left\{ \left( \bigwedge_{i=1}^{k} \psi_i \right) \land \varphi \right\} \bar{T} \{ \chi \},$$

that is, $\rho_{\bar{\tau}} \land \left( \bigwedge_{i=1}^{k} \psi_i \right) \land \chi \rightarrow \chi$, $\forall \bar{\tau} \in \bar{T}$,

in which $\psi_i$ and $\chi$ are defined in GEN rule.

Proof. Because $M_1 \models \phi \varphi$, from GEN rule, we can get

A0: $\mathcal{O} \left\{ \bigwedge_{i=1}^{k} \psi_i \land \chi \rightarrow \varphi \right\}$

A1: $\mathcal{O} \left\{ \bigwedge_{i=1}^{k} \psi_i \land \chi \rightarrow \varphi \right\}$

A2: $I_1 \rightarrow \chi$

A3: $\left\{ \left( \bigwedge_{i=1}^{k} \psi_i \right) \land \chi \right\} T_1 \{ \chi \}.$

In order to prove $M_2 \models \phi \varphi$, we should establish four condition expressions as given by GEN rule. Consider the following:

B0: $\mathcal{O} \psi_1, \ldots, \psi_k$

B1: $\mathcal{O} \left\{ \bigwedge_{i=1}^{k} \psi_i \land \chi \rightarrow \varphi \right\}$

B2: $I_2 \rightarrow \chi$

B3: $\left\{ \left( \bigwedge_{i=1}^{k} \psi_i \right) \land \chi \right\} T_2 \{ \chi \}.$

Obviously, A0 $\rightarrow$ B0, A1 $\rightarrow$ B1. Then we focus on B2 and B3. From $\Delta_1$ and $\Delta_2$, we can get

$\Delta_2 \land \Delta_1 = ((\neg I_2 \lor \varphi) \land (\neg I_2 \land I_1 \land \neg \varphi) \lor (I_2 \land \neg I_1 \land \varphi))$ \hspace{1cm} (17)

$\rightarrow (I_2 \rightarrow \chi) \leftrightarrow B2$

and $\Delta_2 \land A3 \leftrightarrow B3$. B0, B1, B2, and B3 have all been proved. Hence the invariance theorem is proved. \qed

Using Theorem 10, when proving an invariance of a more complicated system, we only need to deal with the additional initial states and the transitions rather than the whole incremental system.

Theorem 11 (invariance 2). CT2 satisfies ATRTTF $M_2 \models \phi \varphi$ if $CT_1$ satisfies $M_1 \models \phi \varphi$, and the proof obligations $\Delta_1$, $\Delta_2$, $\Delta_3$, and $\Delta 4$ hold, where

$$\Delta_1: \mathcal{O} ((\neg I_2 \land I_1 \land \neg \varphi) \lor (I_2 \land \neg I_1 \land \varphi))$$

$$\Delta_2: \left\{ \psi_1 \right\} \bar{T} \{ \varphi \}, \text{ that is, } \rho_{\bar{\tau}} \land \psi_1 \rightarrow \varphi, \forall \bar{\tau} \in \bar{T}$$

$$\Delta_3: I_2 \rightarrow \varphi_2$$

$$\Delta_4: \left\{ \left( \varphi_1 \land \varphi_2 \right) \right\} T_2 \{ \varphi_2 \}, \text{ that is, } \rho_{\bar{\tau}_2} \land \left( \varphi_1 \land \varphi_2 \right) \rightarrow \varphi_2, \forall \bar{\tau}_2 \in T_2.$$
Theorem 11 means when a new ATRTTF is added into the specifications, one only needs to prove \( \rho_2 \land (\varphi_1 \land \varphi_2) \rightarrow \varphi_2 \) rather than the whole transition set of a system.

**Theorem 12** (eventuality). \( CT_2 \) satisfies the eventuality ATRTTF \( M_2 \models \varnothing q \) if \( CT_1 \) satisfies \( M_1 \models \varnothing q \) and the proof obligations \( \Delta_1, \Delta_2, \Delta_3 \), and \( \Delta_4 \) hold, where

\[
\Delta_1: \emptyset \left( \neg I_2 \land I_1 \land \left( q \lor \bigvee_{j=1}^{n} \varphi_j \right) \right)
\]

\[
\lor \left( I_2 \land \neg I_1 \land \left( q \lor \bigvee_{j=1}^{n} \varphi_j \right) \right)
\]

\[
\Delta_2: \{ \varphi_i \} \overline{T} \left\{ q \lor \bigvee_{j=1}^{n} \varphi_j \right\}, \quad \text{that is, } \rho_2 \land \varphi_1 \rightarrow q \lor \bigvee_{j=1}^{n} \varphi_j,
\]

\[
i = 1, \ldots, n, \forall \overline{r} \in \overline{T}
\]

\[
\Delta_3: \{ \varphi_i \} \overline{r}_1 \left\{ q \lor \bigvee_{j=1}^{n} \varphi_j \right\}, \quad i = 1, \ldots, n
\]

\[
\Delta_4: \varphi_i \rightarrow \text{Enb}(\overline{r}_1), \quad i = 1, \ldots, n.
\]

**Proof.** Because \( M_1 \models \varnothing q \), from NEST-CHAIN rule, we can get

\[
A_1: I_1 \rightarrow q \lor \bigvee_{j=1}^{n} \varphi_j
\]

\[
A_2: \{ \varphi_i \} T_1 \left\{ q \lor \bigvee_{j=1}^{n} \varphi_j \right\}, \quad \text{for } i = 1, \ldots, n
\]

\[
A_3: \{ \varphi_i \} \tau_{1,j} \left\{ q \lor \bigvee_{j=1}^{n} \varphi_j \right\}, \quad \text{for } i = 1, \ldots, n
\]

\[
A_4: \varphi_i \rightarrow \text{Enb}(\tau_{1,j}), \quad \text{for } i = 1, \ldots, n.
\]

In order to prove \( M_2 \models \varnothing q \), we should establish four condition expressions as also given by NEST-CHAIN rule. Consider the following:

\[
B_1: I_2 \rightarrow q \lor \bigvee_{j=1}^{n} \varphi_j
\]

\[
B_2: \{ \varphi_i \} T_2 \left\{ q \lor \bigvee_{j=1}^{n} \varphi_j \right\}, \quad \text{for } i = 1, \ldots, n
\]

\[
B_3: \{ \varphi_i \} \tau_{2,j} \left\{ q \lor \bigvee_{j=1}^{n} \varphi_j \right\}, \quad \text{for } i = 1, \ldots, n
\]

\[
B_4: \varphi_i \rightarrow \text{Enb}(\tau_{2,j}), \quad \text{for } i = 1, \ldots, n.
\]

From \( \Delta_1, \Delta_2, \Delta_3, \) and \( \Delta_4 \), we can get

\[
\begin{align*}
A_1 \land A_1 & = \left( \left( \neg I_1 \lor \bigvee_{j=1}^{n} \varphi_j \right) \right) \land \left( \neg I_2 \lor \bigvee_{j=1}^{n} \varphi_j \right) \\
& \lor \left( \neg I_2 \lor \bigvee_{j=1}^{n} \varphi_j \right)
\end{align*}
\]

(22)

and \( \Delta_2 \land A_2 \land B_2, \Delta_3 \land A_3 \land B_3, \) and \( \Delta_4 \land A_4 \land B_4 \) have all been proved. Hence the eventuality theorem is proved. \( \square \)

Using the eventuality theorem, when proving an eventuality of a complicated system, we only need to deal with the additional initial states, transition set, and the just transition set rather than the whole incremental system.

**Theorem 13** (sequence). \( CT_2 \) satisfies the sequence ATRTTF \( M_2 \models \psi_n \psi_{n-1} \cdots \psi_1 \psi_0 \) if \( CT_1 \) satisfies \( M_1 \models \psi_n \psi_{n-1} \cdots \psi_1 \psi_0 \) and the proof obligations \( \Delta_1, \Delta_2, \Delta_3, \) and \( \Delta_4 \) hold, where

\[
\Delta_1: \emptyset \left( \neg I_2 \land I_1 \land \left( \bigvee_{j=1}^{n} \varphi_j \right) \right)
\]

\[
\lor \left( I_2 \land \neg I_1 \land \left( \bigvee_{j=1}^{n} \varphi_j \right) \right)
\]

\[
\Delta_2: \{ \varphi_i \} \overline{T} \left\{ \bigvee_{j=1}^{i} \varphi_j \right\}, \quad \text{that is, } \rho_2 \land \varphi_1 \rightarrow \bigvee_{j=0}^{i} \varphi_j,
\]

\[
i = 1, \ldots, n, \forall \overline{r} \in \overline{T}
\]

**Proof.** Because \( M_1 \models \psi_n \psi_{n-1} \cdots \psi_1 \psi_0 \), from STRONG-NEST-WAIT rule, we can get

\[
A_1: I_1 \rightarrow \bigvee_{j=0}^{n} \varphi_j
\]

\[
A_2: \{ \varphi_i \} T_1 \left\{ \bigvee_{j=1}^{i} \varphi_j \right\}, \quad \text{for } i = 1, \ldots, n
\]

\[
A_3: \varphi_i \rightarrow \psi_i, \quad \text{for } i = 0, \ldots, n.
\]

(24)
In order to prove $M_j |= \psi_0 \wedge \psi_{n-1} \cdots \psi_1 \wedge \psi_0$, we should establish four condition expressions as given by STRONG-NEST-WAIT rule. Consider the following:

$$B1: I_2 \xrightarrow{n} \bigvee_{j=0}^{n} \psi_j$$

$$B2: \{\psi_i\} T_2 \left\{ \begin{array}{l} i \psi_j \end{array} \right\} \text{ for } i = 1, \ldots, n$$

$$B3: \psi_i \xrightarrow{i} \psi_i \text{ for } i = 0, \ldots, n.$$  \hspace{1cm} (25)

From $\Delta 1$ and $A1$, we can get

$$A1 \wedge \Delta 1 = \left( \neg I_1 \lor \left( \bigvee_{j=0}^{n} \psi_j \right) \right) \wedge \left( \neg I_2 \wedge I_1 \wedge \neg \left( \bigvee_{j=0}^{n} \psi_j \right) \right)$$

$$\lor \left( I_2 \wedge \neg I_1 \wedge \left( \bigvee_{j=0}^{n} \psi_j \right) \right)$$

$$\leftrightarrow \left( I_2 \xrightarrow{n} \left( \bigvee_{j=0}^{n} \psi_j \right) \right) \leftrightarrow B1$$  \hspace{1cm} (26)

and $\Delta 2 \wedge A2 \leftrightarrow B2$, $A3 \equiv B3$. B1, B2, and B3 have all been proved. Hence the sequence theorem is proved. \hfill \Box

Using the sequence theorem, when proving the sequence ATRTTF for a complicated system, we only need to deal with the additional initial states and transition set rather than the whole incremental system.

4. Case Study: Specification and Verification of an Open Architecture CNC

In this section, we apply our TTM/ATRTTL to specify and verify an open architecture CNC (OAC) system proposed in [33]. First, we present the structure of a typical CNC system (Section 4.1). Second, we introduce the system-level structure of a typical OAC system (Section 4.2). Third, we specify the component library which is the controlled object (Definition 9) of OAC with our TTM/ATRTTL at system level (Section 4.3). Fourth, we give the closed-loop specifications of properties in ATRTTL (Section 4.4). Next, we use the OAC system controller (Definition 9) as an example to show how to design reliable and dependable CNC software system based on our specification and verification method with modular and bottom-up approaches (Section 4.5). Finally we implement our verification method based on two toolsets, STeP and SF2STeP, and give the results (Section 4.6).

4.1. A Typical CNC System. A CNC system is a typical distributed system. The system-level structure of a CNC system is shown in Figure 1. From Figure 1, we can see that a typical CNC system has one system computer controller and several microcontrollers such as axis controller, logic controller, and feedback sensor controller. It is a challenge to design software for such a complicated distributed computing control system. In practice, component-based design methodology with hierarchy is widely used in software design for CNC systems. Therefore, next we will show how to apply our TTM/ATRTTL to specify and verify an open architecture CNC (OAC) system proposed in [33].

4.2. Open Architecture CNC Systems. In this section, we introduce open architecture CNC (OAC) systems. Frist, we present the basic concepts of OAC systems. Then we present the system-level structure of our OAC in detail.

4.2.1. Basic Concepts. CNC system is used to control the lathes working in a right way. A typical CNC system includes many function modules such as decoder, interpolator, and PLC. Since the modules of CNC vary based on the different applications, an open architecture with a modularized and reconfigurable methodology is needed such that the whole development procedure is independent of hardware [33]. Basically, an OAC is an open architecture CNC, which is a component-based system with the following characteristics: interoperability, transplantability, and scalability. In an OAC, a component is a module corresponding to specific functions of the system. An OAC system can be divided into many small function modules and implemented with corresponding components. A component encapsulates and hides the implementation information to outside environments and provides an interface by which users can utilize its functions.

To develop software with component-based technology can improve the reusability and reliability of the software and reduce the cost of development and maintenance. Based on OAC, we can construct a uniform and reconfigurable system platform. With this “open” architecture, we can easily select, integrate, change, and extend the functions of a system with various user requirements. Furthermore, different function modules of a system can be provided by different suppliers.

4.2.2. The Structure of OAC Systems. The structure of a typical OAC is shown in Figure 2.

As shown in Figure 2, an OAC mainly consists of two main parts: a component library and an OAC system controller. The component library contains various components corresponding to different functions of a CNC. The OAC system controller is to configure, schedule, and cooperate components so as to make them work together to satisfy system requirements. To apply our closed-loop TTM model defined in Definition 9 (Section 3.2) on an OAC system, in which we define a closed-loop TTM as $CT = (O, S, C)$, we have the following: the controlled object (“O”) is used to model the TTM of the components of an OAC; the controller (“C”) is used to model the TTM of the OAC system controller of an OAC; the system specification (“S”) is used to model the system requirements in ATRTTL. Thus, after we obtain the TTM of the components and system controller of an OAC, we can perform various verifications based on the system requirements described with ATRTTL.
with correct timing sequences. For example, as shown in
Figure 3, a system controller can schedule the components
and execute them following the given sequence. As the system
controller is the key part of an OAC, we will analyze it in detail
in Section 4.5.

4.3. The Component Library of OAC Systems. In this section,
we specify the component library of an OAC system with
our specification method at system level. In Section 4.3.1, we
use the specification of the decode component as an example
to show how to specify a component in our framework. In
Section 4.3.2, in order to consider the relationship between
different components, we add the environment TTM and
the interface description to each component. In Section 4.3.3,
we discuss how to specify the whole component library
of the OAC system based on the individual component
specification.

4.3.1. Specification of Components with TTM. In order to
model the whole component library of OAC system, the first
step is to model every component in component library. Next,
we use the modeling of the decoder component (DEC) as an
equation to show how to specify a component with TTM. The
TTM model of DEC is shown in Figure 4.

In Figure 4, a rectangle is used to represent a state and
a directed edge between two states is used to represent one
transition. Every transition contains four parts: the name,
Table 3: Components of OAC.

<table>
<thead>
<tr>
<th>Components</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC program</td>
<td>Decoder: decode input instructions of NC-programs and generate Mcode and Gcode.</td>
</tr>
<tr>
<td>DEC</td>
<td></td>
</tr>
<tr>
<td>Gcode data</td>
<td>Cutting tool compensator: transfer the input contour trajectory of a workpiece into the output trajectory of cutting tools.</td>
</tr>
<tr>
<td>CTC</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>Interpolator: based on the input Gcode and Machine structure data, generate position data.</td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Position data</td>
<td>Position controller: in every sampling cycle, based on the input position and detection data, generate control data for machine tools and display data.</td>
</tr>
<tr>
<td>Detection</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Machine tool data</td>
<td>Displayer: display position of cutting tools and information of machine tools.</td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Display data</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Machine tool feedback data</td>
<td>Detector: based on the feedback data from machine tools, generate machine tool status information.</td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Position data</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Machine tool control data</td>
<td>Programmable logic controller: execute the input Mcode Data and PLC I/O to perform corresponding logical control functions and handle emergent events of machine tools.</td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Machine tool control data</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 4: The TTM model of the DEC component.](image-url)
the lower and upper time bounds, the enable condition, and the transition function. For example, in the transition Reset[1|2](reset == true) → [reset_done : false], “Reset” is the transition name, [1|2] denotes the lower and upper time bounds, (reset == true) denotes the enable condition of the transition, and [reset_done : false] is the transition function. We attach “@” inside a state if it will be further specified in the model. For example, for “BUSY@,” we further specify state “BUSY” in Figure 5. This hierarchy structure can help simplify the modeling of a system.

From Figure 4, we can see that the open-loop behavior of DEC is separated into six states which are IDLE, BUSY, MCODE_DECODED, GCODE_DECODED, FAULT, and RESET. State IDLE represents that the decoder component is in a state waiting for a new NC-program segment to be inputted. State BUSY is the abstract description of the decoding procedure. “MDt” and “GDt” are used to represent the decoding time of one Mcode segment and that of Gcode segment, respectively. MCODE_DECODED and GCODE_DECODED are two states to represent that the decoding processes of Mcode and Gcode have been finished, respectively. State FAULT represents that an inputted NC-program segment is illegal. And state RESET represents that we need to reset the decoding component and restore its normal execution. The transitions of the whole decoding component are managed by the system controller.

As state “BUSY” is the most important phase for a decoder component, it is further specified in Figure 5. From Figure 5, we can see that there are six states, PARSE_START, PARSE_SCAN, PARSE_LEXICAL, PARSE_SYNTAX, PARSE_COMPENSATION, and PARSE_FAULT, in state “BUSY”. Among them, the first five states depict the normal procedure of a decoding scenario and the last one represents that there exists error. States PARSE_SCAN, PARSE_LEXICAL, PARSE_SYNTAX, and PARSE_COMPENSATION represent the main workflow of a decoder in “BUSY”; PARSE_SCAN is to separate the number, operator, and address word of one NC-program segment such as “G01 X10 Z10 F100”; PARSE_LEXICAL is to assign the number with the corresponding address word; PARSE_SYNTAX is to interpret the address word with a real specific engineering meaning; and PARSE_COMPENSATION is to change the result from state PARSE_SYNTAX to a format which a cutting tool compensator component (CTC) can recognize.

4.3.2. Constructing the TTM of Component Environment.
Here we still use the DEC component as the example. As in Section 4.3.1, we can construct the TTM of component DEC without considering the interference of other components and the effect of system controller. In order to construct the component library TTM of OAC system, we should consider the relationship between other components in OAC with DEC. Here, we give the concept of environment TTM and add the interface description to TTM component.
Now we make the TTM component consisting of an interface specification and a body. The body is the TTM component which we already mentioned in Section 4.3.1. The purpose of the interface specification is to list all the shared variables or the message passing channels through which the component communicates with the outside. The shared variables in interface specification have three kinds of modes which are IN, OUT, and EXTERNAL. Mode IN means the variable in interface specification can be read in the body, mode OUT means the variable can be written in the body, and mode EXTERNAL means the variable can be used by another component module. The interface specification can be described as a TTM environment. When finishing construction of all the component-environment pair of TTM of every component, the component library TTM of OAC can be constructed. The DEC TTM component with its TTM environment is shown in Figure 6.

From Figure 6, we construct a new TTM component with two parallel composition TTM of DEC and ENV as

\[
DEC_{\text{ENV}} = [\text{DEC} \parallel \text{ENV}].
\]  

(27)

TTM DEC here is looked at as a body and TTM ENV is looked at as the interface specification of the body DEC. With the TTM ENV, we can restrict the model-checking procedure only to one component rather than the whole system. With this approach, we can effectively deal with the state explosion problem in Section 4.7.2. The ENV considers the interface variables through which the DEC component communicates with the outside. In Figure 6, we can see that the TTM ENV of DEC includes four interface variables: read nc finished, output, M mode, and G mode. Among them, read nc finished is declared to be mode of EXTERNAL IN and the other three variables are declared to be mode of EXTERNAL OUT.

4.3.3. The TTM Model of the Component Library of OAC Systems. After obtaining the TTM model of every component of an OAC system, we can construct a component library with the conjunction of the parallel compositions of the TTM of components. From Definition 6, the TTM model of a whole component library can be obtained as follows:

OAC

\[
= [C_1 \parallel \cdots \parallel C_n]
\]

\[
= [\text{DEC}_{\text{ENV}} \parallel \text{PLC}_{\text{ENV}} \parallel \text{CTC}_{\text{ENV}} \parallel \text{INT}_{\text{ENV}} \parallel \text{PCTL}_{\text{ENV}} \parallel \text{SVO}_{\text{ENV}} \parallel \text{DET}_{\text{ENV}} \parallel \text{DSY}_{\text{ENV}}].
\]  

(28)

Note that the TTM model of a component library constructed above is only used to describe function modules of an OAC. To make the components work together to satisfy system requirements, a system controller is needed to configure, schedule, and cooperate components. Next, we will present how to describe the system requirements in ATRTTL in Section 4.4. In Section 4.5, we will show how to design and verify the system controller based on the TTM model of the component library and the system requirements described in ATRTTL.

4.4. The Specifications of System Requirements in ATRTTL. In the above section, we present how to model the component library of an OAC system. In this section, we present how to describe system requirements of an OAC in ATRTTL. The system requirements are the design goals of a system and they are specified by ATRTTL in this paper. After we obtain this set of specification with ATRTTL, we can perform verification to see if a system controller can satisfy the requirements.

In order to design reliable and dependable CNC systems, we specify six sets of system requirements in ATRTTFs as shown in Table 4: Future Liveness, Past Liveness, Safety, Fail-Safe, System-level Real-Time, and Component Real-Time.

In Table 4, “Liveness” is known as the “good” things that should happen, which partially determines the reliability and dependability of a system. For an OAC, it means that the system should work on right time sequences. There are two categories of liveness property which are Future Liveness and Past Liveness so we can verify events in the past and future. For ATRTTF FL, it means when the NC code has finished processing in component DEC, it should then be processed in component CTC or component PLC. For ATRTTF PL, it means when the NC code is now being processed in component SVO, it should just have gotten the data either from component PLC or from component PCTL.

“Safety” is known as the “bad” things that should not happen, which determines the safety of a system. For S, it means the data of one single NC-program segment cannot be processed in CTC component and PLC component simultaneously.

“Fail-Safe” is used to deal with the conditions when mistakes occur, which relates to reliability, dependability, and safety. For FS, it means that when there are some problems in any component, the component cannot restart to work until this component returns to the RESET state.

“Real-Time” means that a certain event (state) must happen in a specified time interval. Real-Time specification is important for CNC systems since it may cause disaster if a real-time event cannot be finished within its deadline. For ST, it means when the OAC system starts to work, it must finish all the functions in the time interval between Min Dec Time + Min Plc Time + Min Svo Time + Max Dec Time + Max Plc Time + Max Svo Time + Max Dsy Time. For CT, it means when the component DEC starts to work, it must be finished in the time interval between Min Dec Time and Max Dec Time.

In Sections 4.5 and 4.6, we will give the analysis and verification results of the above six categories of functional ATRTTL formula.

4.5. Design of OAC System Controller. Based on the results of the library component in Section 4.3 and the system specifications of an OAC in Section 4.4, in this section, we design a system controller so that the library component can work under it in the manner of what the specifications of OAC require. System controller is the most important part in OACs, and it cooperates, synchronizes, and makes all the components work together to satisfy system requirements that represent as the ATRTTFs based on our TTM/ATRTTL.
specification method. In Algorithm 1, we show an integrated procedure to design a system controller of an OAC. Basically, the whole procedure of designing an OAC system controller has three main steps. The first step is to construct the open-loop component library TTM of OAC system, which is already depicted in Section 4.3. The second step is to express the whole OAC system requirement specifications, which is already given in Section 4.4. The third step is to present the sound system controller which we will give in detail in this section.

4.5.1. Constructing the TTM of Component Controller. From Step 8, the system controller $C$ is constructed based on component controller $C_i$ in a hierarchical manner using the bottom-up approach. Using this method, we can construct or analyze a complicated system controller by a relatively simple component controller. So first in Step 7, we should give the component controller of every component TTM. Here, we still use the DEC component as the example seen in Figure 7.

From Figure 7, the controlling way of TTM DECCONTROLLER on DEC_ENV is as follows: by accepting
Procedure for designing a system controller

**Input:** Each components in TTM; Closed-loops Specifications of properties in ATRTTL

**Output:** OAC System Controller

**Step 1.** Give the corresponding TTM $P_i$ of every component $i$ in OAC (Section 4.3.1)

**Step 2.** Give the environment TTM $E_i$ of corresponding $P_i$ which considering all the variables having relationship with other components (Section 4.3.2)

**Step 3.** Combine the pair $P_i$ and $E_i$ to generate a parallel composition TTM $O_i$ (Section 4.3.2)

**Step 4.** Construct the component library TTM of OAC system by equation $P = ∥ O_i$ (Section 4.3.3)

**Step 5.** Give the specification set $S_i$ in ATRTTL which should be satisfied in $O_i$ (Section 4.4)

**Step 6.** Express the whole OAC system requirement specifications $S$ by equation $S = V S_i$ (Section 4.4)

**Step 7.** Present the component controller $C_i$ for $O_i$ which can satisfy the $S_i$ (Section 4.5.1)

**Step 8.** Make an initial guess of system controller $C$ based on component controller $C_i$ (Section 4.5.2)

**Step 9.** for $i = 0$ to $m$ (Section 4.5.3)

   Verify $S_i$ against the close-loop OAC system $[P ∥ C]$
   - If $[P ∥ C] |= S_i$ then
     - goto Step 9.
   - Else
     - goto Step 10.
   - End if

End for.

**Step 10.** Modify system controller $C$ to reach the specification $S_i$ and goto Step 9. (Section 4.5.3)

**Step 11.** Output OAC system controller $C$ (Section 4.5.3)

---

**Algorithm 1:** The procedure of designing the system controller.

---

**Figure 7:** DEC component controller TTM.
the ENV command start_dec, the DEC_CONTROLLER begins to work and issues the command in to the DEC. Then the DEC starts to work until the decoding procedure is completed by issuing a flag variable done. When the DEC_CONTROLLER receives done, it issues a flag out to DEC and gives the value to Gcode or Mcode to decide whether the NC-program segment is a logic order to PLC or an interpolation order to CTC. By receiving these three flags, the DEC runs on a right way and is ready to decode a new segment. If the input NC-program segment is wrong in expression by receiving false value of flag done, the DEC_CONTROLLER will enter into the MCODE_FAULT state or GCODE_FAULT state based on the kind of segment and in the same time will issue the reset flag to start the reset procedure. The procedure is finished and issues the resetting done flag. By receiving this flag, the DEC_CONTROLLER moves the state to DECODING and decodes a new NC-program segment. With the parallel combination of the TTM DEC_ENV and TTM DEC_CONTROLLER, a closed-loop TTM of DEC is constructed and can be analyzed individually.

4.5.2. Constructing the System Controller C of OAC. In this step, the variable set, the sequence of loading components, the communication between components and starting the whole OAC system are controlled by the given system controller. The procedure of designing a system controller is constructed in a hierarchical manner using a bottom-up approach [2, 31]. Based on the results of Section 4.5.1, we can give an initial guess of system controller of the whole OAC system in Step 8 of Algorithm 1. The procedure of guessing is to construct a system controller in TTM which coordinates all the actions of each closed-loop component TTM.

The actions of each closed-loop component are expressed by the interface variables. The interface variables only preserve the function and connection with other components of the OAC system and encapsulate the detail of the components. So when dealing with the whole OAC system, we only face the reduced complexity interface of components which only preserve the sufficient information of designing the system controller of OAC.

In Section 4.3.2, we have already given the concepts of environment and interface variable and its modes. The environment includes all the information of the relationship with the outside and the number of interface variables determines the coupling level of component. The EXTERNAL IN variable is used to get the information from the other components or the system controller and the EXTERNAL OUT variable is used to issue the results of component to the outside. For the DEC component example, it reads the flag variable read nc_finished to start the process and finishes the process by issuing output, Mcode, and Gcode. These four interface variables include all the information about the component DEC with the outside. So we can only consider these kinds of variables when we design the system controller C of OAC.

When we get all the interface variables of all the components, the next work is guessing a reasonable controller C in Step 8 of Algorithm 1. We should consider both the information of all the closed-loop components and the system-level functional property specifications in ATRTTL which we have already presented in Table 4. Although we can use the bottom-up method to give an initial guess of system controller from the relatively simple component TTM in a modular way, we still need a great deal of insights of designer to give a sound process sequence which determines the work load of the whole design procedure. When we get the initial guess of controller, the next step is to verify the controller using an iterative way in Step 9. We modify the controller by adding or replacing the interface variables for different states of system. The iterative procedure is finished when all the property specifications are verified.

In the following, an example is given to design a system controller for a typical NC-program segment in a two-axis OAC system. The program segment is SEGMENT: G01 X10.0 Z10.0 M05 F1000.0. This SEGMENT is a line-interpolation procedure from the start point (0,0) to the end point (10.0,10.0) and the velocity of interpolation is 1000.0 mm/min. When finishing interpolation, the spindle is shut.

Based on the above NC-program segment, we carry out a reasonable processing flow to control the action of OAC system. The guess of initial system controller perhaps is obviously wrong at the early stage and will become more and more precise under the iterative simulation of the system controller. Based on this idea, we can firstly give a relatively sound initial OAC system controller based on the TTM of every component in Figure 8.

4.5.3. Modifying the System Controller C with Verification Tools. In Section 4.5.2, we have already presented a reasonable guess of system controller to the greatest extent, but we still need a formal way to prove that the controller is right and if it has flaws and to know how to find it and correct it in a sound way. The verification tool should be used to analyze and verify the given ATRTTLs. If the given system controller can satisfy the specification of closed-loop OAC [P || C], then the verification tool can give an answer “yes” with outputting the number of states explored. If the verification tool finds an error in the system controller, it will give out a counter-example explaining in which state of the system controller TTM exists a flaw and based on this information we can modify the controller to reach the specification of the closed-loop OAC in ATRTTL and iteratively compute this procedure until all the specifications can be satisfied using the modified controller. The procedure is shown in Step 9 and Step 10 in Algorithm 1.

4.6. Design of Scheduling Mechanism of OAC. From Sections 4.3 and 4.5, we have constructed a typical closed-loop OAC system in TTM. By this model, we can deal with the functional properties and time properties in some conditions. But the system controller only gives a sound but idealized controlling sequence. In reality, in order to analyze and verify real-time properties, we should introduce the scheduling mechanism into the OAC closed-loop system and based on it we then can get the real-time checking results in a proper
way. In this section, we first give the whole structure of the modified OAC system. Based on it, we will then model the scheduling mechanism in TTM. In the last part, we will present the TTM model of the message passing mechanism which is the standard communication method in asynchronous event driven system.

4.6.1. OAC TTM with Scheduling Mechanism. The new OAC TTM with scheduling mechanism includes three concurrently running component TTM modules. One is called OAC_ with_Scheduler. It includes several OAC processes and a system-level scheduler. Every process represents a single component in OAC. Since at any time only one process is permitted to run, the scheduler is used to determine the execution of time sequence. The second concurrent is hardware interrupt. It is used to simulate the hardware timer which generates the time-out instruction in every constant time interval. The third part is the process FIFO queue. It follows the first-in-first-out strategy and it is used to help the scheduler to deal with the scheduling task. The detail of these three concurrent TTM modules is explained later. The whole structured OAC system with scheduling mechanism is shown in Figure 9.

4.6.2. Scheduling Mechanism and FIFO Queue. The whole scheduling mechanism can be depicted as follows. When any component process is ready to be executed, it will write its ID in FIFO queue and wait to be permitted to run. The permission command is achieved from the scheduler. The scheduler itself is permitted to run under the command of hardware interrupt. The hardware interrupt issues the start_running_scheduler command every constant time interval $IT < T < IT + 1$. On receiving the start_running_scheduler command, the scheduler will get the process ID from the top of FIFO queue and this process will be executed immediately.

The TTM of scheduler is shown in Figure 10. It depicts the procedure of getting the process ID from FIFO queue and the update operation of the queue.

Writing the process ID to FIFO mechanism is shown in Figure 11. We can see if the ID already exists in FIFO, then the writing ID to FIFO operation is canceled.

4.6.3. Message Passing Mechanism. The communication between different processes is executed by a message passing mechanism. It follows several steps below.

In transmitting end, first, because of an occurrence of a transition in process, a variable is changed by this transition. Then, the message corresponding to this changed variable is set to be TRUE in a constant time interval Ttime. In receiving end, first, this changed message is received and decoded. If the information of this message is TRUE, then the variable corresponding to this message is also set to be TRUE after a constant time interval Rtime. In the last, the message is reset to be FALSE in a constant time interval Itime in order to be used in the next time.
The TTM of message passing mechanism is shown in Figure 12.

4.7. Verifying System-Level Specification with ATRTTL

4.7.1. Overview of Formal Verification Method. In our formal verification method, we choose two toolsets to be used which are STeP by Stanford Temporal Prover group and SF2STeP by D. Lalita.

STeP is a system being developed to support the computer-aided analysis and verification of many kinds of system such as concurrent system, reactive system, and real-time system. The analysis and verification process is for the most automatic part. It encapsulates many tools and methods.
such as efficient simplification methods, decision procedures, and invariant generation techniques so we can directly use this toolset without concerning the details of verification procedure.

The inputs of the STeP toolset are temporal logic formula and fair transition system. So there is a gap between our TTM/ATR-TTL model and the inputs of the STeP. We should find an effective way to translate the TTM/ATR-TTL model to a form which the STeP can recognize.

Before verifying the OAC system using STeP, we use another toolset SF2STeP. SF2STeP allows converting a generic Stateflow diagram to a timed Stateflow diagram and also can translate the timed Stateflow diagram to a fair transition system. Then with the fair transition system and the temporal logic in ATR-TTL, we can then use the STeP toolset.

The flow chart of the whole verification procedure is shown in Figure 13.

The whole procedure of verification is used to ensure all states of the system are correct and all time sequences are sound.

4.7.2. Some Analysis and Verification Problems through Simulation in STeP

(1) Rewrite Rules. Because the STeP cannot directly process the formulas with binding operators, the real-time specification in ATR-TTL cannot be verified in the straight way. So we need the rewrite rules to change these kinds of formulas with binding operators to a relatively simple one with only one temporal operator.

For example, let us consider the specification CT. let SDt represent the start time of component DEC and let EDt represent the finished time of component DEC; then we can rewrite the ATR-TTL into the following equation as

$$CT: \forall ((EDt - SDt) \geq \text{MinDecTime}) \land ((EDt - SDt) \leq \text{MaxDecTime}). \tag{29}$$

Furthermore, in order to further simplify the real-time property ATR-TTL equation form, we now only consider a global time variable $T$ as the measure standard. We consider
that the times of events in Table 4 are all measured relative to the start time point. We can change (29) into a pair of equations as follows:

CT.1: \( ∅(Start\_Process\_at\_DEC \rightarrow (0 \leq T \leq MinDecTime)) \).

CT.2: \( ∅(End\_Process\_at\_DEC \rightarrow (MinDecTime \leq T \leq MaxDecTime)) \).

Equation (30) is equivalent with (29). In (30), there is only one variable \( T \) to analyze. With this easy form of ATRTTL formula, the real-time properties can be verified into the STeP.

(2) Verification Rules. Based on Definition 8, we can give a simple verification rule which can be used in simplifying the ATRTTL formula of OAC. The most important feature of verification rules is that it can convert a given temporal logic formula.

With Rule 1 (INV), we can only check the verification condition of formula \( \varphi \). Since the only transition which can influence the real-time properties is the tick transition, we now reduce the original real-time ATRTTL formulas in Table 4 to only prove the verification condition of tick transition. Then the model-checking procedure becomes easier to control.

(3) State Explosion Problem. When dealing with real-time properties, the tick transition happens infinitely often so the exploration of states will eventually occur. There are two methods to deal with the state explosion problem. The first one is reduction to absurdity by \( (A \rightarrow B) \Leftrightarrow (\overline{B} \rightarrow \overline{A}) \). Basically if we want to prove the validity of a real-time ATRTTL specification in a normal way to model-check each state in TTM using STeP, we can give a contrary specification of the original one and prove it by contradiction.

The second method to deal with the explosion is modular verification. For the whole CNC system, the number of states is very big. For example, when dealing with the specification such as component real-time CT in Table 4, we only need to focus on the states which are relevant to DEC component without concerning other components and system controller based on the ENV of DEC. Using this method, we can dramatically reduce the number of states.

(4) Time Bound Restriction. Since our OAC system is a typical loop executing system, when dealing with the complicated property ATRTTL such as ST, we can only verify it in a single loop; that is to say, we add another enable condition to every transition in whole OAC system. This enable condition
restricts the global $T$ variable at the value of one cycle of operation. By this method, we can further lower the number of states to enumerate.

4.7.3. Experiments for Verifying an OAC System

(1) Dead Lock. In our TTM of OAC system, the dead lock is generated from the scheduling and the message passing strategy we have already mentioned in Section 4.6. The origin of this problem is because of the uncertainty of the logical control pattern of OAC system. In order to show the dead-lock problem more clearly, a counter example which is generated by model-checking tool of STeP is shown.

From Table 5, we can see when the whole system wants to schedule the SVO process to operate, it first makes the process C become active and at that time the process PCTL and process PLC give the system-level controller message PCTL_Finished and PLC_Finished, respectively. But because process C gets these two input messages at the same time, the behavior of C is not predictable on sending the output message to PCTL process or PLC process or both. If any of these two processes is ignored by C, then a dead lock is reached because the system controller C and any one of these two processes will wait for each other on input message commands.

This problem can be easily solved by changing the scheduling pattern of Figure 10. Before every cycle of routine scheduling task begins to run, we first check whether the system controller process C is in the FIFO queue. If it is, we will then move C to the top of the FIFO; that is to say, since we give the system controller the highest priority, the undetermination of C can be avoided and, at any time, C can only serve one other parallel process. The improved TTM model of OAC scheduler is shown in Figure 14.

(2) Component Time Bound Checking. When we use the model-checking tool of STeP to verify the ATRTTL real-time property ST, we find that not all the computations can
satisfy the closed-loop OAC system and that the verification procedure reaches a counter example of dead loop of tick. But if we change the original time parameter-upper bound of PLC20 to a new value 35, then the model-check procedure is complete. It means even though the logic relationship of the execution sequence is correct, the unsuitable value of time parameters can also have an effect on the OAC system. Based on this verification procedure, we can determine the permissible time bound of each component of OAC without concerning the detail before the development of the real system. This method is helpful for designer because the change of design at the beginning of development will cost less compared to the latter procedures. In our verification result from STeP, we can get the appropriate time parameters of all components which are shown in Table 6. The time bound is not the real OAC system time bound; it only gives the proportion relationship of time values between different processes.

(3) Verification Results Using STeP. The verification results of specifications using STeP are shown in Table 7. The consuming time and the number of states explored are given in Table 7. From Table 7, since all the six categories of OAC specifications are satisfied and no counter-example state exists, we can draw the conclusion that the OAC library component can work under the system controller in the manner of what the specifications of OAC require. The

---

**Figure 14**: Improved TTM model of OAC scheduler.

**Table 5**: Example of event sequence of deadlock.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>Start_running_controller</td>
</tr>
<tr>
<td>0.02</td>
<td>Start_Decode_Segment</td>
</tr>
<tr>
<td>0.05</td>
<td>Message [C2DEC] = 1</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>87.23</td>
<td>Start_running_controller</td>
</tr>
<tr>
<td>87.28</td>
<td>PLC_Finished</td>
</tr>
<tr>
<td>87.28</td>
<td>PCTL_Finished</td>
</tr>
<tr>
<td>87.28</td>
<td>Begin_SVO_Gcode</td>
</tr>
<tr>
<td>87.28</td>
<td>Ignoring Begin_SVO_Mcode</td>
</tr>
<tr>
<td>87.32</td>
<td>Message [PLC2C] = 0</td>
</tr>
<tr>
<td>87.32</td>
<td>Message [C2PCTL] = 1</td>
</tr>
<tr>
<td>87.32</td>
<td>Message [PCTL2C] = 0</td>
</tr>
<tr>
<td>87.32</td>
<td>Ignoring Message [C2PLC] = 1</td>
</tr>
<tr>
<td>88.14</td>
<td>Start_running_scheduler</td>
</tr>
<tr>
<td>88.15</td>
<td>reading_FIFO</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
Table 6: Correct time parameters of components (microsecond).

<table>
<thead>
<tr>
<th>Component</th>
<th>Lower time bound</th>
<th>Upper time bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>PLC</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>CTC</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>INT</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>PCTL</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>SVO</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>DET</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>DSY</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 7: Verification results of functional specifications.

<table>
<thead>
<tr>
<th>Functional specification</th>
<th>Number of states explored</th>
<th>Consuming time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL</td>
<td>23323</td>
<td>84</td>
</tr>
<tr>
<td>PL</td>
<td>25445</td>
<td>92</td>
</tr>
<tr>
<td>S</td>
<td>29986</td>
<td>34</td>
</tr>
<tr>
<td>FS (modularized)</td>
<td>330</td>
<td>1</td>
</tr>
<tr>
<td>ST (rewritten)</td>
<td>23988</td>
<td>12</td>
</tr>
<tr>
<td>CT (rewritten)</td>
<td>2453</td>
<td>2</td>
</tr>
</tbody>
</table>

designed component library and the system controller of closed-loop OAC system are correct.

5. Conclusion and Future Work

In this paper, we proposed a new modeling method called TTM/ATRTTL (timed transition models/all-time real-time temporal logics) for specifying CNC systems based on TTM/RTTTL. TTM/ATRTTL provides full support for specifying hard real time and feedback that are needed for modeling CNC systems. We also proposed a verification framework with verification rules and theorems and implemented it with STeP and SF2STeP. The proposed verification framework can check reliability, safety, and correctness of systems specified by our TTM/ATRTTL method. We applied our modeling and verification techniques on an open architecture CNC system and conducted comprehensive studies on modeling and verifying a system controller that is the key part of an OAC system.

The results show that our method can effectively model and verify OAC systems and generate OAC software that can satisfy system requirements in reliability, dependability, and safety.

In the future, we will investigate how to integrate our TTM/ATRTTL technique into real-time systems so we can generate reliable and dependable system software for CNC systems with our specification and verification methods. We will also study how to translate TTM/ATRTTL specifications into hardware description language such as VerilogHDL so we can directly implement a CNC control system with FPGA.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

References


