International Journal of Grid Distribution Computing
Vol.8, No.1 (2015), pp.179-188
http://dx.doi.org/10.14257/ijgdc.2015.8.1.17

Verification Method of Real-time System Based on Refinement Relation

Jing GUO[1], Zhong-wei XU [2]and Meng MEI[3]

School of Electronics & Information Engineering ,Tongji University
Shanghai ,China
guo_jing163@163.com[1],xuzhongweish@163.com[2], mei_meng@163.com[3]

Abstract

With the continuous increase in the size and complexity of a real-time computer system, the use of formal verification methods in software development is also on the rise. The traditional formal verification method is not fully applicable to the development of actual system life cycle. Therefore, this paper presents a new real-time system verification method, It takes the deadlock timed *Büchi automata as the medium, and translates the timed temporal logic into timed communicating sequential process language. The tock event is also joined, which can be directly used for the detection of refinement tool FDR. The method verifies the situation of deadlock. To establish the link between the conventional model checking and refinement model checking can well combine the advantages of both and improves system security and reliability.*

Key words: *refinement relation; formal verification; real-time system; safety*

1. Introduction

Recent years, model checking has been a powerful automatic formal verification technique for establishing correctness of hardware and software systems. The method based on temporal logic is to model system as a labeled transition system (LTS) and specification as temporal logic formula. Then we decide whether the formula holds in the LTS. Another approach is based on the notion of refinement and is frequently used by verification such as Communicating Sequential Processes (CSP). In this conception, the system and the property are modeled by same formalism, and the aim is to verify the latter is refinement of the former. The two approaches have merits and demerits and there is connection between them. To study the connection, a significant number of techniques have been proposed.

Temporal logic can also used for describing system in the form of temporal logic of actions[1], which is verified by the notion of refinement and development. Therefore, the idea allows simulation of a single high level step by several lower level steps. Spatio-temporal logic[2] is somewhat alike to temporal logic, but different in the operators. It extends the meaning of syntax, where arise in the development of mobile systems, but the decidability of the model checking problem is negligible. The relationship between refinement-oriented specification and specification using a temporal logic is considered, and further the conversion of temporal logic into process algebra [3]. Researchers verified equivalence of computation tree logic and failure trace, in which the former is expressed in the form of test sequence[4]. Despite validity of verification, the infinite states are born in the test case. The temporal logics LTL, CTL and the μ-calculus convert to the formal language Z in the idea of refinement, which also used for language B, VDM based on state. Every conversion rule does not illustrate the main distinction [5].

ISSN: 2005-4262 IJGDC

In the development of system, system construction is stepwise process. The traditional verification method is appropriate for changes of systems, while the refinement applies to verification about life cycle. Temporal logic is more convenient and common. Therefore, rise both union, temporal logic express the property and verify the property based on refinement. Real time factors are of great importance, so these must be taken into account. In this paper, we study the possibility of doing TLTL(timed linear temporal logic) model checking on TCSP specification in the context of refinement. Firstly, we model property as TLTL and define timed *Büchi* automata. Secondly, through the medium of timed *Büchi* automata TLTL is converted into TCSP, the verification is discussed on two cases in the refinement framework.

2. Preliminaries

Definition 1(Timed CSP Syntax) A timed process is defined by the following grammar.

$P = STOP; \text{TIMESTOP}; a \xrightarrow{d} Q; a@u \to Q; A \to Q(x); Q_1 \overset{d}{\triangleright} Q_2; WAITd;$

$P \setminus A; Q_1 \Box Q_2; Q_1 \prod Q_2; \prod_{i \in J} Q_i; Q_1 \ {}_{A}\Box_{B}\ Q_2; Q_1 \underset{A}{\Box} Q_2; Q_1 \,|||\, Q_2; f(Q);$

$Q_1 \Box Q_2; Q_1 \Box_d Q_2; \mu x.P$

STOP is a process which does nothing but is only capable of letting time pass. TIMESTOP terminates but time is blocked. $a \xrightarrow{d} Q$ initially behaves as a, then after d time units, process Q takes over control. The value of d also can be 0, $a \xrightarrow{d} Q$ becomes $a \to Q$. $a@u \to Q$ is similar to $a \to Q$ except that u is variable which bounded by time t when event a happen. $A \to Q(x)$ initially behaves as one of event set A, after it take place Q(x) immediately take over control. $Q_1 \overset{d}{\triangleright} Q_2$ is timeout process that behaves as Q_1 for d time units, if Q_1 fails to communicate any visible event it becomes Q_2. *WAITd* is a process which let time pass for d time units. $P \setminus A$ behaves like P but with all communications in the set A hidden. $Q_1 \Box Q_2$ denotes the deterministic choice between P and Q, which is decided by the first visible event. $Q_1 \prod Q_2$ is similar to $Q_1 \Box Q_2$ except that the choice is nondeterministic. $\prod_{i \in J} Q_i$ is a process set which choice is between them. $Q_1 \ {}_{A}\Box_{B}\ Q_2$ requires Q_1 and Q_2 to synchronize on event set $A \cap B$ and to behave independently of each other with respect to each other. $Q_1 \underset{A}{\Box} Q_2$ is a parallel composition which requires Q_1 and Q_2 to synchronize on event set A and to behave independently of each other with respect to all each other. $Q_1 \,|||\, Q_2$ is a parallel composition of two processes. $f(Q)$ is a renaming process that allows the process to perform the event f(a) whenever Q could perform a. $Q_1 \Box Q_2$ allows the first process Q_1 to execute, but it may be interrupted at any time by an event from process Q_2. $Q_1 \Box_d Q_2$ allows the Q_1 continued for d units of time, after which control is passed to Q_2 , unless Q_1 has terminated previously. $\mu x.P$ is a recursion process whose unique solution to the equation X=P.

Definition 2 (Timed LTL syntax) Timed LTL extend the notion of clock based on LTL. It uses freeze operator to record time and is bounded by timed constraint, the grammar is as follows:

$\phi = p; \neg p; \phi_1 \vee \phi_2; \phi_1 \wedge \phi_2; t \ \Box\ x + c; x.\phi; \phi_1 U \phi_2; \phi_1 V \phi_2; \Box\ \phi; \Box\phi;\ \ \Diamond\phi$

where $p \in AP$, t refers to the current time in timed trace, x is discrete formula clock, $\Box \in \{<, \leq, =, >, \geq\}$.

p represents atomic proposition, $\neg p$ represents negative atomic proposition. $\phi_1 \vee \phi_2$ represents Q_1 or Q_2 is untenable. $\phi_1 \wedge \phi_2$ represents Q_1 and Q_2 is untenable. $t \square x+c$ checks t against x+c. $x.\phi$ replaces timed formula including x by the time in the ϕ . $\phi_1 U \phi_2$ holds if there is some time in the future t where ϕ_2 holds and on all time points up to this ϕ_1 holds. $\phi_1 V \phi_2$ has two possibilities: there is some time in the future t where ϕ_1 holds and all time points up to this ϕ_2 holds; ϕ_2 hold forever if ϕ_1 does not hold. $\square \phi$ represents the next point in the trace is true. $\square \phi = falseV\phi$ and $\Diamond \phi = trueU\phi$ always hold. $\sigma = \sigma_0 \sigma_1 \cdots \sigma_i \cdots$ is infinite timed trace. $\sigma_i = (s_i, t_i)$ contains two parts: time $t_i(t(\sigma_i))$ and proposition which is true in the time $t_i(s(\sigma_i))$. We define $\square \phi \square_\varepsilon$ of formula ϕ with free clock variables in the domain of environment ε, inductively as follows:

$\square p \square_\varepsilon = \{ p \in s(\sigma_0) \}$

$\square \neg p \square_\varepsilon = \{ p \notin s(\sigma_0) \}$

$\square \phi_1 \vee \phi_2 \square_\varepsilon = \square \phi_1 \square_\varepsilon \vee \square \phi_2 \square_\varepsilon$

$\square \phi_1 \wedge \phi_2 \square_\varepsilon = \square \phi_1 \square_\varepsilon \wedge \square \phi_2 \square_\varepsilon$

$\square t \square x + c \square_\varepsilon = \{ t(\sigma_0) \square \varepsilon(x) + c \}$

$\square x.\phi \square_\varepsilon = \{ \sigma \in \square \phi \square_\varepsilon [t(\sigma_0)/x] \}$

$\square \phi_1 U \phi_2 \square_\varepsilon = \{ \exists i. (\sigma^i \in \square \phi_2 \square_\varepsilon \; and \; \forall j, 0 \le j < i. \sigma^j \in \square \phi_1 \square_\varepsilon) \}$

$\square \phi_1 V \phi_2 \square_\varepsilon = \{ \exists j. (\sigma^j \in \square \phi_1 \square_\varepsilon \; and \; \forall i, 0 \le i \le j. \sigma^i \in \square \phi_2 \square_\varepsilon) \; or \; \forall i \sigma^i \in \square \phi_2 \square_\varepsilon \}$

$\square\square \phi \square = \square\square \square\square \phi \square$

3. Refinement Verification based on TLTL

3.1 Refinement Verification

Specification contains all behaviors which system should satisfy. Refinement resolves whether system behavior set is subset of specification behavior set. Pair $(s, \aleph)$ is termed a timed failure. Trace s will be the sequence of events occurring in the execution, the refusal set $\aleph$ will be set of timed events which can be refused in the execution. For example, $(\langle (2,a) \rangle, [3,4) \times \{b\})$ records that during the execution the process performed event a at time 2, and refused event b over the interval from 3 to 4. For environment, at special time point it provides events to process. If timed event is member of trace, process will receive them. Otherwise, process will refuse them. Specification is predicable of timed failure, while timed failure is constraint of process. So If specification $S(s, \aleph)$ contains timed failure about process Q, Q will satisfy specification $S(s, \aleph)$.

Q sat $S(s, \aleph) \Leftrightarrow \forall S(s, \aleph) \in TF \square Q \square \bullet S(s, \aleph)$

Specification can be process-oriented approach. Refinement relation $P_1 \hat{o}_{FDR} P_2$ is used to verify specification. It holds when the latter process has fewer things than the former, Remove the times of the timed trace, we get strip(s). If Q and S are defined processes, there must be some untimed P which allows the relationship to be completed:

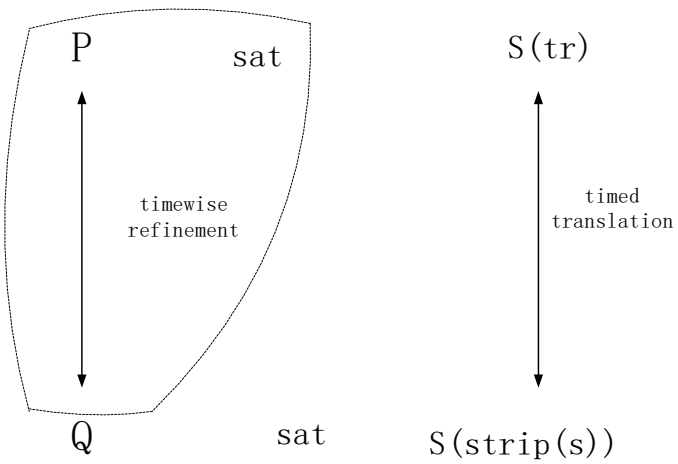


Figure 1. Timewise Refinement and Timed Translation

The refinement can be investigated through transitivity, and the relationship between traces and timed failures is also included. The traces model interacts with timed failures model within timewise refinement and refinement:

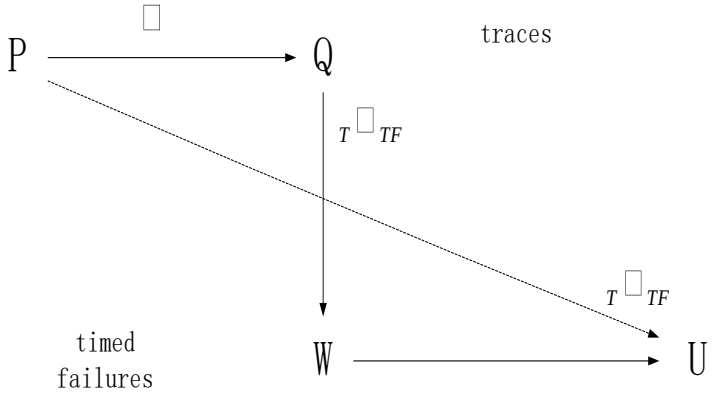
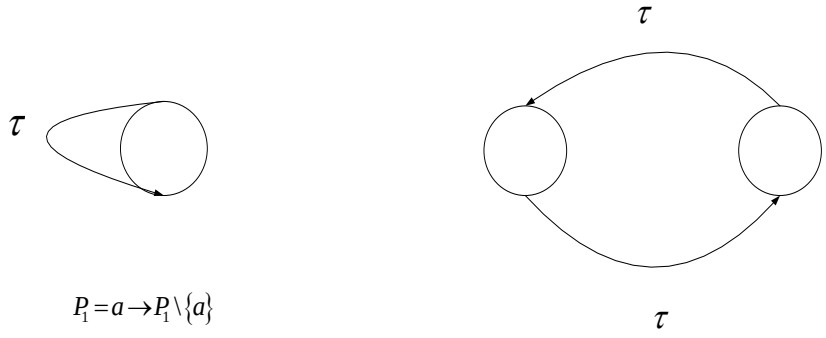


Figure 2 Timewise Refinement and Transitivity

We suppose that the model was stable failures model, which records events as it perform them, and refusal after the process stabilize. When process able to perform an sequence of infinite internal transitions. For example, the process $P_1 = a \to P_1 \setminus \{a\}$ is divergence execution. The process P_2 has <c>, <a,b,c> as possible divergence traces, (<a,b>,<b>) as possible failure.



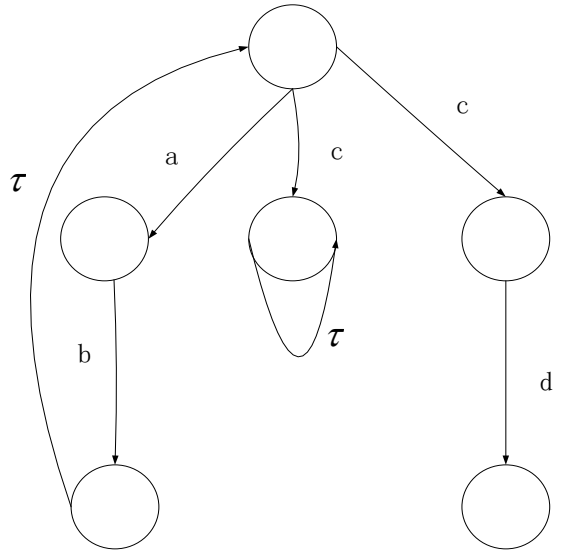


Figure 3. Divergence Process

3.2 Conversion of TLTL

Translation of TLTL into TCSP is of first importance. Timed $B\ddot{u}chi$ automata tests emptiness, and events being tested are visible. During execution doing nothing but empty events, process goes into deadlock state. We start with definition of deadlock timed $B\ddot{u}chi$ automata.

(Deadlock Timed $B\ddot{u}chi$ Automata) $A_D = (C, Q, B, q_0, E, I_c, F, D)$

1)C refers to a set of finite clock.

2)Q refers to a set of discrete states, the initial state is q_0.

3)B refers to a set of finite channel. Every channel has two visible action: input action $a?$ indicates that through channel a; output $a!$ indicates that through channel a.

4)E refers to a set of transition($e = (q, B_{?!}, G(c), R_c, q') \in E$), q and q' are separately source and destination state. $B_{?!}$ contains input and output actions. $g \in G(c)$ is connection of clock constraint($g ::= c \rhd\lhd n; g \wedge g; true$, $n \in N_0$, $\rhd\lhd \in \{<, \leq, =, >, \geq\}$), $R_c \subseteq C$ is a set of clock sets which should be reset. Transition is a form of $q \xrightarrow{g, R_c} q'$.

5) $I_c : Q \to G(c)$ is a set of invariants mapping state to guard.

6) $F \subseteq Q$ is a set of accepting states.

7) $D \subseteq Q$ is a set of deadlock states.

A_D includes two acceptance conditions: traditional acceptance condition for Timed $B\ddot{u}chi$ Automata and special acceptance condition end with deadlock state. We should handle transition about deadlock state. Transition contains change of states and clock variables, and the accepting states including deadlock events are deadlock states. The deadlock transitions are removed from automata. The transition system of automata is of great importance for transformation of automata into TCSP, we define it as follows:

(Syntax of Transition System about Deadlock timed $B\ddot{u}chi$ Automata) $TS^{A_D} = (S, s_0, T)$

1) $s \in S$ is a pair of (q,v), among which q is state and v is valuation of clock variable. v' is clock variable of next state, which has two forms: $v'(c) = v(c) + \delta$ and $v' = v[R]$.If $c \in R$, clock should be reset. q_D and q_A are separately deadlock state and acceptance state.

2)Initial state is a form of $s_0 = (q_0, v_0)$, for every $c \in C$, $v_0(c) = 0$.

3)Transition T has two forms: $(q,v) \xrightarrow{a,g,R} (q', v')$ whereas $v \models g$, $v' = v[R]$ and $v' \models I_c(q')$; $(q,v) \xrightarrow{a,\delta} (q', v') = (q', v+\delta)$ whereas $v + \delta \models I_c(q)$.

According to the above transition system, we establish the relationship between it and TCSP. We define the translation of process as follows:

1)We map state q to a process $P \in \Sigma$, name the initial state and take it as start point.

2)For every non-accepting state, it is expressed as $label(q) = a \xrightarrow{d} label(q')$. If clock is reset, d will be 0. If clock is increasing, d will be δ.

3)For every accepting state, all destination states are $q_1', q_2' \cdots q_n'$ and actions are $a_1, a_2 \cdots, a_n$. We add one event accept, the choice of visible events are possible transitions. The process is $label(q) = accept \to \left(a_1 \xrightarrow{d} label(q_1') \Box \cdots \Box a_n \xrightarrow{d} label(q_n') \right)$.

4)For every deadlock states, all destination states are $q_1', q_2' \cdots q_n'$ and actions are $a_1, a_2 \cdots, a_n$. We add two events: deadlock and special. The process is $label(q) = deadlock \to \left(a_1 \xrightarrow{d} special \to STOP \Box \cdots \Box a_n \xrightarrow{d} special \to STOP \right)$.

3.3 Refinement Verification Process

Using TCSP process to verify TLTL formula, the prime thing is to verify $S \models \phi$. We show the specification as series of TLTL, negate the formula and translate it into timed $B\ddot{u}chi$ automata. Secondly, we translate timed $B\ddot{u}chi$ automata into deadlock timed $B\ddot{u}chi$ automata. The last thing is the translation of them into TCSP.

Two processes, which under test have same result, could be same to some extent. $\left(P \underset{\Sigma}{\Box} T \right) \backslash \Sigma$ hides all the events in the Σ.Must testing consider the combination's maximal executions that is same as timed failure model. May testing is weaker than it, only considering finite duration timed failures. May and must testing give rise to related notions of refinement:

$P_1 \hat{\delta}_{must} P_2 = \forall \bullet (P_1 \; must \; T) \Rightarrow (P_2 \; must \; T)$

$P_1 \hat{\delta}_{may} P_2 = \forall \bullet \neg (P_1 \; may \; T) \Rightarrow \neg (P_2 \; may \; T)$

The may testing is a form of traces refinement, and must testing is FDI refinement. The relationship between untimed CSP processes and timed CSP processes is timewise refinement. Time removing operator extracts an untimed transition from the timed operational semantics for timed CSP process. It is just a mechanism which provides an untimed view of a timed process.

$$\frac{T \xrightarrow{\mu} T'}{\Theta(T) \xrightarrow{\mu} \Theta(T')}$$

$$\frac{T \overset{d}{\mapsto} T' \quad T \overset{\mu}{\to} T'}{\Theta(T) \overset{\mu}{\to} \Theta(T')}$$

May testing also can defined by the removing operator:

$P \hat{o}_{may}^{time} Q = \forall T \square \neg (P \ may \ \Theta(T)) \Rightarrow \neg (Q \ may \ T)$

Q and T are timed CSP processes, and P is untimed CSP process. Must testing is defined analogy with the definitions for must testing:

$P \hat{o}_{must}^{time} Q = \forall T \square (P \ must \ \Theta(T)) \Rightarrow Q \ must \ T$

Related equivalence is as follows:

$P_1 \hat{o}_{must} P_2 \Leftrightarrow P_1 \hat{o}_{TF} P_2$

$P_1 \hat{o}_{may} P \Leftrightarrow P_1 \hat{o} f_{FTF} P_2$

After the translation of Deadlock Timed *Büchi* automata into TCSP, can test as form of $\left(P \underset{\Sigma}{\square} T\right) \setminus (\Sigma \cup \{v, \square\})$. S is description of system, T is the result of translation. Not only do we consider emptiness, but also we must consider deadlock. So test has two conditions: the former accept is test event and deadlock, special should be hidden. We test whether the result has infinite accept, which is analogous to may test, test accept is refinement of $\left(P \underset{\Sigma}{\square} T\right) \setminus (\Sigma \cup \{deadlock, special\})$; the latter test whether $deadlock \to STOP$ execute, which is analogous to must test, test $deadlock \to STOP$ is refinement of $\left(P \underset{\Sigma}{\square} T\right) \setminus (\Sigma \cup \{accept\})$.

4. Verification Case: Railway Crossing

The system consists of three components: a train, a gate, and gate controller. When no train is approaching, the gate should be up to allow traffic to pass. When train is close to reaching the crossing, the gate should be lowed to obstruct traffic. The controller is used to monitor the approach of a train, to instruct the gate to be lowered within the appropriate time. From the aspect of system, only focus on part of behavior.

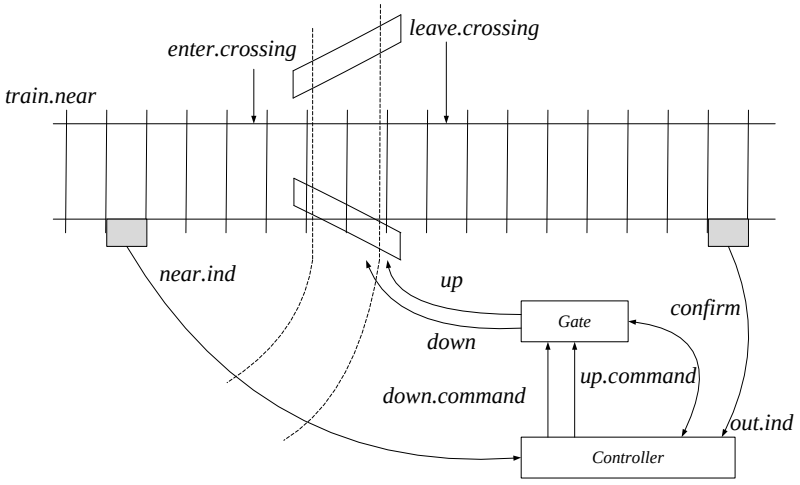


Figure 4 The Railway Crossing System

The description of the crossing system is as follows:

$$Train = train.near \to near.ind \xrightarrow{300} enter.cros\sin g$$
$$\xrightarrow{20} leave.cros\sin g \to out.ind \to Train$$
$$Gate = down.command \xrightarrow{100} down \to confirm \to Gate$$
$$\square up.command \xrightarrow{100} up \to confirm \to Gate$$
$$Controller = near.ind \xrightarrow{\varepsilon} down.command \to confirm \to Controller$$
$$\square out.ind \xrightarrow{\varepsilon} up.command \to confirm \to Controller$$
$$Cros\sin g = Controller \ {}_{C}\square_{G} \ Gate$$
$$System = Train \ {}_{T}\square_{C\cup G} \ Cros\sin g$$

The events $near.ind$, $out.ind$ model respectively tell the sensors that train has enter and leave. The events $train.near$, $enter.cros\sin g$, $leave.cros\sin g$ model respectively the situations where the train is close to the crossing, the train enters the crossing, and the train leaves the crossing. $down.command$, $up.command$ instruct the gate to go down and up respectively. C, G and T are set of events which describe controller, gate, and train respectively.

If train enter the crossing, during 10 units of time up and down do not occur. Check the property, the form of TLTL is $\neg\Diamond(enter.cros\sin g \Rightarrow x.t \le x+10 \wedge \neg down \wedge \neg up)$, the result TCSP of translation is as follows:

$$T = State1$$
$$State1 = enter.cros\sin g \to State2$$
$$State2 = down \xrightarrow{[0,10]} State3$$
$$State2 = up \xrightarrow{[0,10]} State3$$
$$State3 = accept \to ((down \to State3)\square(up \to State3))$$
$$State3 = deadlock \to ((down \to special \to Stop)\square(up \to special \to Stop))$$

The FDR tool support the test, tock event represents one unit of time that establishes the relationship between TCSP and CSP. Firstly, we test emptiness, test whether the result has infinite accept:

$$Acept = accept -> Acept$$
$$assert\ Composition1\ [T = Acept$$

The test result is refinement fails=>no infinite trace violates formula=>OK, showing that it does not produce infinite accept.

We test deadlock, test deadlock trace whether receive the negative of the property:

$$Deadlock = deadlock -> Stop$$
$$assert\ Composition2\ [T = Deadlock$$

The test result is refinement fails=>no deadlock trace violates formula=>OK, showing that there is not deadlock trace in the system. According to the two results, system meet the property.

5 . Conclusions

This paper explored the relationship between TLTL and TCSP. We took deadlock timed $B\ddot{u}chi$ automata as medium, realized the translation between them. TCSP provided a counter example if a refinement check fails and not. We highlighted a case, railway

crossing, verified the property. The experiments showed that people can verify the TLTL property in the framework of FDR, so the result is to improve efficiency, and can be well used in the development of system.

Acknowledgements

The work is support by the National Natural Science Foundation of China under grant No .60674004 , No.61075002, the National "Twelfth Five-Year" Plan for Science & Technology Support under grant No.2011BAG01B03, the railway ministry basic research program of China under grant No.2012AA112801, the National High Technology Research and Development Program 863 under grant No. 2012AA112801.

References

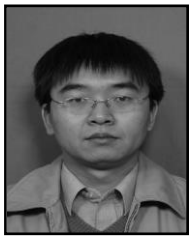
[1] A. Pnueli, "System Specification and Refinement in Temporal Logic", Proceedings of Foundations of Software Technology and Theoretical Computer Science, proceeding of the 12th conference, **(1992)**; Bangalore, India.
[2] S. Merz, M. Wirsing and J. Zappe, "A Spatio-Temporal Logic for the Specification and Refinement of Mobile Systems", In 6th International Conference Fundamental Approaches to Software Engineering, **(2003)**; Warsaw, Poland.
[3] G. Lowe, "Specification of communicating processes: temporal logic versus refusals-based refinement", Formal Aspects of Computing, vol. 20, no. 3, **(2008)**.
[4] Z. Y. Zhang, "Model Checking is Refinement-Computational Temporal Logic is Equivalent to Failure Trace Testing", Sherbrooke, Quebec, Canada: Bishop's Unversity, **(2008)**.
[5] J. Derrick and G. Smith, "Temporal-logic property preservation under Z refinement", Formal Aspects of Computing, vol. 24, no. 3, **(2012)**.
[6] S. Schneider, "Concurrent and Real-time Systems: The CSP Aproach", Wiley Press, USA, **(2000)**.
[7] K. J. Kristoffersen, C. Pedersen and H. R. Andersen, "Runtime verification of timed LTL using disjunctive normalized equation systems", Electronic Notes in Theoretical Computer Science, vol. 89, no. 2, **(2003)**.
[8] S. C. Cheung, D. Giannakopoulou and J. Kramer, "Verification of Liveness Properties Using Compositional Reachability Analysis", Proceeding of: Software Engineering - ESEC/FSE, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering, **(1997)**; Zurich, Switzerland.
[9] J. Bengtsson and W. Yi, "Lectures on Concurrency and Petri Nets", vol. 3098, **(2004)**, pp.87-124.
[10] O. Maler,Dejan Nickovic and A. Pnueli, "Real Time Temporal Logic: Past, Present, Future", Third International Conference, **(2005)**; Uppsala, Sweden.
[11] C. Morgan, A. McIver, K. Seidel and J. W. Sanders, "Refinement-oriented probability for CSP", Formal Aspects of Computing, vol. 8, no. 6, **(1996)**.
[12] T. Murray, "On the Limits of Refinement-Testing for Model-Checking CSP", Formal Aspects of Computing, vol. 25, no. 2, **(2013)**.
[13] A.W. Roscoe, "On the expressive power of CSP refinement", Formal Aspects of Computing, vol. 17, no. 2, **(2005)**.
[14] T. Göthel and S. Glesner, "An approach for machine-assisted verification of Timed CSP Specifications", Innovations in Systems and Software Engineering, vol. 6, no. 3, **(2010)**.
[15] R. M. Hierons and K. Bogdanov, "Using Formal Specifications to Support Testing", ACM Computing Surveys, vol. 41, no. 2, **(2009)**.
[16] S. Schneider, "Abstraction and Testing in CSP", Formal Aspects of Computing, vol. 12, no. 3, **(2000)**.
[17] Y. Isobe1 and M. Roggenbach, "A Generic Theorem Prover of CSP Refinement", In European Conferences on Theory and Practice of Software, **(2005)**; Edinburgh, UK.
[18] G. Behrmann, A. David, K. G. Larsen and W. Yi, "Unification & sharing in timed automata verification", Proceedings, Tenth International SPIN Workshop, **(2003)**; Portland, USA.
[19] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina and N. Sinha, "State/Event-based Software Model Checking", Proceeding of Integrate Formal Methods 4th International Conference, **(2004)**; Cnaterbury, UK
[20] Oxford University Timed CSP Group, Timed CSP: Theory and practice, vol. 600, **(1992)**, pp. 640-675.
[21] G. M. Reed, A. W. Roscoe and S. A. Schneider, "CSP and timewise refinement", Proceedings of the Fourth BCS-FACS Refinement Workshop, **(1991)**; Cambridge.
[22] U. Montanari, "True concurrency: Theory and practice Springer Berlin Heidelberg", **(1993)**.

[23] S. A. Schneider, "Concurrent and Real Time Systems: the CSP approach", John Wiley, **(2000)**.
[24] V. Sokolsky and S. A. Smolka, "Local model checking for real time systems", Proceedings of the Seventh International Conference on Computer-Aided Verification(CAV), **(1995)**; Belgium.

Authors

Jing GUO, she is currently a student in School of Electronics & Information Engineering at Tongji university. Her research interests include formal methods, software testing.

Zhong-wei XU, he is professor of School of Electronics & Information Engineering at Tongji university. His research interests include Communication based train control System, testing of safety and communication protocol.



Meng MEI, he is a Lecturer at Tongji university. His research interests include service trust and security, software engineering.

