Model Checking of Embedded Systems Using RTCTL

Yajun Wu * and Satoshi Yamane †
* Graduate School of Natural Science and Technology
Kanazawa University, Kanazawa, Japan
Email: kwu@csl.ec.t.kanazawa-u.ac.jp
† Institute of Science and Engineering
Kanazawa University, Kanazawa, Japan
Email: syamane@is.t.kanazawa-u.ac.jp

***Abstract*—For embedded systems, verifying both real-time properties and logical validity are important. In order to verify real-time properties, we develop a simulator to propose model checking method. The simulator generates timed Kripke structure by dynamic program analysis. We implement the simulator to construct timed Kripke structure including the execution time. Also, we implement model checker in order to verify whether timed Kripke structure satisfies RTCTL formulas.**

I. Introduction

In recent years, embedded systems have been widely used in autonomous car, medical equipment and IoT (Internet of Things). Verifying both real-time properties and logical validity are important for embedded systems.

Recently software model checking [1] and program verification [2] have received widespread attention. S. Yamane and others have developed a verification system [3] for verifying embedded assembly programs. It generates Kripke structure by the simulator (dynamic program analysis) and verifies stack overflow by CTL (Computational Tree Logic) model checking.

In this paper, we will describe model checking using RTCTL (Real-Time CTL) in order to verify whether timed Kripke structure satisfies real-time properties. We develop the simulator to generate timed Kripke structure from assembly program. In our approach, we implement the simulator to construct timed Kripke structure which includes the execution time. Also, we implement model checker in order to verify whether timed Kripke structure satisfies real-time properties. In general, as timed Kripke structure has infinite states, our approach is effective. In our study, to evaluate a proposed method, we conducted experiments with the implementation of the verification system.

II. Related Works

B. Schlich studied model checking by using static program analysis to verify assembly program, but real-time properties are not verified. B. Schlich can verify the hardware dependency problem by using the assembly program [4].

E. A. Emerson and others proposed RTCTL, and also developed model checking algorithms for RTCTL [5]. But our semantics of RTCTL is quite different from E. A. Emerson's semantics of RTCTL.

R. Alur and D. L. Dill studied timed automata [6]. Timed automaton is an extension of a finite state automaton, and is a model that describes the system by both discrete event and continuous time lapse according to state transitions. On the other hand, in this study, we develop discrete-time timed Kripke structure for the execution time of assembly program. Our study is different from timed automata.

S. Yamane and others have developed a verification system. The simulator generates Kripke structure, and model checker verifies CTL formulas. The simulator includes clock cycles while generating models, but real-time properties and execution time are not taken into consideration. S. Yamane and others have developed abstraction techniques by using DND (Delayed NonDeterminism) of the bit level. Our proposed method is quite different from [3] as follows: (1) Generating timed Kripke structure including execution time. (2) Using RTCTL formulas to verify real-time properties. (3) Proposing RTCTL model checking algorithm after generating timed Kripke structure.

III. Computational Model of Embedded Assembly Program

As a computational model, we define a timed Kripke structure which extends Kripke structure [7] by time function.

[Definition 3.1] (Timed Kripke structure) Timed Kripke structure is $M = (S, S_0, R, L, TM)$.

(1) A finite set of states S.

(2) A set of initial states $S_0 \subseteq S$

(3) A transition relation $R \subseteq S \times S$

(4) A labeling function $L: S \rightarrow 2^{AP}$. L is a function which assigns to each state a set of atomic propositions. AP is atomic proposition.

(5) A time function TM: S $\rightarrow$ N . TM is a function that assigns the execution time of each instruction to each state. Here N is any natural number.

IV. RTCTL

For model checking of real-time properties, we define Real-Time temporal logic (RTCTL(Real-Time CTL)). We use RTCTL developed by E. A. Emerson [5].

[Definition 4.1] (Syntax of RTCTL) Syntax of RTCTL formulas as follows:

(1) Each atomic proposition AP is a RTCTL formula.

(2) If p, q are RTCTL formulas, similarly p $\wedge$ q and $\neg$p are RTCTL formulas.

(3) If p, q are RTCTL formulas, similarly E(p U q), A(p U q) and EXp are RTCTL formulas.

(4) If p, q are RTCTL formulas and k is any natural number for execution time, similarly E(p $U^{\leq k}$ q) and A(p $U^{\leq k}$ q) are RTCTL formulas.

Some other RTCTL are defined as follows:

$AF^{\leq k}$ q = A(true $U^{\leq k}$ q) $\quad$ $EF^{\leq k}$ q = E(true $U^{\leq k}$ q)

$AG^{\leq k}$ p = $\neg EF^{\leq k}\neg$p $\quad$ $EG^{\leq k}$ p = $\neg AF^{\leq k}\neg$p

We propose the semantics of RTCTL as follows.

[Definition 4.2] (Semantics of RTCTL) In the semantics of RTCTL formulas, other than E(p $U^{\leq k}$ q) and A(p $U^{\leq k}$ q) are the same as those of RTCTL by E. A. Emerson and others. Our semantics of E(p $U^{\leq k}$ q) and A(p $U^{\leq k}$ q) is quite different from E. A. Emerson's semantics [5]. In the semantics of E(p $U^{\leq k}$ q) and A(p $U^{\leq k}$ q) is defined over timed Kripke structure M = (S, S_0, R, L, TM).

(1) E(p $U^{\leq k}$ q)

For a state sequence $s_0, s_1, \ldots, s_j, \ldots, s_i, \ldots$ in M, $\exists i\ 0 \leq i,\ s_i \models q$ and $\sum_{\alpha=0}^{i} TM(s_\alpha) \leq k$ and $\forall j\ 0 \leq j < i,\ s_j \models p$.

(2) A(p $U^{\leq k}$ q)

For all state sequence $s_0, s_1, \ldots, s_j, \ldots, s_i, \ldots$ in M, $\exists i\ 0 \leq i,\ s_i \models q$ and $\sum_{\alpha=0}^{i} TM(s_\alpha) \leq k$ and $\forall j\ 0 \leq j < i,\ s_j \models p$.

V. Verification System

The verification system is shown in Fig. 1.

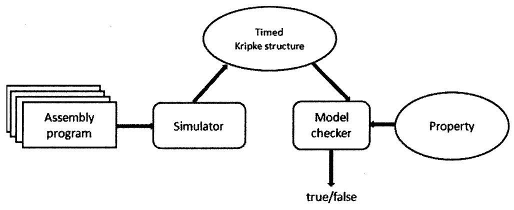

Fig. 1. Verification system

Next, the model checking algorithm of E(p $U^{\leq k}$ q) is defined in Algorithm 1. A(p $U^{\leq k}$ q) is almost the same as E(p $U^{\leq k}$ q). We generate states for the verification target and output a timed Kripke structure by the simulator. Simultaneously, model checking is performed by inputting RTCTL formula and timed Kripke structure.

We use |f| to denote the length of RTCTL formula f (E(p $U^{\leq k}$ q)). The calculation of |f| is the same as that of E. A. Emerson and others [5]. When |f'| is the length of the CTL formula f' obtained from f by deleting time constraint, |f| = |f'| + c and c is the sum of the lengths of the bit strings representing in binary the time constraint of f. Thus, we see that the complexity of executing the entire procedure E(p $U^{\leq k}$ q) model check is O(|f| (|S| + |R|)).

Algorithm 1 Algorithm of MocelCheckE(p $U^{\leq k}$ q)

```
1  S := { s | q ∈ L(s) }
2  for s ∈ S do L(s) := L(s) ∪ E(p U^{≤k} q)
3  N := k H := k                    // initialize N and H
4  while S ≠ ∅ do
5    choose s ∈ S
6    S := S \ {s}                    // remove s from S
7    N := H                          // update N by H
8    for all t such that R(t, s) do
9      if E(p U^{≤H} q) ∉ L(t) and N ≥ 0 and p ∈
   L(t) then
10        H := N - TM(s)                 // update H
11        L(t) := L(t) ∪ {E(p U^{≤H} q)}
12        S := S ∪ {t}                   // add t to S
13      else if N < 0 and S ≠ ∅ then
14       choose y ∈ S// other path has not checked
15        choose E(p U^{≤H} q) from L(y)
16        choose H from E(p U^{≤H} q)
17      end if
18   end for all
19   if R(t, s) = ∅ then //state s is the initial state
20     H := N - TM(s)
21      if H < 0 and S ≠ ∅ then
22      choose y ∈ S
23      choose E(p U^{≤H} q) from L(y)
24      choose H from E(p U^{≤H} q)
25      else if H ≥ 0 then
26        break
27        end if
28      end if
29   end if
30 end while
31 N := H                            // update N by H
32 if N ≥ 0
33 return true
34 else return false
```

[Example 1] We will give an example in order to explain Algorithm 1. We specify verification property by RTCTL as follows.

$$E(p\ U^{\leq 10}\ q)$$

We verify whether timed Kripke structure in Fig. 2 sataisfies $E(p\ U^{\leq 10}\ q)$ according to Algorithm 1 as follows.

$S = \{s_1, s_2, s_3, s_4, s_5\}$	$S_0 = \{s_1, s_3\}$
$R = \{(s_1, s_2), (s_2, s_5), (s_3, s_4), (s_4, s_5)\}$	
$L(s_1) = \{p\}$	$L(s_2) = \{p\}$
$L(s_3) = \{\}$	$L(s_4) = \{p\}$
$L(s_5) = \{q\}$	$TM(s_1) = 1$
$TM(s_2) = 1$	$TM(s_3) = 1$
$TM(s_4) = 100$	$TM(s_5) = 1$

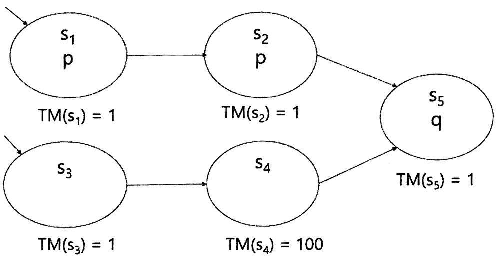

Fig. 2. An example of timed Kripke structure

This formula is obviously satisfied since there is a path $s_1 \rightarrow s_2 \rightarrow s_5$.

To verify the RTCTL formula Algorithm 1 proceeds the following steps. Fig. 3, Fig. 4 and Fig. 5 give snapshots of Algorithm 1 in operation on the timed Kripke structure for the labeling function.

(1) In the first stage:

- Initially, $S := \{s_5\}$ (line 1) and $L(s_5) := \{q, E(p\ U^{\leq 10}\ q)\}$ (line 2), as shown in Fig. 3. Then $N := 10$, $H := 10$ (line 3).
- Choose and remove s_5 from S (line 4, 5, 6).
- Assign $N := H := 10$ (line 7).
- There are two states s_2 and s_4 that satisfy $R := \{(s_2, s_5), (s_4, s_5)\}$(line 8).
  - Choose $t = s_2$ first. Then assign $H := N - TM(s_5) := 9$ (line 10). Next add $E(p\ U^{\leq 9}\ q)$ to $L(s_2)$ (line 11) as shown in Fig. 4, and add s_2 to S (line 12).
  - Next, Choose $t = s_4$ (line 9). But $p \notin L(s_4)$, nothing to do here.

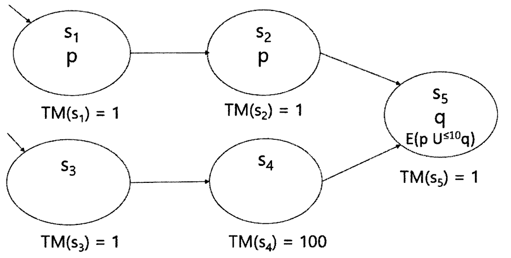

Fig. 3. First step of labeling

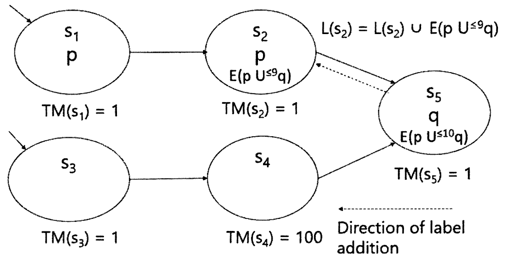

Fig. 4. Second step of labeling

(2) In the second stage:

- Now, $S := \{s_2\}$ (line 4). Similarly choose and remove s_2 and assign $N := H := 9$ (line 7).
- There is one state s_1 that satisfies $R := \{(s_1, s_2)\}$, and choose it (line 8).
  - Then assign $H := N - TM(s_2) := 8$ (line 10). Next add $E(p\ U^{\leq 8}\ q)$ to $L(s_1)$ (line 11) as shown in Fig. 5, and add s_1 to S (line 12).

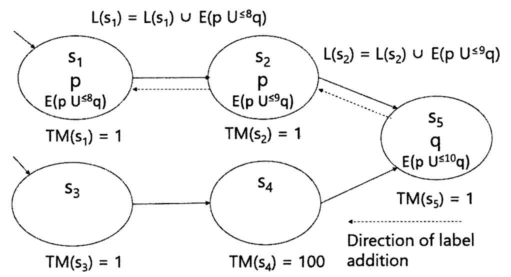

Fig. 5. Third step of labeling

(4) Only in the final stage:

- $S := \{s_1\}$ (line 4). Similarly choose and remove s_1. Then assign $N := H := 8$ (line 7).
- There is no state t that satisfies $R := \{(t, s_1)\}$ (line 8, line 19).
  - Then updata $H := N - TM(s_1) := 7$ (line 20).

- Therefore, H $\geq$ 0, then leave the loop, and updata N = 7 (line 31).
- Due to N $\geq$ 0. So the verification result becomes true (line 33).

VI. EXPERIMENTS

We used LED program written by H8/3687 microcontroller [8][9]. The size of the program has 64 lines of assembly code.

The execution time is obtained from the execution state number of the assembly instruction. Since the operating frequency of H8/3687 microcontroller is 20 MHz, the execution time per state is as follows.

$$1 / 20\text{Mhz} = 0.05\mu\text{s}$$

Therefore, we assume the unit time of time constraint k is 0.05μs.

The experiment was conducted on a laptop as follows:

- CPU : Intel(R) Core(TM) i7-7700HQ processors running at 2.80GHz
- Memory : 16GB main memory
- OS : windows 10
- Simulator is written in a combination of Java and Scala, and Model Checker is written in Scala as follows.
- Java 1.8.0_121 , 15000 lines
- Scala 2.11.8 , 6000 lines
- Tools : JFlex [10], Jacc [11]

Program 1 (LED program)lights up one LED by a sensor of the microcontroller outputs. The sensor can identify black and white. When the sensor of the microcontroller detects white, it outputs 1 and LED lights up, when it detects black, it outputs 0. Furthermore, we preset the sensor always detected white and it turns off immediately after the LED lights up. For example, Total$_{led}$ times LED lights up by Total$_{led}$ times the sensor outputs. We specify timing constraints by RTCTL for this experiment as follows:

RTCTL 1: $EF^{\leq k}(\text{Total}_{led} = n) = E(\text{true } U^{\leq k} (\text{Total}_{led} = n))$

Total$_{led}$ denotes the total number of lights up times. The n is the number of lights up times, indicated by the value of the register R0 in the state. The formula intuitively means that LED lights up Total$_{led}$ times happen at some state on some path from initial states within the timing constraint k. Besides we preset the specific LED light up number while timing constraint k is limited within one millisecond (k = 20000).

The experimental results are shown in Table I. The first column presents the total number of lights up times. The second column gives the number of states. The third column indicates the number of transition relations. The fourth column presents the total time of simulator and model checking. The fifth column presents execution time of assembly program for some path satisfies Total$_{led}$ times lights up LED or the execution time of assembly program for a certain path is only not satisfied time constraint. The last column shows the result which shows satisfiability of RTCTL formula. Execution time is obtained from the total execution time of executed states. The execution time of each state is an execution time of assembly instruction executed in each state.

TABLE I
THE RESULTS OF VERIFYING PROGRAM 1

n	states	relations	verification time(s)	execution time(ms)	result
10	370	369	4.7	0.0507	true
50	1490	1489	23.3	0.2207	true
100	2890	2889	48.4	0.4332	true
200	5690	5689	115.0	0.8582	true
233	6614	6613	118.5	0.9985	true
300	8490	8489	149.9	1.2832	false

The experimental results showed the effectiveness of the proposed approach. We implement model checker in order to verify whether timed Kripke structure satisfies RTCTL formulas. These resulting data show that it can be verified whether or not lights up Total$_{led}$ times within the time constraint k. Moreover, as a result, we showed that the maximum number of LED lights up within the time constraint k. As shown in Table I, if n increases, the number of states and the number of transition relations polynomially increases. Therefore, the verification time and execution time polynomially increases. Further, when n becomes 234 or more, the verification result becomes false.

VII. CONCLUSION

In this paper, we have developed a verification system. Model checker after generating timed Kripke structure by using RTCTL formula verifies whether timed Kripke structure satisfies real-time properties.

In the case of RTCTL model checking after generating timed Kripke structure, when the timed Kripke structure satisfies E(p U q) and does not satisfy time constraint k, we can figure out the execution time of that path by calculation. By doing so, when we repeat the experiment, we can know the time constraint is nearly satisfied with that path.

As future work, we will develop model checking of embedded systems using counterexample-guided abstraction refinement (CEGAR). And we will compare with model checking while after generating all states.

Beside this, we want to detect other properties in embedded systems.

VIII. ACKNOWLEDGEMENTS

This work was supported by the Research Institute for Mathematical Sciences, a Joint Usage/Research Center located in Kyoto University.

References

[1] Ranjit Jhana, Rupak Majumdar, ”Software model checking,” ACM Computing Surveys (CSUR) 41 (4), 2009: 21.

[2] Leonardo de Moura, Nikolaj Bjorner, ”Z3: An Ecient SMT Solver,” LNCS 4963, pp. 337-340, 2008.

[3] S. Yamane, R. Konoshita, T. Kato, ”Model checking of embedded assembly program based on simulation,” IEICE Trans. Information and systems, Vol.E100-D, No.8, pp1819-1826, 2017.

[4] Schlich, B, ”Model Checking of Software for Microcontrollers,” ACM Transactions on Embedded Computing Systems 9 (4), 2010: 36

[5] E. A. Emerson, A. K. Mok, A. P. Sistla, J. Srinivasan, ”Quantitative Temporal Reasoning,” Real-Time Systems 4 (4) pp. 331-352, 1992.

[6] R. Alur, D. L. Dill: “A theory of timed automata,” TCS, 126(2), pp. 183-235, 1994.

[7] Edmund M. Clarke Jr., Orna Grumberg, Doron Peled, “Model Checking,” MIT Press, 1999.

[8] Corporation, R. E. : Renesas Electronics, Renesas Electronics Corporation (online), http://japan.renesas.com/

[9] nuvo WHEEL: ZMP, http://www.zmp.co.jp/products/wheel

[10] Klein, G.: JFlex - The Fast Scanner Generator for Java, CSE UNSW （online）, available from （http://jflex.de/）

[11] Jones, M. P.: Jacc: just another compiler compiler for Java, Department of Computer Science and Engineering at the OGI School of Science & Engineering at OHSU （online）, available from （http://jflex.de/）