

An Approach to Reducing Verification Complexity of Real-Time Embedded Systems

Luis Alejandro Cortés, Petru Eles, and Zebo Peng
 Dept. of Computer and Information Science
 Linköping University, Linköping, Sweden
 {luico,petel,zebpe}@ida.liu.se

Abstract

We present an approach to the formal verification of real-time embedded systems by using model checking. We address the verification of systems modeled in a timed Petri net representation and introduce a technique for reducing verification complexity. We translate the Petri net based model into timed automata and make use of available model checking tools to prove the correctness of the system with respect to design properties expressed in the temporal logics CTL and TCTL. Experimental results demonstrate considerable improvements in verification efficiency when the degree of parallelism of the system is considered.

1. Introduction

Embedded systems are part of larger systems and typically interact continuously with their environment. Such systems are often characterized by their dedicated function, real-time behavior, and high requirements in terms of correctness and reliability.

Correctness plays a key role in many embedded systems. The cost of a failure can be extremely high, in terms of loss of both human lives and money. Traditional validation techniques, like simulation and testing, are not sufficient to verify the correctness of such systems. Formal verification is becoming a practical way to ensure the correctness of designs by complementing simulation and testing.

In this paper we propose a technique for verifying systems represented in PRES+ [5]. PRES+ is a Petri net based model that can capture relevant characteristics of real-time embedded systems. In order to make use of existing model checking tools, we first translate the PRES+ model into timed automata and then model-check the resulting automata against required properties expressed in Computation Tree Logic (CTL) and Timed CTL (TCTL).

Previous work on verification via formal methods of PRES+ models include [5] and [6]. The former is a straightforward translation into a collection of timed automata, where one automaton is obtained for each transition. The latter extracts sequential parts of the Petri net representation in such a way that the resulting automata model is more efficient for verification.

The major contribution of this paper is the further reduction of verification time. Our method is based on a translation procedure from PRES+ into timed automata that significantly benefits the efficiency of the model checking process. The translation procedure is composed of three steps. First, we compute the concurrency relation of the underlying Petri net. This relation contains the pairs of transitions that can occur concurrently. Second, we form groups of transitions such that no two transitions in a group can fire

simultaneously, aiming at minimizing the number of such groups. Third, for each group, we construct the product automaton by composing the automata (which are obtained by the procedure defined in [5]) corresponding to the transitions in that particular group.

2. The Design Representation

The notation we use to model real-time embedded systems is an extension to Petri nets, called PRES+ (Petri Net based Representation for Embedded Systems). Such a formalism overcomes some of the drawbacks of uninterpreted Petri nets when modeling real-time embedded systems: for instance, PRES+ explicitly captures timing aspects and tokens carry information. This section briefly presents relevant characteristics of PRES+. The reader is referred to [5] for a formal definition of the model.

A PRES+ model is a five-tuple $N=(P, T, I, O, M_0)$ where P is a set of places, T is a set of transitions, I is a set of input (place-transition) arcs, O is a set of output (transition-place) arcs, and M_0 is the initial marking of the net. A marking is an assignment of tokens to the places of the net. A token is a pair $k=\langle v, r \rangle$ where v is the token value (may be of any type) and r is the token time (a non-negative real number). Thus tokens carry data and time information attached to them as stamps. In the model shown in Figure 1, p_a and p_b are the only places initially marked. For instance, the token $k_a=\langle v_a, r_a \rangle$ in place p_a has token value $v_a=3$ and token time $r_a=0$.

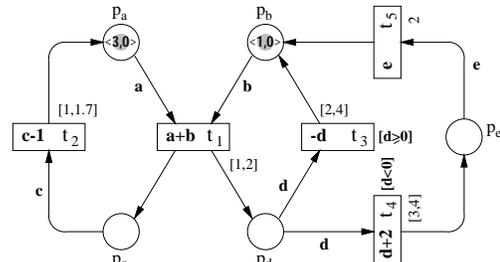


Figure 1. A PRES+ model

Every transition $t \in T$ has one transition function associated to it. Such a function takes as arguments the token values of tokens in the pre-set of the transition. The pre-set ${}^{\circ}t$ of a transition $t \in T$ is the set of input places of t . In Figure 1 we inscribe transition functions inside transition boxes: the function associated to t_1 , for example, is given by $f_1(a, b)=a+b$ where a and b are the token values of tokens in p_a and p_b respectively. We use inscriptions on the input arcs of a transition in order to denote the arguments of its transition function and/or those of its guard.

A transition $t \in T$ may have a guard, a condition that must be satisfied in order to enable the transition when all

its input places hold tokens. The guard of a transition is a predicate whose arguments are the values of tokens in the places of its pre-set. For instance, $d < 0$ is the guard of t_4 in the model of Figure 1. Note that when p_d gets a token, either t_3 or t_4 becomes enabled because their guards are complementary.

For every transition $t \in T$, there exist a *minimum transition delay* d^- and a *maximum transition delay* d^+ . The non-negative real numbers d^- and d^+ ($d^- \leq d^+$) represent the lower and upper bounds for the execution time (delay) of the function associated to the transition. Transition delays give the limits in time for the firing of a transition since it becomes enabled. When a transition fires, all tokens in its output places get the same value and time. For the initial marking in Figure 1, t_1 is the only transition enabled. It may not fire before 1 time units and must fire before or at 2 time units. Assuming t_1 fires at 2 time units, and accordingly tokens in p_a and p_b are removed and new tokens $k_c = \langle 4, 2 \rangle$ and $k_d = \langle 4, 2 \rangle$ are deposited in p_c and p_d , then both t_2 and t_3 become enabled at 2 time units but note that t_2 must fire strictly before t_3 , in the time interval $[3, 3.7]$.

3. Formal Verification of PRES+ Models

There are different types of analysis that can be performed on systems represented in PRES+: the absence/presence of tokens in places of the net, time stamps of such tokens, and their token values. Such analyses have been termed *reachability*, *time*, and *functional* analyses respectively. Our approach to verification focuses on the first two, that is, we reason about the marking of places in the net and token times. Thus, if the system model does not bear guards, we can ignore transition functions as reachability and time analyses will not be affected by token values.

For the sake of verification, we restrict ourselves to *safe* PRES+ nets, that is, models in which each place holds at most one token for every reachable marking. Otherwise, the formal analysis would become more cumbersome. This is a trade-off between expressiveness and analysis power.

We use formal methods in order to verify the correctness of real-time embedded systems modeled in PRES+. Model checking is a well-established approach to formal verification. It is an automatic procedure intended to determine whether the model of a system satisfies a set of required properties, usually expressed as temporal logic formulas.

Our approach allows to determine the truth of CTL (Computation Tree Logic) [4] and TCTL (Timed CTL) [1] formulas with respect to a (safe) PRES+ model. Formulas in CTL are composed of atomic propositions, boolean connectors, and temporal operators. CTL temporal operators consist of forward-time operators (**G** globally, **F** in the future, **X** next time, and **U** until) preceded by a path quantifier (**A** all computation paths, and **E** some computation path). TCTL is a real-time extension of CTL that permits to inscribe subscripts on the temporal operators to limit their scope in time. For example, the formula $\mathbf{AF}_{<n} Q$ expresses that, along all computation paths, the property Q is satisfied within n time units. In our methodology the atomic proposi-

tions of CTL/TCTL correspond to the marking of places in the net: the atomic proposition p holds iff $p \in P$ is marked.

There exist different tools for the analysis and verification of real-time systems based on the Timed Automata (TA) model, namely HyTech [9], KRONOS [12], and UPPAAL [13]. Such tools have been developed along many years and nowadays are quite mature and widely accepted within the real-time community. On the other hand, to the best of our knowledge, there are no tools that support TCTL model checking of timed Petri nets. In order to make use of available tools, we first translate PRES+ models into timed automata and then use one of the existing tools for model checking of TA.

A systematic procedure to translate PRES+ into timed automata was first defined in [5] (in the sequel this method will be referred to as *naive* translation), where the resulting model consists of a collection of automata that operate and coordinate with each other through shared variables and synchronization labels: one automaton with one clock variable is obtained for each transition of the Petri net. This approach, though, is not feasible for medium or large systems because the model checking of timed automata grows exponentially in the number of clocks.

An attempt to reduce the number of automata and clocks in the resulting TA model was introduced in [6]. An algorithm extracts sequential parts of the Petri net by *clustering* transitions. Intuitively, each *cluster* consists of a sequence of transitions where the firing of one of them *enables* the next one. The input of the algorithm is a safe Petri net and its output is a set of clusters (that form a partition of T , the set of transitions) each representing a sequential part of the net. Once clustering has been performed, timed automata are got by using a particular translation technique tuned for this specific approach, where one automaton with one clock is obtained per cluster (instead of one automaton and one clock per transition).

Since model checking of timed automata is exponential in the number of clocks, the translation into TA is crucial for our approach and must therefore try to minimize the number of resulting clocks. This paper presents a technique for model checking of real-time embedded systems represented in PRES+. This approach (called *coloring* as explained in the next section) not only reduces the number of automata/clocks but finds the minimum or near-minimum number of clocks necessary in the resulting TA. In this way the efficiency of verification is improved considerably.

4. Reduction of Verification Complexity by Coloring the Concurrency Relation

A major gain in verification efficiency would be achieved if we could get the “smallest” collection of automata as a result of the PRES+-to-TA translation procedure. This means that what would be verified is a model consisting of the minimum number of automata and clocks. This section introduces an approach that aims at finding such an optimal or near-optimal solution in terms of number of clocks/automata. The first step of this method is to find out

the pairs of transitions in the Petri net that may occur concurrently, that is those transitions that may fire at the same time for some reachable marking. Thus, for example, if we know that there is no reachable marking for which two given transitions may fire in parallel, then we can use *one* clock for accounting for the firing time semantics of *both* transitions because they will never fire simultaneously.

4.1. Computing the Concurrency Relation

The *concurrency relation* $\parallel \subseteq T \times T$ of an uninterpreted Petri net is the set of pairs (t_1, t_2) such that t_1, t_2 can fire concurrently for some reachable marking. In order to find those transitions in the PRES+ model that may fire in parallel, we take the underlying Petri net corresponding to the PRES+ model and compute its concurrency relation. For instance, in Figure 2 we show the model of a concurrent buffer [7] and its concurrency relation represented as a graph. The vertices of the graph are the transitions $t_i \in T$ and an edge joining two vertices indicates that the corresponding transitions can fire simultaneously.

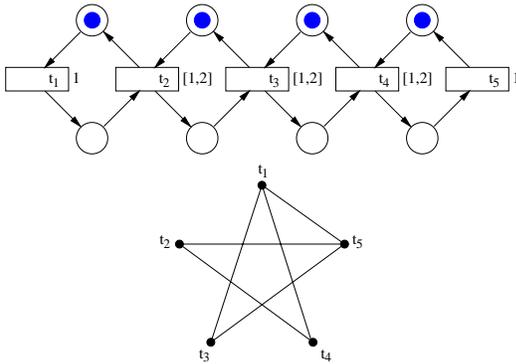


Figure 2. Buffer of capacity 4 and its concurrency relation

The problem of deciding if two given transitions of a Petri net may concurrently fire can be solved in polynomial time for *live* and *extended free-choice* nets [11]. It is important to note that extended-free choice is a structural property of the net and therefore easy to check, and that liveness of safe and extended-free choice nets is decidable in polynomial time [3].

We compute the concurrency relation of the live and extended free-choice Petri net by using an algorithm that, in general, gives better results than the one presented in [11], albeit both have a worst-case time complexity $O(n^3)$ where n is the number of places and transitions of the net. As illustrated by the experimental results in Section 5, obtaining the concurrency relation is computationally cheap.

4.2. Grouping Transitions

The naive way of grouping transitions is forming groups each consisting of a single transition [5]. However, we can do better by exploiting the information given by the concurrency relation. In Figure 2, for instance, t_2 and t_3 can be grouped together since we know that they will never fire concurrently. That means that the two timed automata corresponding to each transition may share the same clock vari-

able. Furthermore, it is possible to construct a single automaton (with one clock) equivalent to the behavior of both transitions.

We aim at obtaining as few groups of transitions as possible so that the automata equivalent to the PRES+ model have the minimum number of clocks. This problem is precisely MINIMUM GRAPH COLORING (MGC): given the concurrency relation as a graph $G=(T,E)$, find a *coloring* of T , i.e. a partitioning of T into disjoint sets T_1, \dots, T_k , such that each T_i is an independent set¹ for G and the size k of the coloring is minimum. This is known to be an NP-complete problem [8]. Nonetheless MGC is a very well-known problem and many approximation algorithms have been proposed as well as different heuristics that find near-optimal solutions. There are also algorithms that find the optimal coloring in reasonable time for some instances of the problem. For the particular example we use in Section 5, we are able to find the optimal solution in short time by using an algorithm based on Br elaz’s DSATUR [2].

From the point of view of our approach, we can study interesting trade-offs when coloring the concurrency relation. For instance, we can evaluate the results of letting some heuristics, e.g. simulated annealing, run longer and find a better solution in relation to the gain in verification time.

4.3. Composing Automata

We can reduce the number of resulting automata by composing those that correspond to transitions with the same color, after the concurrency relation has been colored. Thus we get one automaton with one clock for each color.

Automata are composed by applying the product construction [10]. In the general case, the product construction suffers from the so-called state-explosion problem, i.e. the number of locations of the product automaton is an exponential function of the number of components. However, in our approach we do not incur a blow-up in the number of states because the automata are tightly linked through synchronization labels and, most importantly, the composing automata are not concurrent. Recall that we do not construct the product automaton of the whole system. We construct one automaton per color, so that the composing automata (corresponding to that color) can not occur in parallel. In our experiments we have used an utility implemented in KRONOS [12] in order to compose timed automata.

5. Experimental Results

In order to illustrate our verification approach, we have model-checked a scalable example, comparing the technique based on a naive translation from PRES+ into automata introduced in [5], the clustering approach presented in [6], and the coloring-based method proposed in this paper.

The example that we use represents a number n of processes arranged in a ring configuration. The model for one such process is illustrated in Figure 3. Each one of the n processes in the system has a bounded response requirement, namely whenever the process starts it must strictly finish

¹An *independent set* is a subset $T_i \subseteq T$ such that no two vertices in T_i are joined by an edge in E .

within a time limit, in this case 25 time units. Referring to Figure 3, the start of one such process is denoted by the marking of p_{start} while the marking of p_{end} denotes the end of the process. This requirement is expressed by the TCTL formula $\mathbf{AG}(p_{start} \Rightarrow \mathbf{AF}_{<25} p_{end})$.

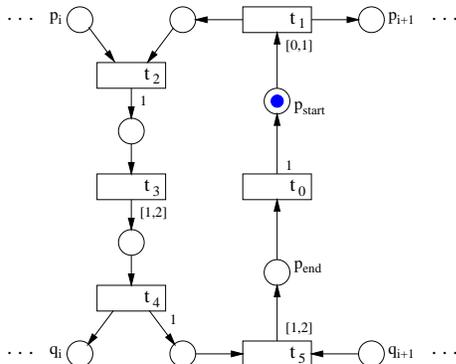


Figure 3. Model for one ring-configuration process

We have used UPPAAL [13], running on a Sun Ultra 10 workstation, in order to model-check the timing requirements of the processes in the ring-configuration example. The results are summarized in Table 1.

Table 1. Verification of the ring-configuration example

Num. Processes (n)	Verification Time [s]							
	Naive [5]	Clustering [6]	Coloring					Total Verification
			Comp. Conc. Relation	Coloring Conc. Relation	Product Automata	Model Checking		
2	0.078	0.054	0.001	0.001	0.071	0.049	0.122	
3	0.595	0.201	0.003	0.002	0.109	0.085	0.199	
4	8.252	2.071	0.006	0.005	0.142	0.493	0.646	
5	114.066	27.107	0.012	0.014	0.178	5.779	5.983	
6	1200.61	268.639	0.021	0.056	0.214	55.171	55.462	
7 [†]	18702.5	2309.61	0.032	0.185	0.249	464.596	465.062	
8 [†]	NA*	NA*	0.048	0.408	0.289	8341.44	8342.18	

[†] Specification does not hold

* Not available: out of time

The second column of Table 1 corresponds to the verification time using the approach of [5] (naive translation of PRES+ into timed automata). The third column in Table 1 shows the results of verification when using the clustering technique of [6]. When applying our approach, the time spent in computing the concurrency relation is given in the fourth column. The fifth column shows the execution time of the algorithm that finds the optimal coloring of the concurrency relation. The sixth column corresponds to the time spent in constructing the product automata. The model checking time of the resulting time automata is given in the seventh column. The last column of Table 1 shows the total verification time using the method presented in this paper.

Observe that for $n \geq 7$ the bounded response requirement expressed by the formula $\mathbf{AG}(p_{start} \Rightarrow \mathbf{AF}_{<25} p_{end})$ is not satisfied, a fact which is not obvious at all. An infor-

mal explanation is that since transition delays are given in terms of intervals, one process may take longer to execute than another; thus different processes can execute “out of phase” and this phase difference may be accumulated depending on the number of processes, causing one such process to take eventually longer than 25 time units (for $n \geq 7$). It is also worth mentioning that, although the model has relatively few transitions and places, this example is rather complex because of its large state space which is due to the high degree of parallelism.

6. Conclusions and Future Work

We have presented an approach to the problem of CTL/TCTL model checking of real-time embedded systems modeled in a Petri net based representation. Experimental results have shown that, by exploiting the levels of concurrency of the system, the complexity of verification can importantly be reduced.

As part of our future work, we intend to analyze the trade-offs when coloring the concurrency relation and their impact on the overall verification cost. For instance, when there are several optimal solutions to MINIMUM GRAPH COLORING, we plan to study how the choice of coloring influences the verification process.

References

- [1] R. Alur, C. Courcoubetis and D. L. Dill, “Model Checking for Real-Time Systems,” in *Proc. Symposium on Logic in Computer Science*, 1990, pp. 414-425.
- [2] D. Brélaz, “New Methods to Color the Vertices of a Graph,” in *Communications of the ACM*, vol. 22, pp. 251-256, April 1979.
- [3] A. Cheng, J. Esparza, and J. Palsberg, “Complexity results for 1-safe nets,” in *Theoretical Computer Science*, vol. 147, pp. 117-136, Aug. 1995.
- [4] E. M. Clarke, E. A. Emerson, and A. P. Sistla, “Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications,” in *ACM Trans. on Programming Languages and Systems*, vol. 8, pp. 244-263, April 1986.
- [5] L. A. Cortés, P. Eles, and Z. Peng, “Verification of Embedded Systems using a Petri Net based Representation,” in *Proc. ISSS*, 2000, pp. 149-155.
- [6] L. A. Cortés, P. Eles, and Z. Peng, “Verification of Real-Time Embedded Systems using Petri Net Models and Timed Automata,” in *Proc. RTCSA Conference*, 2002, pp. 191-199.
- [7] J. Esparza, “Model checking using net unfoldings,” in *Science of Computer Programming*, vol. 23, pp. 151-195, Dec. 1994.
- [8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: W.H. Freeman, 1979.
- [9] HyTech: The HYbrid TECHnology Tool, <http://www-cad.eecs.berkeley.edu/~tah/HyTech/>
- [10] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Boston, MA: Addison-Wesley, 2001.
- [11] A. Kovalyov and J. Esparza, “A polynomial algorithm to compute the concurrency relation of free-choice Signal Transition Graphs,” in *Proc. Intl. Workshop on Discrete Event Systems*, 1996, pp. 1-6.
- [12] KRONOS, <http://www-verimag.imag.fr/TEMPORISE/kronos/>
- [13] UPPAAL, <http://www.uppaal.com/>