Memory Efficient Data Structures for Explicit Verification of Timed Systems

Peter Gjøl Jensen, Kim Guldstrand Larsen, Jiří Srba,
Mathias Grund Sørensen, and Jakob Haar Taankvist

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark

Abstract. Timed analysis of real-time systems can be performed using continuous (symbolic) or discrete (explicit) techniques. The explicit state-space exploration can be considerably faster for models with moderately small constants, however, at the expense of high memory consumption. In the setting of timed-arc Petri nets, we explore new data structures for lowering the used memory: PTries for efficient storing of configurations and time darts for semi-symbolic description of the state-space. Both methods are implemented as a part of the tool TAPAAL and the experiments document at least one order of magnitude of memory savings while preserving comparable verification times.

1 Introduction

Semantics of real-time systems can be defined via real-valued time delays (continuous semantics) or integral time delays (discrete semantics). It is a folklore knowledge (see e.g. [2]) that both semantics coincide up to reachability as long as the formal model uses only closed (non-strict) clock guards. Continuous state-spaces are usually explored via zone-based abstractions (using the DBM data structure [8]), giving us a finite approximation of the model behaviour. Alternatively, we can explore the discrete state-space in an explicit manner, assuming a suitable extrapolation operator that guarantees termination of the search.

Explicit model checking is less studied even though it can successfully compete with zone-based methods on models without too large constants [3,12,10,1]. One of the main criticisms of explicit model checking is a high memory usage. We shall study possible solutions for saving memory in explicit model checking using BDDs [4], time darts data structure [10] and a new data-structure PTrie. We provide a realistic comparison of their performance on several case-studies from the literature. We base our study on the formal model of timed-arc Petri nets (TAPN) and the associated model checker TAPAAL [7] where the above mentioned techniques were implemented and made publicly available.

An example of a timed-arc Petri net, describing a researcher submitting papers for peer-reviewed conferences, is given in Figure 1. The net consists of places (circles), transitions (rectangles) and arcs (arrows). Places contain tokens, each with a real-time age, forming a marking of the net. In our example the initial

J.M. Badger and K.Y. Rozier (Eds.): NFM 2014, LNCS 8430, pp. 307–312, 2014.

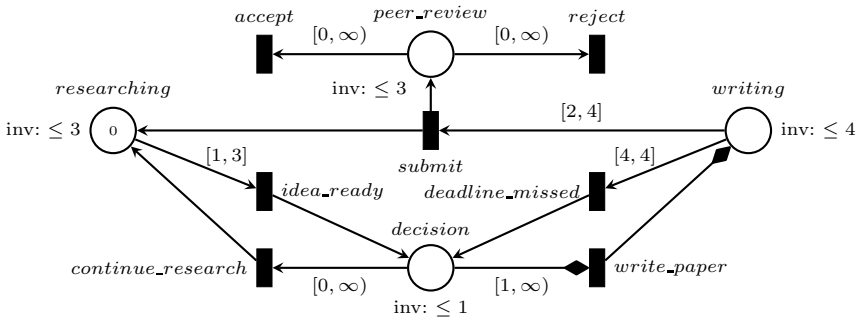

Fig. 1. A timed-arc Petri net describing a publication process

marking contains just one token of age 0 in the place *researching*. The place *researching* contains the age invariant ≤ 3, meaning that the token in this place cannot be older than 3. The net can so delay up to 3 time units (months) and once the age of the token is at least 1, the transition *idea_ready* gets enabled as the token's age fits into the interval [1,3]. The transition can now fire, consuming the token from the place *researching* and adding a new token of age 0 to the place *decision*. Now a decision whether to continue the research or write a paper must be taken within one month. If the researcher decides to write a paper, the token of age 1 is moved from the place *decision* to the place *writing* while preserving its age 1 due to the use of transport arcs (with diamond-shaped arrow-tips). Hence the time of the decision counts into the total number of months used for writing the paper. After writing for at least 1 month, the researcher can submit the paper while producing two new tokens of age 0 into the places *researching* and *peer_review*. By repeating the process, it is possible to have two publications under peer-review at the same time, though the timing constraints imply that having submitted three publications concurrently is impossible. As the net is bounded, this can be verified using the model checker TAPAAL [7] that supports also other primitives like weights, inhibitor arcs, urgent transitions, constants and components with interfaces.

2 PTries and Time Darts

The basic reachability algorithm based on explicit state-space search is given in Figure 2 where φ is a propositional formula over the number of tokens in the places of the net, and $M \xrightarrow{t} M'$ and $M \xrightarrow{1} M'$ represent transition firing resp. a delay of one time unit. The function *cut* is an extrapolation of token ages that exceed their maximum relevant bounds, yielding a canonical representative for each marking. This guarantees finiteness of the state-space for bounded nets (see [1]). A fragment of the state-space for our running example is shown in Figure 2; here e.g. $(r, 0)$ stands for a token of age 0 in place *researching*.

The algorithm utilizes two data structures, the *passed* and *waiting* sets that store the discovered state-space. The size of these sets (in particular the *passed*

Input: A closed TAPN N with initial marking M_0, proposition φ and a bound k.
Output: True if $M_0 \rightarrow^* M$ via k-bounded markings and $M \models \varphi$, False otherwise.

Passed := ∅; *Waiting* := ∅;
AddToPW(M_0);
while *Waiting* ≠ ∅ **do**
  Remove M from *Waiting*;
  Passed := *Passed* ∪ {M};
  foreach $M \xrightarrow{t} M'$ or $M \xrightarrow{1} M'$
  do AddToPW($cut(M')$)
return False;

AddToPW(M): **begin**
  if $M \notin$ *Passed* ∪ *Waiting* ∧
  $size(M) \leq k$ **then**
    if $M \models \varphi$ **then**
      return True and **exit**;
    Waiting := *Waiting* ∪ {M};

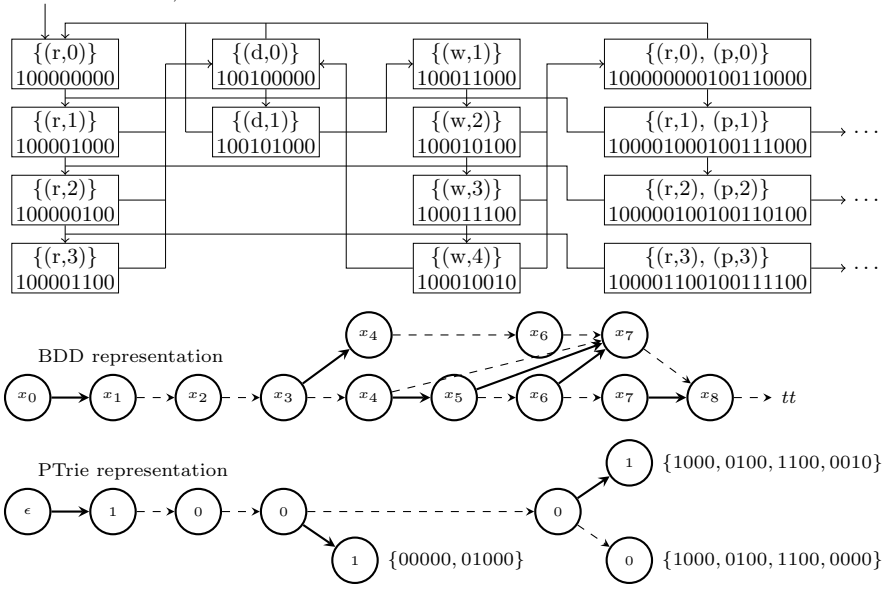

Fig. 2. Explicit reachability algorithm and the initial fragment of the state-space for the net from Figure 1. Places are abbreviated by their first letters and the full state-space contains 29 markings. The markings include their binary encodings and the first three columns of the state-space are stored using BDD and PTrie.

one) can be large and we need an efficient way to store them. For this purpose, we represent each marking as a binary string (in an arbitrary but fixed manner) as shown in Figure 2. An obvious way to store the binary encodings of markings is using BDDs. However, the repeated additions to the sets make this approach inefficient as BDDs are normalized after each insertion. We suggest instead a new data structure Partial Trie or *PTrie* (based on Trie [9]) that stores the binary values in the path through the decision tree rather than in the nodes. As the cost of storing a single node in the PTrie exceeds one bit, using a fully unfolded tree may not necessarily preserve any memory. For this purpose we introduce buckets at different levels of the tree that contain suffixes of the binary encodings after the initial prefix that is encoded in the path leading to the bucket, as demonstrated

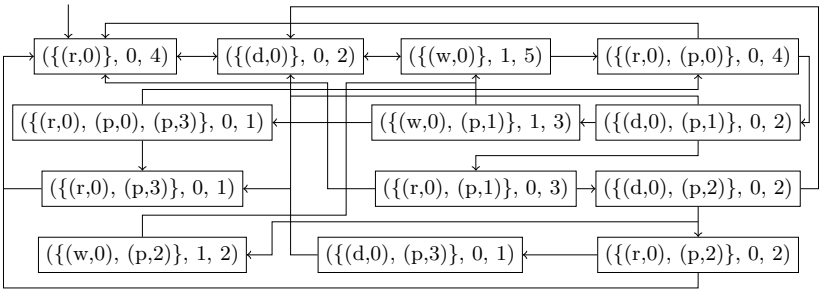

Fig. 3. The state-space of the net from Figure 1 represented with time darts

in Figure 2. As soon as the number of strings with the same most-significant bit in a bucket exceeds a predefined constant (3 in our example), the node splits.

In the explicit state-space in Figure 2 we can notice that the markings in each column are simply delays of the top most marking. The time dart data structure, first suggested in [10] for timed automata, exploits this fact by representing all such markings in a single dart. A base marking contains at least one token of age 0. For any marking M' there is a unique base marking M s.t. $M \xrightarrow{d} M'$ for some $d \geq 0$. A *time dart* is a triple (M, w, p) where M is a base marking, $w \in \mathbb{N}_0$ is a waiting distance and $p \in \mathbb{N}_0^\infty$ is a passed distance such that $w \leq p$. The dart simultaneously represents the set of waiting markings M' such that $M \xrightarrow{d} M'$ where $w \leq d < p$, and the set of passed markings M' such that $M \xrightarrow{d} M'$ where $d \geq p$. Figure 3 depicts the full state-space of time darts for our running example. It contains only 13 darts compared to 29 markings in the explicit state-space.

3 Experiments

We report on four case studies of Alternating Bit Protocol [13] (ABP), Business Activity with Participant Completion [11] (BAwPC), Patient Monitoring System [6] (PMS) and MPEG-2 video encoder [14]. The reachability queries for all models require a complete state-space search. All TAPAAL models are available at `http://www.tapaal.net` and the experimental data can be reproduced by using TAPAAL 2.4.1. The experiments (run on a Macbook Pro 2.7GHz Intel Core i7) were terminated once the memory usage exceeded 6GB (OOM) or the verification took longer than one hour (🕑); for the BDD-based engine we allowed a two-hour timeout. The TAPAAL column refers to the verification performed by the DBM-based continuous engine of TAPAAL, while the UPPAAL column reports on the best automatic translation [5] from TAPAAL to UPPAAL timed automata. The remaining columns deal with explicit state-space exploration using BDDs, combination of time-darts with PTries, time darts and PTries separately, and no memory optimization (Basic); all efficiently implemented in C++.

The aim of this paper is not to compare the zone-based vs. explicit methods as this largely depends on the concrete models (see e.g. [2,3,12])—we see that while the explicit methods are faster on the first three models, the zone-based

	Zone-based		Explicit				
Scale	TAPAAL	UPPAAL	BDD	Darts+PTries	Darts	PTries	Basic
Alternating Bit Protocol (ABP), scaled by the number of messages							
15	116.8 s 278 MB	4.0 s 32 MB	2701.2 s 8 MB	3.1 s 6 MB	2.4 s 23 MB	7.6 s 12 MB	5.4 s 77 MB
16	328.4 s 501 MB	5.6 s 34 MB	4057.7 s 9 MB	3.9 s 7 MB	3.1 s 34 MB	9.7 s 15 MB	7.1 s 105 MB
17	1233.7 s 979 MB	7.6 s 41 MB	5929.2 s 10 MB	5.0 s 8 MB	3.9 s 34 MB	12.4 s 18 MB	9.0 s 132 MB
30	◷	182.2 s 288 MB	◷	46.3 s 53 MB	38.8 s 377 MB	120.2 s 139 MB	91.0 s 1107 MB
40	◷	1063.0 s 920 MB	◷	151.2 s 155 MB	120.8 s 1088 MB	390.3 s 410 MB	288.6 s 3465 MB
50	◷	◷	◷	378.2 s 362 MB	298.3 s 2593 MB	1018.4 s 962 MB	OOM
Business Activity Protocol (BAwPC), scaled by the number of messages							
2	2.9 s 30 MB	7.8 s 9 MB	635.0 s 23 MB	4.4 s 35 MB	6.8 s 50 MB	8.3 s 10 MB	7.5 s 66 MB
3	14.3 s 114 MB	19.6 s 55 MB	4529.6 s 57 MB	25.0 s 23 MB	22.2 s 162 MB	26.2 s 25 MB	24.1 s 197 MB
4	60.4 s 392 MB	84.0 s 116 MB	◷	66.7 s 52 MB	62.9 s 415 MB	71.5 s 55 MB	67.5 s 502 MB
8	OOM	◷	◷	861.9 s 501 MB	770.8 s 4934 MB	918.3 s 490 MB	846.5 s 5198 MB
Patient Monitoring System (PMS), scaled by the sampling frequency							
18	22.2 s 52 MB	◷	158.0 s 5 MB	0.9 s 3 MB	0.6 s 18 MB	0.8 s 4 MB	0.4 s 19 MB
12	399.6 s 215 MB	◷	578.5 s 9 MB	3.0 s 8 MB	2.0 s 51 MB	2.5 s 7 MB	1.4 s 54 MB
10	2315.3 s 635 MB	◷	1537.8 s 17 MB	7.4 s 15 MB	5.0 s 122 MB	6.4 s 16 MB	3.6 s 112 MB
6	◷	◷	◷	94.4 s 149 MB	68.0 s 1482 MB	82.5 s 154 MB	51.6 s 1665 MB
5	◷	◷	◷	671.9 s 826 MB	OOM	575.6 s 815 MB	OOM
MPEG2 Encoder (MPEG2), scaled by the number of B frames							
3	0.1 s 2 MB	0.1 s 10 MB	◷	0.6 s 4 MB	0.3 s 13 MB	19.9 s 117 MB	19.7 s 665 MB
4	0.1 s 2 MB	0.2 s 14 MB	◷	5.2 s 22 MB	3.7 s 95 MB	156.7 s 880 MB	165.3 s 4811 MB
5	0.1 s 3 MB	0.2 s 18 MB	◷	46.7 s 147 MB	40.4 s 870 MB	1104.8 s 5162 MB	OOM

methods will eventually outperform any explicit search on models with large enough constants as demonstrated on MPEG2 (the constants here are in thousands of nanoseconds). We instead aim at comparing the memory performance of the different data structures. The first observation is that the BDD encoding, while memory efficient, has an unacceptable runtime performance. On the other hand PTries cause only 20-30% slowdown compared to the basic algorithm and still provide similar memory savings as BDDs. The time dart method usually provides both time and memory improvements; this is most visible on models with large constants like in MPEG2. The combination of PTries and time darts gives the largest memory savings with a very acceptable performance.

4 Conclusion

The general data structure PTrie provides significant memory savings at marginal runtime overhead and can be directly employed by any explicit model checker. The semi-symbolic method of time darts requires a more substantial adaptation but it gives in general both time and memory improvements, especially for models with larger constants. Both data structures have been implemented within an open source, publicly available tool TAPAAL, and show promissing experimential results.

References

1. Andersen, M., Gatten Larsen, H., Srba, J., Grund Sørensen, M., Haahr Taankvist, J.: Verification of liveness properties on closed timed-arc Petri nets. In: Kučera, A., Henzinger, T.A., Nešetřil, J., Vojnar, T., Antoš, D. (eds.) MEMICS 2012. LNCS, vol. 7721, pp. 69–81. Springer, Heidelberg (2013)
2. Asarin, E., Maler, O., Pnueli, A.: On discretization of delays in timed automata and digital circuits. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 470–484. Springer, Heidelberg (1998)
3. Bozga, M., Maler, O., Tripakis, S.: Efficient verification of timed automata using dense and discrete time semantics. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 125–141. Springer, Heidelberg (1999)
4. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
5. Byg, J., Jacobsen, M., Jacobsen, L., Jørgensen, K.Y., Møller, M.H., Srba, J.: TCTL-preserving translations from timed-arc Petri nets to networks of timed automata. In: TCS (2013), http://dx.doi.org/10.1016/j.tcs.2013.07.011
6. Cicirelli, F., Furfaro, A., Nigro, L.: Model checking time-dependent system specifications using time stream Petri nets and UPPAAL. Applied Mathematics and Computation 218(16), 8160–8186 (2012)
7. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012)
8. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
9. Fredkin, E.: Trie memory. Communications of the ACM 3(9), 490–499 (1960)
10. Jørgensen, K.Y., Larsen, K.G., Srba, J.: Time-darts: A data structure for verification of closed timed automata. In: SSV 2012. EPTCS, vol. 102, pp. 141–155. Open Publishing Association (2012)
11. Marques Jr., A.P., Ravn, A.P., Srba, J., Vighio, S.: Model-checking web services business activity protocols. International Journal on Software Tools for Technology Transfer (STTT) 15(2), 125–147 (2013)
12. Lamport, L.: Real-time model checking is really simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
13. Lynch, W.C.: Computer systems: Reliable full-duplex file transmission over half-duplex telephone line. Communications of the ACM 11, 407–410 (1968)
14. Pelayo, F.L., Cuartero, F., Valero, V., Macia, H., Pelayo, M.L.: Applying timed-arc Petri nets to improve the performance of the MPEG-2 encoding algorithm. In: MMM 2004, pp. 49–56. IEEE Computer Society (2004)