

# Finite and Infinite Model Checking of Dual Transition Petri Net Models

Mauricio Varea, Bashir Al-Hashimi and Michael Leuschel

Department of Electronics and Computer Science  
University of Southampton, SO17 1BJ, UK  
{ m.varea , bmah , mal }@ecs.soton.ac.uk

## [Extended Abstract]

The formal verification of embedded systems is becoming a key research area due to the ever increasing design complexity involved in the modelling and validation of embedded systems. Traditional methods of validation, such as simulation and testing, are practically becoming an infeasible solution for large design models. Nowadays, only a small part of a real-life embedded system's state space can be explored by such traditional methods and, therefore, alternative ways of reasoning about the correctness of an embedded system are rapidly gaining popularity in hardware [1] and software [2] verification. The model checking [3] approach has been successfully applied to the verification of finite state concurrent systems, and is becoming of much interest in both industry and academia. This approach automatically verifies a system model, given a set of properties expressed in temporal logics [4], *e.g.* Computation Tree Logic (CTL) or Linear Temporal Logic (LTL) [5].

The specification of an embedded system may be well represented in Petri Net (PN) based models, which are capable of exploiting some desired features of the design, *e.g.* concurrency. The purpose of a model checking algorithm here, is to formally reason about behavioural properties of embedded systems described in terms of PN based models. An approach aimed to reduce the complexity of reachability trees in Petri nets, by means of BDDs, has been presented in [6, 7]. This verification methodology applies symbolic techniques to the encoding of the states in a concurrent formalism. PRES+ is another Petri net oriented model aimed to represent embedded systems, which has been applied to formal verification [8] of Timed CTL (TCTL) properties. In order to cope with verification, PRES+ models are transformed into Timed Automata. We had recently proposed a new PN based model which efficiently captures both control and data flow structure from a behavioural description of an embedded system [9]. This model, namely Dual Transition Petri Net (DTPN), is based on a PN structure and it is aimed to exploit the linkage between control and data flow in an embedded system specification, leading to better implementation results.

In this work, we tackle the model checking of the recently introduced DTPN models [10] and extend this approach to also consider infinite state models. Our aim is to propose an homogeneous approach to undertake the problem of formal verification in an heterogeneous design model, *e.g.* embedded systems. One form of heterogeneity present in embedded systems is due to the existence of two separate but related parts, *i.e.* control and data flow. In order to capture the intrinsic structural and behavioural characteristics of embedded systems, this work exploits the features related to the coexistence of both control and data flow in embedded systems specification.

## Dual Transition Petri Nets

An homogeneous formalism which allows heterogeneity among the functionality of its elements is of relevance to embedded system design, because different parts in the design can be captured by diverse semantics of the model (*i.e.* heterogeneous) and still be possible to reason about properties which are of the same nature (*i.e.* homogeneous) and relate to those various parts. Therefore, Dual Transition Petri Net (DTPN) is an extension of classical PNs which explicitly models both control and data parts of an embedded system design, by using a supplementary set  $Q$  to represent data operation in addition to the classical set of transitions  $T$  used for control purposes.

The structure of a DTPN model  $\mathcal{N}$  is composed of a *finite* set of places ( $P$ ), a set of *control* transitions ( $T$ ), a set of *data* transitions ( $Q$ ), two sets of arcs ( $F_C, F_D$ ), a weight function  $W$ , and a guard function  $G$ . This modified structure requires extending the classical definition of the marking function  $\mu$ , in order to allow a state representation which captures not only the control information of the system, but also its data flow. Therefore a marking function is defined in the domain of the complex numbers as shown in Definition 1.

**Definition 1** A marking function ( $\mu$ ) is a mapping from the set of places  $P$  to the set of complex numbers  $\mathbb{C}$ . This is:

$$\mu : P \mapsto \mathbb{C}$$

And, for each element  $m_i = \mu(p_i)$ , the modulus and phase of the underlying complex number represents a quantum in the data domain ( $\alpha_i$ ) and is proportional to the number of tokens in the control domain ( $\gamma_i$ ), respectively.

$$m_i = \alpha_i \cdot e^{i \frac{\pi}{2} \gamma_i}$$

Where  $\alpha_i \in \mathbb{Z}, \gamma_i \in \mathbb{N}, 1 < i \leq |P|$

Although the marking function is defined as a mapping into a dense domain, *i.e.* the complex number domain  $\mathbb{C}$ , it should be noted that only the discrete values determined by the pair  $\langle \alpha_i, \gamma_i \rangle$  are considered. The reachability tree  $\mathcal{RT}$  generated from a DTPN model can be either finite or infinite, depending on the structure of the net. When the reachability set (*i.e.* the set of all labels  $\in \mathcal{RT}$ ) is finite, we apply the Cadence SMV tool [11] in order to reason about properties of the DTPN model [10]. However, paradigms such as *recursion, dynamic* or *unbounded data structures* leads to an infinite number of states in the reachability set, which must indeed be covered by a model checking tool. Therefore, we also propose an extension of the approach presented in [10] by means of automated logic programming techniques which can verify infinite state models by abstraction and specialisation [12].

## Finite State Model Checking

Since the structure of DTPN models is based on two set of transitions ( $T, Q$ ), as opposed to classical PNs which have only one set of transition ( $T$ ), common model checking techniques investigated for PN based models [6, 8] are not applicable to the problem of formal verification in DTPNs. In DTPN one set of transitions ( $T$ ) is used for the control domain while the other set ( $Q$ ) is used for the data domain, hence the model checking of such a model is of highly heterogeneous nature. For instance, the verification of a control transition  $t \in T$  might involve checking that it will only fire given a certain condition while the verification of a data transition  $q \in Q$  might entail an arithmetic operation.

In our approach [10], the finite state DTPN formal verification is carried out through SMV modules, which can cope with the heterogeneity aforementioned by means of a inter-module communication structure which resembles a *multiplexor*, as shown in Figure 1. This multiplexor-like structure allows a wisely interaction of both control and data transition modules. Places ( $P$ ) of the DTPN structure are represented by an array which can contain both control and data information of the embedded system. On the one hand, an instantiation of the control transition module directly affects one part of the array of places, *i.e.* control domain, which deals with tokens as in classical PNs. On the other, an instantiation of the data transition module takes into account the effects on the data domain of the embedded system modelled.

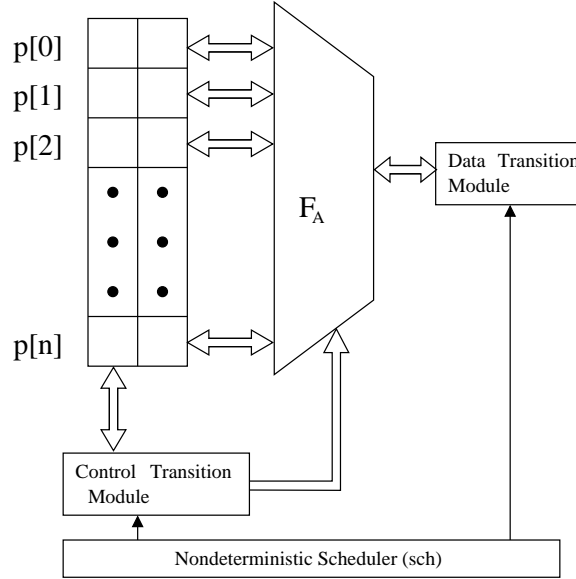


Figure 1: Communication among SMV modules in DTPN model checking

A nondeterministic scheduler synchronises both modules and an implicit structure  $F_A$  grant the transformation in the data domain *only* where it is necessary, thus reducing verification complexity. Figure 2 illustrates the SMV code added in the main module to perform such a synchronisation. Since SMV follows a *single assignment rule*, the set  $t_j^\bullet$ ,  $1 < j \leq |T|$ , is first assigned individually to an auxiliary variable within the control transition module, *i.e.*  $t[] . pp$ , and later altogether assigned to the array  $p[]$ .

```

typedef TRANSITIONS 1..(M);
...
sch : TRANSITIONS;
sch := { i : i = 1..(M), t[i].en };
...
forall (i in PLACES)
  next (p[i].modulus) := t[sch].pp[i];

```

Figure 2: Nondeterministic scheduler

The proposed methodology has been applied to a number of embedded system specifications in order to demonstrate its validity. Some examples have been presented in [10], illustrating the DTPN scalability to real-life examples and showing that the proposed methodology has got more affinity to *reactive* rather than *transformational* embedded systems. One of the advantages of using

the Cadence SMV tool to verify finite states DTPN models, is the actual support of both LTL and CTL properties.

## Infinite State Model Checking

Since the temporal logic properties verified in a model checking approach resemble a denotational program semantics, it turns out that (Constraint) Logic Programming (CLP) [13] is a natural approach to model checking both reactive and transformational embedded system models expressed in DTPN. Therefore, the combination of model checking techniques with CLP programs [14, 15] is a promising idea to extend model checking capabilities into infinite state systems [16].

Abstraction is a key issue in infinite model checking. When reasoning about finite labelled transition systems (LTS) which have an infinite number of states, some abstraction techniques, such as program specialisation [12] or partial deduction [17, 18], have to be used, in order to approximate the infinite state system by a (simpler) finite state model.

Unlike [10], where DTPN models are bounded, this approach allows the model checking of unbounded DTPN models, where the number of tokens may increase as much as necessary, even towards *infinity*. This has a great applicability in software verification.

Furthermore, the approach proposed in this paper has also shown better results than [10] for  $k$ -bounded DTPNs with  $k > 1$ , since CLP is “naturally” designed to handle *constraints*, such as the enabling conditions presented in [9].

## References

- [1] Christopher Kern and Mark R. Greenstreet. Formal Verification in Hardware Design: A Survey. *ACM Transaction on Design Automation of Embedded Systems*, 4(2):1–67, April 1999.
- [2] John Dennis Gannon, James M. Purtillo, and M. V. Zelkowitz. *Software Specification: A Comparison of Formal Methods*. Ablex, Norwood, NJ, 1994.
- [3] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.
- [4] E. Allen Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 995–1072. Elsevier Science Publishers, B.V., 1990.
- [5] P. Bellini, R. Mattolini, and P. Nesi. Temporal Logics for Real-Time System Specification. *ACM Computing Surveys*, 32(1):12–42, March 2000.
- [6] Jordi Cortadella. Combining structural and symbolic methods for the verification of concurrent systems. In *Int. Conf. on Application of Concurrency to System Design*, pages 2–7, March 1998.
- [7] Enric Pastor, Jordi Cortadella, and Marco A. Peña. Structural Methods to Improve the Symbolic Analysis of Petri Nets. In *Proceedings of the 20<sup>th</sup> International Conference on Applications and Theory of Petri Nets*, volume 1639 of *Lecture Notes in Computer Science*, pages 26–45. Springer-Verlag, June 1999.
- [8] Luis Alejandro Cortés, Petru Eles, and Zebo Peng. Verification of Embedded Systems using a Petri Net based Representation. In *Proceedings of the 13<sup>th</sup> International Symposium on System Level Synthesis (ISSS)*, pages 149–155, Madrid, Spain, 20-22 September 2000.
- [9] Mauricio Varea and Bashir Al-Hashimi. Dual Transitions Petri Net based Modelling Technique for Embedded Systems Specification. In *Proceedings of the 4<sup>th</sup> Proc. Design, Automation and Test in Europe (DATE)*, pages 566–71, Munich, Germany, March 2001. IEEE/ACM.
- [10] Mauricio Varea, Bashir M. Al-Hashimi, Luis A. Cortés, Petru Eles, and Zebo Peng. Symbolic Model Checking of Dual Transition Petri Nets. In *Proceedings of the 10<sup>th</sup> International Workshop on Hardware/Software Codesign (CODES/CASHE)*, Colorado, USA, 6-8 May 2002. Accepted for publication.

- [11] The SMV Model Checker.  
<http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/>.
- [12] Michael Leuschel and Thierry Massart. Infinite State Model Checking by Abstract Interpretation and Program Specialisation. In Annalisa Bossi, editor, *Logic-Based Program Synthesis and Transformation*, LNCS 1817, pages 63–82, Venice, Italy, September 1999.
- [13] Joxan Jaffar and Michael Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
- [14] Giorgio Delzanno and Andreas Podelski. Model Checking in CLP. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 223–239, 1999.
- [15] Laurent Fribourg. Constraint logic programming applied to model checking. In *Proceedings of the 9<sup>th</sup> International Workshop on Logic-based Program Synthesis and Transformation (LOPSTR'99)*, volume 1817, pages 31–42, Venezia, Italy, September 1999. Springer-Verlag.
- [16] E. Allen Emerson and Kedar S. Namjoshi. On Model Checking for Non-Deterministic Infinite-State Systems. *LICS*, pages 70–80, 1998.
- [17] Michael Leuschel and Helko Lehmann. Solving Coverability Problems of Petri Nets by Partial Deduction. In Maurizio Gabbrielli and Frank Pfenning, editors, *Proceedings of the PDP'2000*, pages 268–279, Montreal, Canada, 2000. ACM Press.
- [18] Michael Leuschel and Helko Lehmann. Coverability of Reset Petri Nets and other Well-Structured Transition Systems by Partial Deduction. In John Lloyd, editor, *Proceedings of the International Conference on Computational Logic (CL)*, LNAI 1861, pages 101–115, London, UK, 2000. Springer-Verlag.