Electronic Communications of the EASST
Volume 68 (2014)

Proceedings of the 8th International Workshop on Graph-Based Tools (GraBaTs 2014)

Towards Model Checking Reconfigurable Petri Nets using Maude

Alexander Schulz, Julia Padberg

14 pages

Guest Editors: Matthias Tichy, Bernhard Westfechtel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards Model Checking Reconfigurable Petri Nets using Maude

Alexander Schulz, Julia Padberg

Hamburg University of Applied Sciences, Germany
julia.padberg@haw-hamburg.de, alexander.schulz1@haw-hamburg.de

Abstract: This paper introduces an approach to model checking of reconfigurable Petri nets. The main task is to flatten the two levels of dynamic behavior that reconfigurable nets provide, the firing of transitions on the one hand and the transformation of the nets on the other hand. We show how to translate a reconfigurable net into Maude modules. Maude's LTL model-checker is then used to verify properties of these modules.

Keywords: Reconfigurable Petri nets, ReConNet, Maude, model checking, LTL

1 Motivation

Software systems are increasingly characterized by dynamic structures that require execution and reconfiguration at run-time to adjust the system's behavior to its changing environment. Their main feature results from their complex coordination behavior within dynamically adapting infrastructures. Such dynamic structures need a suitable formal description technique that allows the separation at different levels of dynamic behavior.

Reconfigurable Petri nets provide dynamic changes at the process level, (as typical for Petri nets) and additionally at the structure level. They are based on the algebraic approach to Petri nets, with operations describing the pre- and post-domain of transitions as introduced in [MM90] and are equipped with rules for the transformation of the net. These rules allow the modification of the net's structure at run time. Reconfigurable Petri nets form a family (e.g., in [EP03, LO04, EHP+07, PEHP08, KCD10, Mod12, Gab14]) depending on the underlying kind of Petri net and have been applied in various application areas where complex coordination and structural adaptation at run-time is required (e.g., mobile ad-hoc networks, communication spaces, ubiquitous computing, workflows in a dynamic infrastructure). The distinction between the net behavior and the dynamic change of its net structure is the characteristic feature that makes reconfigurable Petri nets so suitable for systems with dynamic structures.

Dynamic changes at two levels leads to a very complex behavior that is hard to understand, hence verification support is crucial. In this paper we suggest a first approach to verification of reconfigurable nets based on model checking. Model checking is a powerful verification technique to improve the quality of the software system. Basically, it is a systematic check of specified properties in all reachable states of the system's behavioral model. By now it is a highly effective verification technology, that is widely adopted in the hardware and software industries. Petri nets with a collective token approach can be easily be described in terms of theories and theory morphisms in partial membership equational logic, see ,e.g.,[BMMS98]. There are some experiences of translating Petri nets to Maude, as in [SMÖ01, AS02, BBR+11].

The paper is organized as follows: First we define reconfigurable Petri nets based on decorated place/transition nets and introduce the tool ReConNet. In the next section we outline rewriting in Maude and the subsequent section is concerned with the translation from reconfigurable nets to Maude and the use of its LTL model checker. Section 5 concerns related work, there we discuss the relation to graph transformation. Future work is discussed in the concluding remarks.

2 ReConNet: a Simulation Tool for Reconfigurable Petri Nets

The tool ReConNet (see [PEOH12]) has been developed at the HAW Hamburg and supports the modeling and simulation of reconfigurable nets. An intuitive graphic-based user interface allows the user to create, modify and simulate reconfigurable nets. The net and rule in Fig. 1

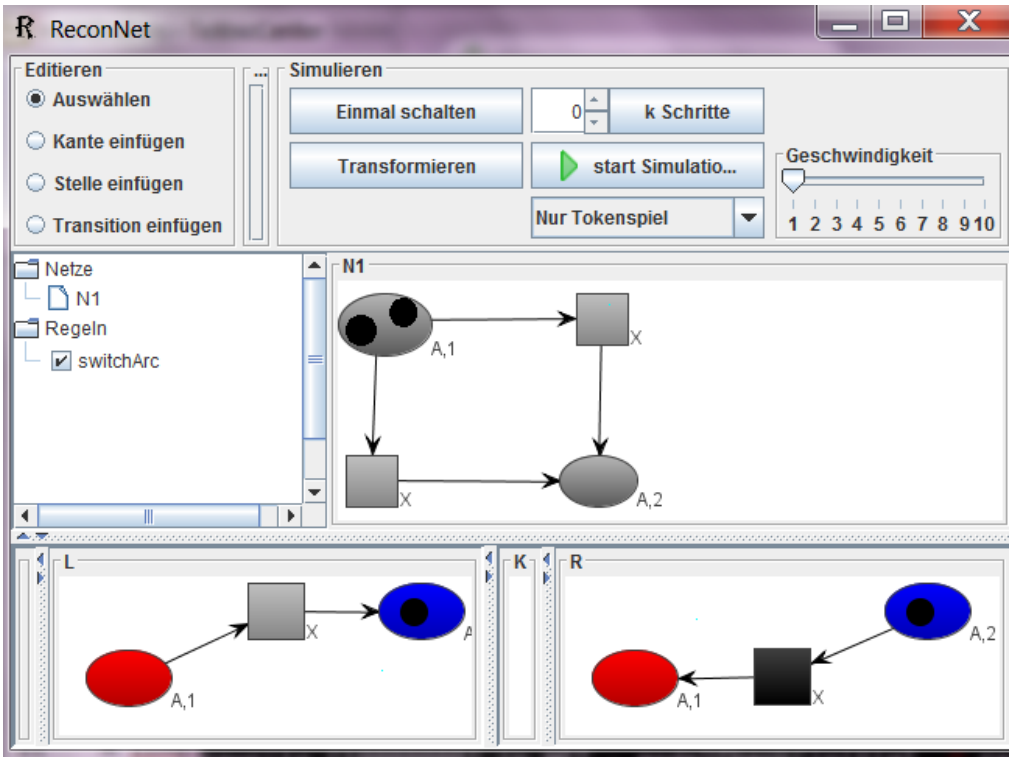


Figure 1: Reconfigurable net[1] $rN = (N1, \{switchArc\})$ with net $N1$ and the rule *switchArc*

have been edited and computed by ReConNet. The simulation of nets can be done in different modes for firing transitions only, for applying rules only and for both. There are two more advanced simulation options - steps and continuous simulation. The steps option executes a definable amount of steps (firing transitions or applying rules) on the net. The simulation option allows running the net non-deterministically until it is stopped by the user. The following example is a purely theoretical one, unfortunately the space limitations do not allow a larger example, e.g., from [Rei12].

[1] For readability the identities have been given explicitly, so place $A,1$ has the name A and the id 1.

In Fig. 1 we state not only the name of a place A, but the identity of the place explicitly as well. In net $N1$ (see Fig. 1) the transitions with name X can fire twice yielding two tokens on the place $A,2$. But then there is a deadlock. In contrast the reconfigurable net $rN = (N1, \{switchArc\})$ consists of the net $N1$ together with the rule *switchArc*, that replaces a transition by one with switched arcs. In this case either the transition with name X fires or the rule *switchArc* is applied, yielding $N1 \overset{switchArc}{\Longrightarrow} N2$. This transformation deletes one of the transitions and adds a transition, also with the name X, but with the arcs going in the opposite direction. Hence, the resulting net $N2$ is cyclic and live. So, there exists no longer a deadlock, this will be confirmed in Section 4 by the Maude's LTL model-checker.

Decorated nets, as supported by ReConNet belong to the algebraic approach to Petri nets, so they have pre- and post-domain functions $pre, post : T \to P^{\oplus}$ and a marking $M \in P^{\oplus}$, where $P^{\oplus}$ is the multiset of of places, formally defined by a free commutative monoid. A transition $t \in T$ is M-enabled for a marking $M \in P^{\oplus}$ if we have $pre(t) \leq M$, and in this case the follower marking M' is given by $M' = M \ominus pre(t) \oplus post(t)$ and $M[t\rangle M'$ is called firing step. In [Pad12] new features have been added to gain an adequate modeling technique. The extension to capacities and names is quite obvious. More interesting are the transition labels that may change, when the transition is fired. This allows a better coordination of transition firing and rule application, for example can be ensured that a transition has fired (repeatedly) before a transformation may take place. This last extension is conservative with respect to Petri nets as it does not change the net behavior. A decorated place/transition net is a marked P/T net $N = (P, T, pre, post, M)$ together with a capacity as a function $cap : P \to \mathbb{N}^{\omega}$, A_P, A_T name spaces with $pname : P \to A_P$ and $tname : T \to A_T$ the function $tlb : T \to W$ mapping transitions to transition labels W and the function $rnw : T \to END$ where END is a set containing some endomorphisms on W, so that $rnw(t) : W \to W$ is the function that renews the transition label. For the sake of clarity we have omitted the decorations up to the names.

For decorated place/transition nets as given above, we obtain with a suitable notion of morphisms an $\mathcal{M}$-adhesive HLR category (see [Pad12]). $\mathcal{M}$-adhesive HLR systems can be considered as a unifying framework for graph and Petri net transformations providing enough structure that most notions and results from algebraic graph transformation systems are available, as results on parallelism and concurrency of rules and transformations, results on negative application conditions and constraints, and so on (e.g. in [EEPT06, EGH+12]).

Net morphisms map places to places and transitions to transitions. They are given as a pair of mappings for the places and the transitions, so that the structure and the decoration is preserved and the marking must be mapped so that the tokens are be preserved. A rule in the DPO approach[2] is given by three nets called left hand side L, interface K and right hand side R, respectively, and two strict net morphisms $K \to L$ and $K \to R$. Additionally, a match morphism $m : L \to N_1$ is required that identifies the relevant parts of the left hand side in the given net N_1.

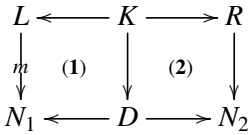


Figure 2: Transformation of a net

[2] Note, the transformations in ReConNet are DPO transformation in the cospan approach, so that the intermediate K is the union and not the intersection of the left-hand L and the right-hand R. The cospan approach is known to be equivalent to the usual DPO-span approach (see [EHP09]), so we refer in this paper to the usual DPO approach.

Then a transformation step $N_1 \overset{(r,m)}{\Longrightarrow} N_2$ via rule r can be constructed in two steps. Given a rule with a match $m : L \to N_1$ the gluing conditions have to be satisfied in order to apply a rule at a given match. These conditions ensure the result is again a well-defined net. In this case, we obtain a net N_2 leading to a direct transformation $N_1 \overset{(r,m)}{\Longrightarrow} N_2$ consisting of the following pushouts (1) and (2) in Fig. 2. Hence we can combine one net N together with a set of rules $\mathcal{R}$ leading to reconfigurable place/transition nets $rN = (N, \mathcal{R})$.

3 Introduction to Maude and its Model-Checker

Maude is a rewriting system operating on typed terms and has been developed at SRI International[3] for over two decades. The underlying theory of rewriting logic can be considered as a unifying framework for concurrency formalisms [Mes92]. Various concurrency formalisms are known to be modeled in terms of Maude, especially different kinds of Petri nets, e.g. P/T nets, colored Petri nets, and algebraic Petri nets ([SMÖ01]).

A system is represented using membership equational logic describing its set of states and a set of rewrite rules representing its state transitions. Maude is strictly typed, where the types are called sorts and can be built hierarchically using subsorts. Maude's basic programming statements are equations and rules, and have in both cases a simple rewriting semantics in which instances of the left-hand side pattern are replaced by corresponding instances of the right-hand side.

```
mod PN is
        sorts Place Marking .
        subsort Place < Marking .
        op __ : Marking Marking -> Marking       [assoc comm] .
        ops $ q a c : -> Place .

        rl [buy-c] : $ => c .
        rl [buy-a] : $ => a  q .
        rl [change] : q  q  q  q => $ .
endm
```

Listing 1: Net from Fig. 3 in Maude

Maude modules comprises a term-language plus sets of equations and rewrite-rules that specify the dynamics of a system, given by `rl [l] : t => t`, with `l` being the rule label. These rules describe the local, concurrent transitions that can occur in the system. Here we give a short example how a Petri net can be specified in Maude (see [CDE+99]). The Maude specification[4] in Listing 1 models the Petri net given in Fig. 3 where the places are given by constants of sort `Places` and the

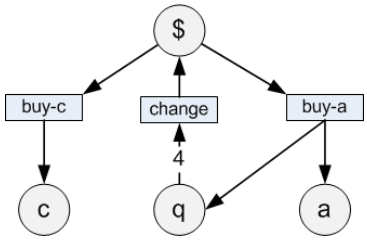


Figure 3: Net from List. 1

[3] URL: http://maude.cs.uiuc.edu/
[4] URL: http://maude.cs.uiuc.edu/maude1/manual/CoreMaudeExamples/petri-net.maude

transitions by rewrite rules. The marking of net is a multiset of places. So firing a transition is modeled by rewriting one multiset of places by another one.

We assume the reader to have some basic knowledge on Linear Temporal Logic[5] (LTL) and merely give a basic outline. LTL is a temporal logic with a widespread use, well-developed proof methods and decision procedures. It allows the specification of properties such as safety properties (ensuring that something bad never happens) and liveness properties (ensuring that something good eventually happens). Model checking can be used to prove properties, specified in LTL when the set of states reachable from an initial state in a system module is finite. LTL model checking is based on a Kripke structure $\mathcal{K}$. Then, the model checker solves a satisfaction problem $\mathcal{K}, s \models \phi$ for a Kripke structure $\mathcal{K}$, one of its states s and a property $\phi \in LTL(AP)$ where $LTL(AP)$ is the set of LTL formulas over the atomic propositions AP. LTL formulas are built up inductively from a finite set of atomic propositions AP, the logical connectives negation and conjunction, and the temporal modal operators "Next" **O** and "Until" **U**. Other LTL connectives can be defined in terms of the above minimal set of connectives. In this paper we need the following LTL connectives that given for each path of the Kripke structure $\mathcal{K}$ starting in state s and each LTL-formula ϕ.

- Next: $\mathcal{K}, s \models \mathbf{O}\phi$ states that ϕ has to hold at the next state, informally: $s \to \phi \to \bullet \to \bullet \to \cdots$
- Finally: $\mathcal{K}, s \models \mathbf{F}\phi$ states that ϕ eventually has to hold somewhere on the subsequent path, informally: $s \to \bullet \to \cdots \to \phi \to \bullet \to \cdots$
- Globally: $\mathcal{K}, s \models \mathbf{G}\phi$ states that ϕ has to hold on the entire subsequent path, informally: $\phi \to \phi \to \phi \to \phi \to \cdots$

An important way to model check a system is to express the desired properties using LTL formulas and actually check if the model satisfies this property. If the property is satisfied, the result of model checking is true; if not, then there is a counterexample, that is a trace of the model in which the property does not hold.

The Maude LTL model checker[6] supports on-the-fly explicit-state model checking of concurrent systems expressed as rewrite theories [EMS02]. Maude's LTL syntax is defined in the module `LTL`. The logical and temporal operators used in Section 4 in Maude's notation are negation `~`, conjunction `/\`, next `O`, finally `<>` and globally `[]`. Based on the Maude module a Kripke structure needs to defined. The sorts that are considered to be states and the set of atomic propositions needs to be defined explicitly depending on the model (see module `mod TRANS` in Section 4). In the `MODEL-CHECKER` module the operator `modelCheck` is defined. `modelCheck` maps a state and an LTL formula to a Boolean value. If the formula is not satisfied, the Boolean is false and a counterexample is returned, else it is true.

[5] See e.g. URL: http://en.wikipedia.org/wiki/Linear_temporal_logic
[6] URL: http://maude.cs.uiuc.edu/maude2-manual/html/maude-manualch10.html

4 Model-Checking of Reconfigurable Nets

The conversion of a reconfigurable Petri net to a Maude specification employs the term rewriting and the equation logic modules. Term rewriting is used for the computation of the markings and for the transformation of the net.

For better readability we have simplified the Maude modules slightly by leaving out some line consuming details, as capacities, dealing with identities an so on. A comprehensive report of the conversion of decorated Petri nets to Maude and its implementation can be found in [Sch14].

First we exemplify the translation of the reconfigurable net in Fig. 1 to a Maude specification. Maude rules are explicitly called rewrite rules (or Maude rules) to distinguish them from the net rules belonging to a reconfigurable Petri net. Module `mod RPN` models the net structure, the rules and the firing behavior and module `mod PROP` defines the atomic propositions the LTL is based on. The Maude modules `mod NET` and `mod RULES` can be generated from the reconfigurable net $rN = (N1, \{switchArc\})$ in Fig. 1. Module `mod NET` defines the place/-transition net together with the initial stated and `mod RULES` specifies the net transformation rules and their application. Since the whole specification (see [Sch14]) is quite large we only present clippings of the modules. Module `mod PN` exploits the algebraic definition of place/-transition nets and is given accordingly, see List. 2. There are the obvious sorts modeling the corresponding sets in the formal definition. The sort `MappingTuple` in line 8 and operation `->` in line 19 are the set representation of functions and are used for the definition of the pre- and post-functions in lines 22 and 23. The operator `_,_` (line 15) together with `emptyPlace` (line 13) specifies a multiset (or free commutative monoid) by the attributes `[ctor assoc comm id: emptyPlace]`, stating the operator to be an associative, commutative constructor with the identity `emptyPlace`. Similar operators are given for transitions (op `_:_`) and markings. Places and transitions have a name and an identifier (line 26 and 27). that contains the pre-domain of `T`. The rewriting rule in lines 30 to 40 describes the firing of a set of transitions.

A net consists of some set of places, i.e. the variable `P`, of transitions, those to be fired `T` and the others `TRest`, the pre- and post-domain functions of `T` and a marking `markingPreValue ; M`. This net can be rewritten to the net, that has then same structure, but a new marking `calc(((PreValue ; M) minus PreValue) plus PostValue)` modeling exactly the formal definition: $M' = M \ominus pre(t) \oplus post(t)$. Rules are given by a left hand side and a right hande side net (lines 42-44). And at last a reconfigurable net is given by a place/transition net and a set of rules (line 46).

Note, that for rules that delete places conditional rewrite rules need to be generated. The dangling condition, i.e., no deletion that leads to dangling arcs or markings, is the ensured by an additional condition. This condition guarantees that all adjacent transitions are deleted as well and that the place to be deleted has he same amount of tokens in the net as well as in the rule. Examples of such rules can be found in [Sch14].

If several rewrite rules are available, Maude always chooses the first that is applicable. In terms of simulation this would lead to a strongly reduced, and hence wrong behavior. Since model checking is only concerned with the existence of paths this fact only reduces the amount of non-determinism.

```
mod RPN
[...]
  sort Net .
  sort Places .
  sort Transitions .
  sort Pre .
  sort Post .
  sort MappingTuple .
  sort Markings .
[...]
  subsort Places < Markings .
[...]
  op emptyPlace : -> Places .
[...]
  op _,_ : Places Places -> Places [ctor assoc comm id: emptyPlace] .
[...]
  op p(_|_) : String Int -> Places .
  op t(_|_) : String Int -> Transitions .
  op (_-->_) : Transitions Places -> MappingTuple .
  op places{_} : Places -> Places .
  op transitions{_} : Transitions -> Transitions .
  op pre{_} : MappingTuple -> Pre .
  op post{_} : MappingTuple -> Post .
  op marking{_} : Markings -> Markings .
[...]
  op calc : Markings -> Markings .
  op _plus_ : Markings Markings -> Markings .
  op _minus_ : Markings Markings -> Markings .
[...]
  rl [fire] :
      net(P,
          transitions{T: TRest},
          pre{(T --> PreValue), MTupleRest1},
          post{(T --> PostValue), MTupleRest2},
          marking{PreValue ; M}) =>
      net(P,
          transitions{T: TRest},
          pre{(T --> PreValue), MTupleRest1},
          post{(T --> PostValue), MTupleRest2},
          calc(((PreValue ; M) minus PreValue) plus PostValue)) .
[...]
  op l : Net -> LeftHandSide .
  op r : Net -> RightHandSide .
  op rule : LeftHandSide RightHandSide -> Rule .
  sort Configuration .
  op __ : Net Rule -> Configuration .
```

Listing 2: Maude specification of reconfigurable place/transition nets

The second module `mod PROP` in List. 3 contains the definitions connecting the nets algebraic structure to the LTL model checking modules. The atomic propositions `enabled` and `reachable` are defined as operators of sort `Prop` for checking activation of transitions or for checking reachability of places. If a place is in the marking, then the operator `enabled` equals `true`. If the pre-domains of the transitions are in the marking the corresponding operator `enabled` equals `true`.

The Maude modules `mod NET` and `mod RULES` specify the reconfigurable net $rN = (N1, \{switchArc\})$ in Fig. 1. The equation in List. 4 states that the initial configuration contains net $N1$ in lines 4 to 8 and the rule *switchArc* in lines 9 to 18, where the left-hand side is in lines 10 to 13 and the right-hand side in lines 14 to 18. The marking of net $N1$ (namely, two tokens on place $A,1$) is given by `marking p("A" | 1) ; p("A" | 1) )` in line 8.

```
mod PROP is
  including RULES .
  including SATISFACTION .
[...]
  subsort Markings < State .
  subsort Net < State .
[...]
  op reachable : Markings -> Prop .
  eq net(P ,
         T ,
         Pre ,
         Post ,
         marking{ M ; MRest } )
     RRest
     |= reachable(M) = true .
[...]
  op enabled : -> Prop .
  eq net(P ,
         T ,
         pre{ (T --> M) } ,
         Post ,
         marking{ M ; MRest } )
     RRest
     |= enabled = true .
[...]
```

Listing 3: Maude specification of atomic properties for reconfigurable nets

```
1  mod NET is
2  [...]
3    eq initial =
4    net(places{p("A"|1),p("A"|2)},
5      transitions{t("X"|3) : t("X"|4)},
6      pre{(t("X"|3)-->p("A"|1)),(t("X"|4)-->p("A"|1))},
7      post{(t("X"|3)-->p("A"|2)),(t("X"|4)-->p("A"|2))},
8      marking{p("A"|1);p("A"|1)})
9    rule(
10     l(net(places{p("A"|1),p("A"|2)},
11       transitions{t("X"|3)},
12       post{(t("X"|3)-->p("A"|2))},
13       marking{p("A"|2)})),
14     r(net(places{p("A"|1),p("A"|2)},
15       transitions{t("X"|23)},
16       pre{(t("X"|23)-->p("A"|2))},
17       post{(t("X"|23)-->p("A"|1))},
18       marking{p("A"|2)}))).
```

Listing 4: Maude specification of $rN = (N1, \{switchArc\})$

Module `mod RULES` in List. 5 defines the transformation step induced by *switchArc*. The rewrite rule **`rl`** `[switchArc]` (line 4-19) requires a net (line 5-9) that contains the left-hand side, where the identifiers are variables (`Irule1`,`Irule2`,`Irule3`) and a rule (omitted in List. 5) and yields a net (line 13-17), where the arcs of the transition with name `X` are switched.

```
1  mod RULES is
2    including RPN .
3  [...]
4    rl [switchArc]:
5      net(places{p("A"|Irule1),p("A"|Irule2),PRest},
6        transitions{t("X"|Irule3) : TRest},
7        pre{(t("X"|Irule3)-->p("A"|Irule1)),MTupleRest1},
8        post{(t("X"|Irule3)-->p("A"|Irule2)),MTupleRest2},
9        marking{p("A"|Irule2) ; MRest})
10 rule(l(net(places{p("A"|Irule1),p("A"|Irule2)},
11 [...]
12   =>
13   net(places{p("A"|Irule1),p("A"|Irule2),PRest},
14     transitions{t("X"|Irule23) : TRest},
15     pre{(t("X"|Irule23)-->p("A"|Irule2)),MTupleRest1},
16     post{(t("X"|Irule23)-->p("A"|Irule1)),MTupleRest2},
17     marking{p("A"|Irule2); MRest})
18  rule(l(net(places{p("A"|Irule1),p("A"|Irule2)},
19 [...]
```

Listing 5: Maude module for rule *switchArc*

Loading these modules together with the Maudes's LTL model checker allows the rewrite system to prove LTL formulas based on the atomic propositions given in module `mod PROP`. So, we use Maude's LTL model checker to verify properties of reconfigurable Petri nets, checking whether these properties are satisfied by the nets and its rules or not. Subsequently we give examples how to use the Maude LTL checker for proving or disproving LTL formulas on reconfigurable Petri nets. These examples are given in the listings below and are the output at Maude's console.

Considering the reconfigurable Petri net $rN = (N1, \{switchArc\})$ in Fig. 1 consisting of the net $N1$ and the rule *switchArc*, the net together with the rules is free of deadlocks. The corresponding property means that `enabled` holds globally; in terms of Maude `[] enabled`. In module `mod PROP` the property `enabled` has been defined to be true, whenever the pre domain of some transition is part of the current marking or the left-hand side of the rule is part of the net. So, at each state some transition is enabled or some rule may be applied, hence there is no deadlock.

```
Maude> rew  modelCheck(initial, []enabled).
rewrite in NET : modelCheck(initial, []enabled).
rewrites: 66 in 1628036047000ms cpu (6ms real) (0 rewrites/second)
result Bool: true
```

Listing 6: No deadlock in rN; proven by Maude

In $rN = (N1, \{switchArc\})$ the second place A is reachable, i.e. there is in each path at least one state, where $A,2$ is marked with at least one token. In Maude this is denoted by `<> reachable(p("A" | 2) ) )`. A liveness condition is that the place $A,2$ can always (from all reachable markings) be reached again. This can be phrased as "For all paths and from all states $A,2$ can finally be reached". In Maude this is stated by `[] <> reachable(p("A" | 2)` and proven by its model checker in Listing 7.

```
Maude> rew modelCheck(initial, [] <> reachable(p("A" | 2) ) ) .
rewrite in NET : modelCheck(initial, []<> reachable(p("A" | 2))) .
rewrites: 69 in 1628036047000ms cpu (4ms real) (0 rewrites/second)
result Bool: true
```

Listing 7: Liveness condition for rN; proven by Maude

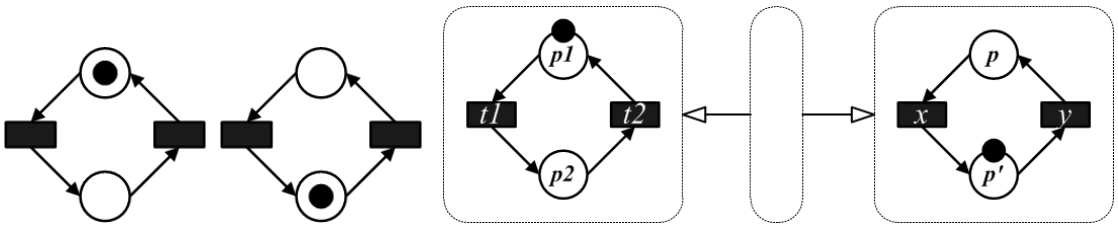
For properties that are not satisfied, the model checker returns a counterexample consisting of two lists of transitions. A transition is given by a state and the label of the rule applied to reach the next state. The first list describes a path starting from the initial state, that violates the given property. Since an LTL property being not valid in a finite Kripke structure always results in a path ending with a cycle, the second list of transitions is a loop. For an extensive discussion of a counterexample see [Sch14].

5 Related Work

To regard Petri nets as special bipartite graph is an obvious and sound idea. In [MEE10, MEE12] a concise, comprehensive and categorical translation of net transformations to graph transformations is given. An $\mathcal{M}$-functor from the category of place/transition nets with individual tokens to the category of typed attributed graphs translates rules preserving and reflecting applicability and transformations. The approach to reconfigurable Petri nets based on AGG [AGG13] and RON [BEHM07] exploits this translation and could also be used as input to Maude using one of the existing graph transformation encodings for Maude [CEC13, RGLV09, AS02]. Another minor difference to the approach we have been following with reconfigurable Petri nets is that our approach has the classical collective token semantics.

Nevertheless, there are some differences concerning the notion of a state. In algebraic graph transformations a state is usually an isomorphism class of graphs due the construction using pushouts. In Petri nets a state is a marked net. In Fig. 4 (see also [Pad12]) there are the isomorphic nets N_1 and N_2, in terms of Petri nets they are distinct states, whereas in terms of graph transformations they denote the same state. The sequential firing of the transitions in net N_1 in Fig. 4(a) leads to $N_1[t_1\rangle N_2[t_2\rangle N_1$. Obviously, the nets N_1 and N_2 are isomorphic, but they have to be differentiated in order to describe the firing adequately. In [Pad12] this problem is solved using standard isomorphisms. A promising approach to avoid the standard isomorphism construction is to examine whether transformation and firing sequences have common nodes that have been preserved over the different sequences. This approach has been developed in [Plu05] to obtain a Critical Pair Lemma for hypergraph rewriting that guarantees local confluence.

To ensure that in a graph transformation system the nets N_1 and N_2 can be modeled distinctly, labels can obviously be used. But then the explicit handling of labels needs to modeled as



(a) Net N_1 (b) Net N_2 (c) Rule $L \Longrightarrow R$

Figure 4: Notion of States in Petri nets and graph transformations

well. For purposes of automatic model checking this is only a small drawback, but for modeling purposes it is not adequate. The Maude modules also require a sophisticated handling of labels in terms of identities.

We have achieved a conversion from reconfigurable Petri nets to Maude modules that then can be checked using Maude's LTL model checker. Related work concerns mainly various attempts to model Petri nets or graph transformations in Maude, e.g. [CEC13, RGLV09, SMÖ01, AS02, BBR+11]. Closely related is [FL04], where object Petri nets are translated to Prolog and Maude. Model-checking the corresponding Maude modules has been the focus in [CEC13, RGLV09, AS02, BBR+11, FL04].

Model-checking of graph transformation systems (that is for the SPO approach) can be achieved by translating them into abstract state machines [Var04]. Model checking for graph transformation systems in the DPO approach is not known (at least by the authors). Nevertheless the example reconfigurable net in Fig. 1 can be modeled in Groove (an SPO graph transformation tool with model-checking features [Gro14, KR06]) using for rules for the firing of the transitions and the switching of the arcs. Another possibility for bounded reconfigurable Petri nets is the translation into Petri nets and the subsequent analysis with existing model checking tools as for example Snoopy and Charlie [sno14, HHL+12].

6 Conclusion

This paper introduces a conversion from reconfigurable Petri nets to Maude modules that then are employed for checking LTL properties. This is a first step towards a suitable verification approach for reconfigurable Petri nets. Of course there are other possibilities to tackle this problem (see Sect. 5). These approaches may have some restrictions, but for fitting examples it is very interesting to use them for benchmarking the model checking.

Future work concerns the presentation of the results gained from the model checking as well. The representation of the counterexamples in Maude as text is hard to interpret. An interpretation of the counterexamples in ReConNet would be very helpful. The paths described in the counterexamples could be used to animate the corresponding sequence of transformations and firing steps.

Acknowledgements: We are grateful to the referees for their valuable remarks.

References

[AGG13] AGG. The Attributed Graph Grammar System. 2013. Revision: 02/10/2013 15:25. http://user.cs.tu-berlin.de/~gragra/agg/

[AS02] N. Aoumeur, G. Saake. Integrating and Rapid-Prototyping UML Structural and Behavioural Diagrams Using Rewriting Logic. In Pidduck et al. (eds.), *Advanced Information Systems Engineering (CAiSE 2002)*. Lecture Notes in Computer Science 2348, pp. 296–310. Springer, 2002.

[BBR+11] P. Barbosa, J. P. Barros, F. Ramalho, L. Gomes, J. Figueiredo, F. Moutinho, A. Costa, A. Aranha. SysVeritas: A Framework for Verifying IOPT Nets and Execution Semantics within Embedded Systems Design. In *Technological Innovation for Sustainability*. Pp. 256–265. Springer, 2011.

[BEHM07] E. Biermann, C. Ermel, F. Hermann, T. Modica. A Visual Editor for Reconfigurable Object Nets based on the ECLIPSE Graphical Editor Framework. In Juhas and Desel (eds.), *Proc. 14th Workshop on Algorithms and Tools for Petri Nets (AWPN'07)*. GI Special Interest Group on Petri Nets and Related System Models, Universität Koblenz-Landau, Germany, 2007.

[BMMS98] R. Bruni, J. Meseguer, U. Montanari, V. Sassone. A Comparison of Petri Net Semantics under the Collective Token Philosophy. In Hsiang and Ohori (eds.), *Advances in Computing Science ASIAN 98*. Lecture Notes in Computer Science 1538, pp. 225–244. Springer Berlin Heidelberg, 1998.
urlhttp://dx.doi.org/10.1007/3-540-49366-2_18

[CDE+99] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J. F. Quesada. Maude: Specification and Programming in Rewriting Logic. Technical report, Computer Science Laboratory, SRI International, 1999.

[CEC13] W. Chama, R. Elmansouri, A. Chaoui. Using graph transformation and maude to simulate and verify UML models. In *Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*. Pp. 459–464. 2013.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer, 2006.

[EGH+12] H. Ehrig, U. Golas, A. Habel, L. Lambers, F. Orejas. M-AdhesiveTransformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence. *Fundam. Inform.* 118(1-2):35–63, 2012.

[EHP+07] H. Ehrig, K. Hoffmann, J. Padberg, U. Prange, C. Ermel. Independence of Net Transformations and Token Firing in Reconfigurable Place/Transition Systems. In Kleijn and Yakovlev (eds.), *Petri Nets and Other Models of Concurrency - ICATPN 2007*. Lecture Notes in Computer Science 4546, pp. 104–123. Springer, 2007.

[EHP09] H. Ehrig, F. Hermann, U. Prange. Cospan DPO Approach: An Alternative for DPO Graph Transformations. *Bulletin of the EATCS* 98:139–149, 2009.

[EJPR01] H. Ehrig, G. Juhás, J. Padberg, G. Rozenberg (eds.). *Unifying Petri Nets, Advances in Petri Nets*. Lecture Notes in Computer Science 2128. Springer, 2001.

[EMS02] S. Eker, J. Meseguer, A. Sridharanarayanan. The Maude LTL Model Checker. *Electr. Notes Theor. Comput. Sci.* 71:162–187, 2002.

[EP03] H. Ehrig, J. Padberg. Graph Grammars and Petri Net Transformations. In Desel et al. (eds.), *Lectures on Concurrency and Petri Nets*. Lecture Notes in Computer Science 3098, pp. 496–536. Springer, 2003.

[FL04] B. Farwer, M. Leuschel. Model Checking Object Petri Nets in Prolog. In *Principles and Practice of Declarative Programming*. Pp. 20–31. ACM, 2004.

[Gab14] K. Gabriel. *Interaction on Human-Centric Communication Platforms: Modelling and Analysis using Algebraic High-Level Nets and Processes*. PhD thesis, Technische Universität Berlin, 2014.

[Gro14] Groove. GRaphs for Object-Oriented VErification. 2014. Revision: 28/06/2014 18:25. http://groove.cs.utwente.nl/

[HHL+12] M. Heiner, M. Herajy, F. Liu, C. Rohr, M. Schwarick. Snoopy - A Unifying Petri Net Tool. In Haddad and Pomello (eds.), *Application and Theory of Petri Nets - 33rd International Conference, PETRI NETS 2012, Hamburg, Germany, June 25-29, 2012. Proceedings*. Lecture Notes in Computer Science 7347, pp. 398–407. Springer, 2012.

[KCD10] L. Kahloul, A. Chaoui, K. Djouani. Modeling and Analysis of Reconfigurable Systems Using Flexible Petri Nets. In *Theoretical Aspects of Software Engineering (TASE)*. Pp. 107–116. 2010.

[KR06] H. Kastenberg, A. Rensink. Model Checking Dynamic States in GROOVE. In Valmari (ed.), *Model Checking Software, 13th International SPIN Workshop, Vienna, Austria, March 30 - April 1, 2006, Proceedings*. Lecture Notes in Computer Science 3925, pp. 299–305. Springer, 2006.

[LO04] M. Llorens, J. Oliver. Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Petri Nets. *IEEE Trans. Computers* 53(9):1147–1158, 2004.

[MEE10] M. Maximova, H. Ehrig, C. Ermel. Formal Relationship between Petri Net and Graph Transformation Systems based on Functors between M-adhesive Categories. *ECEASST* 40, 2010.

[MEE12] M. Maximova, H. Ehrig, C. Ermel. Transfer of Local Confluence and Termination between Petri Net and Graph Transformation Systems Based on M-Functors. *ECEASST* 51, 2012.

[Mes92] J. Meseguer. Conditioned Rewriting Logic as a United Model of Concurrency. *Theor. Comput. Sci.* 96(1):73–155, 1992.

[MM90] J. Meseguer, U. Montanari. Petri Nets Are Monoids. *Information and Computation* 88(2):105–155, 1990.

[Mod12] T. Modica. *Formal Modeling, Simulation, and Validation of Communication Platforms*. PhD thesis, Technische Universität Berlin, 2012.

[Pad12] J. Padberg. Abstract Interleaving Semantics for Reconfigurable Petri Nets. *ECEASST* 51, 2012.

[PEHP08] U. Prange, H. Ehrig, K. Hoffman, J. Padberg. Transformations in Reconfigurable Place/Transition Systems. In Degano et al. (eds.), *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*. Lecture Notes in Computer Science 5065, pp. 96–113. Springer Verlag, 2008.

[PEOH12] J. Padberg, M. Ede, G. Oelker, K. Hoffmann. ReConNet: A Tool for Modeling and Simulating with Reconfigurable Place/Transition Nets. *ECEASST* 54, 2012.

[Plu05] D. Plump. Confluence of Graph Transformation Revisited. In Middeldorp et al. (eds.), *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*. Lecture Notes in Computer Science 3838, pp. 280–308. Springer, 2005.

[Rei12] F. Reiter. Modellierung und Analyse von Szenarien des Living Place mit rekonfigurierbaren Petrinetzen. Bachelor Thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2012. http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/reiter.pdf

[RGLV09] J. E. Rivera, E. Guerra, J. de Lara, A. Vallecillo. Analyzing Rule-Based Behavioral Semantics of Visual Modeling Languages with Maude. In Gasevic et al. (eds.), *Software Language Engineering, First International Conference*. Lecture Notes in Computer Science 5452, pp. 54–73. Springer, 2009.

[Sch14] A. Schulz. Model checking for reconfigurable Petri nets. 2014. http://arxiv.org/abs/1409.8404

[SMÖ01] M.-O. Stehr, J. Meseguer, P. C. Ölveczky. Rewriting Logic as a Unifying Framework for Petri Nets. In *Unifying Petri Nets*. Pp. 250–303. 2001.

[sno14] snoopy. 2014. http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy

[Var04] D. Varró. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling* 3(2):85–113, 2004.