Verification of Reconfigurable Petri Nets

Julia Padberg
Outline

- Motivation
- Example: Dynamic Hardware Reconfiguration
- Basic Ideas
- Reconfiguration of Petri Nets
- Model-checking reconfigurable place/transition nets with Maude
- Conclusion
What are Reconfigurable Petri Nets?

- a family of formal modelling techniques
- providing a powerful and intuitive formalism to model complex coordination and structural adaptation at run-time (e.g. mobile ad-hoc networks, communication spaces, ubiquitous computing)
- possibility to discriminate between different levels of change
Basic Ideas of Reconfigurable Petri Nets

First level of dynamics

Petri Net

- place/transition, decorated or algebraic high-level nets
- token game
- typically: modelling the dynamic of a system at some point

Second level of dynamics

Net transformation

- given in terms of rules
- replacement of subnet $L$ with $R$ considering an overlap $K$
- typically: modelling the change of a system during some evolution
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Net Rules and Transformations
Labeled Transition System $LTS_{RPN}$
Outline

- Motivation
- Example: Dynamic Hardware Reconfiguration
  - Industrial Mixer
  - Petri Net Model of Mixer’s Control Logic
  - Dynamic Reconfiguration of Control Logic
- Basic notions
- Reconfiguration of Petri nets
- Model-checking reconfigurable place/transition nets with Maude
- Conclusion
Partial Dynamic Reconfiguration of FPGAs

Basic observations

- Petrinets are established modeling technique for Field Programmable Gate Arrays (FPGAs)
- Dynamic reconfiguration of FPGAs can be modelled using reconfiguration of Petri nets
Example: Industrial Mixer

- mixing two components and water from feeders
- water is heated before being fed into the mixer
- sensors $x_1, \ldots, x_8$ measuring the fill level
- $x_9$ measures the water temperature
- actuators $y_1, \ldots, y_9$ controlling valves, heater and mixer
Control Logic of an Industrial Mixer

- determines the order of feeding the components
  - simultaneous feeding: both components and the hot water are added at the same time,
  - sequential feeding (1): first the heated water, then both components
  - sequential feeding (2): first component 1 and the water, and at last component 2.
- is modeled using a Petri net
- can be translated into the binary coding of an FPGA
Petri Net Model of Mixer’s Control Logic

Simultaneous feeding
Petri Net Model of Mixer’s Control Logic

Simultaneous feeding

Actuators:
- $y_1$, $y_2$ and $y_3$ for valves
Simultaneous feeding

Actuators
- $y_1$, $y_2$ and $y_3$ for valves
- $y_4$ and $y_5$ for valves, $y_6$ for heater
Simultaneous feeding

Petri Net Model of Mixer’s Control Logic

Actuators

- y₁, y₂ and y₃ for valves
- y₄ and y₅ for valves,
  y₆ for heater
- y₄, y₅ and y₇ for valves
Simultaneous feeding

Petri Net Model of Mixer’s Control Logic

actuators

- $y_1$, $y_2$ and $y_3$ for valves
- $y_4$ and $y_5$ for valves, $y_6$ for heater
- $y_4$, $y_5$ and $y_7$ for valves
- $y_8$ for mixer,
Simultaneous feeding

**Actuators**
- $y_1$, $y_2$ and $y_3$ for valves
- $y_4$ and $y_5$ for valves, $y_6$ for heater
- $y_4$, $y_5$ and $y_7$ for valves
- $y_8$ for mixer
- $y_9$ for valve
Simultaneous feeding

Petri Net Model of Mixer’s Control Logic

- $y_1$, $y_2$ and $y_3$ for valves
- $y_4$ and $y_5$ for valves, $y_6$ for heater
- $y_4$, $y_5$ and $y_7$ for valves
- $y_8$ for mixer
- $y_9$ for valve
- $y_1$, $y_2$ and $y_3$ for valves
Dynamic Reconfiguration of Control Logic

Simultaneous feeding
both components and the hot water are added at the same time,

Sequential feeding (1)
first the heated water, then both components

or

Sequential feeding (2)
first component 1 and the water, and at last component 2.
Simultaneous feeding
both components and the hot water are added at the same time,

sequential feeding (1)
first the heated water, then both components
Dynamic Reconfiguration of Control Logic

**Simultaneous feeding**
both components and the hot water are added at the same time,

**rule 1**

**sequential feeding (1)**
first the heated water, then both components

actuators $y_4$, $y_5$ and $y_7$ for valves from the feeders and $y_8$ for mixer
Dynamic Reconfiguration of Control Logic

Simultaneous feeding

Sequential feeding (1)

Simultaneous feeding

Sequential feeding (1)

**Rule 1**

Simultaneous feeding

Sequential feeding (1)
Dynamic Reconfiguration of Control Logic

Simultaneous feeding
both components and the hot water are added at the same time,

Sequential feeding (2)
first component 1 and the water, and at last component 2.

$\text{Simultaneous feeding}$

$\text{Sequential feeding (2)}$

$\text{rule 1}$
Reconfigurable Petri Nets \( (N_{\text{Mixer}}, R) \)

Consisting of

- the Petri net \( N_{\text{Mixer}} \)
- and a set of rules \( R = \{ \text{rule}_1, \text{rule}_2, \text{rule}_1^{-1}, \text{rule}_2^{-1} \} \)
Model-Checking the Mixer’s Control Logic

Techniques for verification of a Petri net’s correctness

- static analysis (invariants, traps, siphons etc.)
- Model-Checking

But not for reconfigurable Petri nets

- not sufficient to verify the resulting nets
  - deadlocks can be resolved by the rules
  - livelock if no further rules can be used
- Model-Checking the reconfigurable Petri net

 Desired properties for the mixer’s control logic

- no deadlocks
- reversibility
- liveness
Model-Checking the Mixer’s Control Logic

Techniques for verification of a Petri net’s correctness

- static analysis (invariants, traps, siphons etc.)
- Model-Checking

But not for reconfigurable Petri nets

- not sufficient to verify the resulting nets
- deadlocks can be resolved by the rules
- livelock if no further rules can be used
- Model-Checking the reconfigurable Petri net

Desired properties for the mixer’s control logic

- no deadlocks
- reversibility
- liveness
## Model-Checking the Mixer’s Control Logic

### Techniques for verification of a Petri net’s correctness
- static analysis (invariants, traps, siphons etc.)
- Model-Checking

### But not for reconfigurable Petri nets
- not sufficient to verify the resulting nets
  - deadlocks can be resolved by the rules
  - livelock if no further rules can be used
- Model-Checking the reconfigurable Petri net

### Desired properties for the mixer’s control logic
- no deadlocks
- reversibility
- liveness
Outline

- Motivation
- Example: Dynamic Hardware Reconfiguration
- Basic notions
  - Place/transition nets
  - Reconfigurable place/transition nets
  - Model-checking with Maude
- Reconfiguration of Petri nets
- Model-checking reconfigurable place/transition nets with Maude
- Conclusion
Algebraic Notion of PT Nets

\[ N = (P, T, \text{pre}, \text{post}, m) \]

- a set of places \( P \)
- a set of transitions \( T \)
- the labeling \( \text{pname} : P \rightarrow C_P, \text{tname} : T \rightarrow C_T \)

- the pre- and post-domain \( \text{pre, post} : T \rightarrow P^\oplus \)

- markings \( m_i \in P^\oplus \)
- firing \( m_1[t]m_2 \) with \( m_1 \ominus \text{pre}(t) \oplus \text{post}(t) \)
Algebraic Notion of PT Nets

\[ N = (P, T, pre, post, m) \]

- a set of places \( P \) \( P = \{p_1, ..., p_4\} \)
- a set of transitions \( T \) \( T = \{t_1, t_2, t_3\} \)
- the labeling \( \text{pname} : P \rightarrow C_P, \text{tname} : T \rightarrow C_T \)
  \( \text{pname} : p_1 \mapsto $, \quad p_2 \mapsto c, \quad p_3 \mapsto q, \quad p_4 \mapsto a \)
  \( \text{tname} : t_1 \mapsto \text{buy-c}, \quad t_2 \mapsto \text{change}, \quad t_3 \mapsto \text{buy-a} \)
- the pre- and post-domain \( pre, post : T \rightarrow P^\oplus \)
- markings \( m_i \in P^\oplus \)
- firing \( m_1[t]m_2 \) with \( m_1 \ominus pre(t) \ominus post(t) \)
Algebraic Notion of PT Nets

\[ N = (P, T, \text{pre}, \text{post}, m) \]

- a set of places \( P \)
  \[ P = \{p_1, ..., p_4\} \]
- a set of transitions \( T \)
  \[ T = \{t_1, t_2, t_3\} \]
- the labeling \( \text{pname} : P \rightarrow \mathbb{C}_P, \text{tname} : T \rightarrow \mathbb{C}_T \)
  \[ \text{pname} : p_1 \mapsto \$ \quad p_2 \mapsto c \quad \text{tname} : t_1 \mapsto \text{buy-c} \quad t_2 \mapsto \text{change} \]
  \[ p_3 \mapsto q \quad p_4 \mapsto a \quad t_3 \mapsto \text{buy-a} \]
- the pre- and post-domain \( \text{pre}, \text{post} : T \rightarrow P^\oplus \)
  \[ \text{pre} : t_1 \mapsto p_1 \quad t_2 \mapsto 4p_3 \quad \text{post} : t_1 \mapsto p_2 \quad t_2 \mapsto p_1 \quad t_3 \mapsto p_1 \]
  \[ t_3 \mapsto p_3 \oplus p_4 \]
- markings \( m_i \in P^\oplus \quad m = 2p_1 \)
- firing \( m_1[t]m_2 \) with \( m_1 \ominus \text{pre}(t) \oplus \text{post}(t) \)
Algebraic Notion of PT Nets

\[ N = (P, T, \text{pre}, \text{post}, m) \]

- a set of places \( P \)
  \[ P = \{p_1, ..., p_4\} \]
- a set of transitions \( T \)
  \[ T = \{t_1, t_2, t_3\} \]
- the labeling \( p\text{name} : P \rightarrow C_P, t\text{name} : T \rightarrow C_T \)
  \[ p\text{name} : p_1 \mapsto \$ \quad p_2 \mapsto c \quad t\text{name} : t_1 \mapsto \text{buy-c} \quad t_2 \mapsto \text{change} \]
  \[ p_3 \mapsto q \quad p_4 \mapsto a \quad t_3 \mapsto \text{buy-a} \]
- the pre- and post-domain \( \text{pre}, \text{post} : T \rightarrow P^\oplus \)
  \[ \text{pre} : t_1 \mapsto p_1 \quad t_2 \mapsto 4p_3 \quad \text{post} : t_1 \mapsto p_2 \quad t_2 \mapsto p_1 \quad t_3 \mapsto p_3 \oplus p_4 \]
- markings \( m_i \in P^\oplus \) \( m = 2p_1 \)
- firing \( m_1[t]m_2 \) with \( m_1 \ominus \text{pre}(t) \oplus \text{post}(t) \)
- \( m[t_3]m_1 \) with

\[ m_1 = m \ominus \text{pre}(t) \oplus \text{post}(t) = 2p_1 \ominus p_1 \oplus p_3 \oplus p_4 = p_1 \oplus p_3 \oplus p_4 \]
Reconfigurable P/T Nets

- rule by span of morphisms \( L \leftarrow K \rightarrow R \)
  - deletion of \( L - K \), obsolete items
  - addition of \( R - K \), fresh items
- match of \( L \) in \( N \) by occurrence morphisms
- transformation \( N^{(r,o)} \rightarrow M \)
  - construction of \( C = N - o(L - K) \), deleting obsolete items in \( N \)
  - construction of \( M = d + c(R - K) \), adding fresh items to \( C \)
Reconfigurable P/T Nets

- rule by span of morphismsm \( L \leftarrow K \rightarrow R \)
  - deletion of \( L - K \), obsolete items
  - addition of \( R - K \), fresh items

- match of \( L \) in \( N \) by occurrence morphisms

- transformation \( N \xrightarrow{r,o} M \)
  - construction of \( C = N - o(L - K) \), deleting obsolete items in \( N \)
  - construction of \( M = d + c(R - K) \), adding fresh items to \( C \)
Reconfigurable P/T Nets

- rule by span of morphisms $L \leftarrow K \rightarrow R$
  - deletion of $L - K$, obsolete items
  - addition of $R - K$, fresh items
- match of $L$ in $N$ by occurrence morphisms
- transformation $N \xrightarrow{(r,o)} M$
  - construction of $C = N - o(L - K)$, deleting obsolete items in $N$
  - construction of $M = d + c(R - K)$, adding fresh items to $C$
Reconfigurable P/T Nets

- rule by span of morphismsm $L \leftarrow K \rightarrow R$
  - deletion of $L - K$, obsolete items
  - addition of $R - K$, fresh items

- match of $L$ in $N$ by occurrence morphisms

- transformation $N \xrightarrow{(r,o)} M$
  - construction of $C = N - o(L - K)$, deleting obsolete items in $N$
  - construction of $M = d + c(R - K)$, adding fresh items to $C$
Model Checking with Maude

- rewriting system operating on typed terms
- developed at Stanford University
- powerful & mature
- unifying framework for concurrency formalisms
  P/T nets, colored Petri nets, algebraic Petri nets, graph transformations
- membership equational logic for states
- (conditional) rewrite rules for state transition
Example Candy/Apple Maschine

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

  rl [buy-c] : $ => c .
  rl [buy-a] : $ => a q .
  rl [change] : q q q q => $ .
endm
LTL using Maude

- based on a Kripke structure $\mathcal{K}$, basically a transition system
- set of atomic propositions $AP$
- $LTL(AP)$ is the set of LTL formulas over these
- some property $\phi \in LTL(AP)$
- $\mathcal{K}, s \models \phi$ holds if for $\phi$ holds for all paths in $\mathcal{K}$ starting in state $s$
- model checker solves $\mathcal{K}, s \models \phi$

LTL connectives:

- Finally: $\mathcal{K}, s \models F\phi$:
  $\phi$ finally has to hold somewhere: $s \rightarrow \bullet \rightarrow \cdots \rightarrow \phi \rightarrow \bullet \rightarrow \cdots$
  Maude ModelChecker $<>$

- Globally: $\mathcal{K}, s \models G\phi$
  $\phi$ has to hold globally: $\phi \rightarrow \phi \rightarrow \phi \rightarrow \phi \rightarrow \cdots$
  Maude ModelChecker $[ ]$
Important Properties for some $rN = ((N_0, m_0), R)$

- Is a specific marking always reachable?
  - requires a atomic proposition $reachable$
    
    $reachable(m)$ iff $(N_0, m_0) \xrightarrow{\ast} (N_i, m_j) \text{ st } m \subseteq m_j$
    
    LTL property: $\text{GF} \ reachable(m)$

- Is the reconfigurable Petri net reversible?
  - requires the atomic proposition $reachable$
    
    LTL property: $\text{GF} \ reachable(m_0)$

- Is a reconfigurable Petri net free of deadlocks?
  - requires a atomic proposition $enabled$
    
    $enabled(N, m)$ iff
    
    $(N_0, m_0) \xrightarrow{\ast} (N_i, m_j) \text{ st } (N, m) = (N_i, m_j)$ and $\exists t \in T : \text{pre}(t) \subseteq m_j$ or $\exists N_i \xrightarrow{r.o} N_{i+1}$
    
    LTL property: $\text{G} \ enabled(m)$
Important Properties for some $rN = ((N_0, m_0), R)$

- Is a specific marking always reachable?
  - requires a atomic proposition $reachable$
    
    $$reachable(m) \iff (N_0, m_0) \xrightarrow{*} (N_i, m_j) \text{ st } m \subseteq m_j$$
  - LTL property: $GF reachable(m)$

- Is the reconfigurable Petri net reversible?
  - requires the atomic proposition $reachable$
  - LTL property: $GF reachable(m_0)$

- Is a reconfigurable Petri net free of deadlocks?
  - requires a atomic proposition $enabled$
    
    $$enabled(N, m) \iff$$
    
    $$(N_0, m_0) \xrightarrow{*} (N_i, m_j) \text{ st } (N_i, m_j) = (N, m) \text{ and } \exists t \in T : pre(t) \subseteq m_j \text{ or } \exists N_i \xrightarrow{r,0} N_{i+1}$$
  - LTL property: $G enabled(m)$
Important Properties for some \( rN = ((N_0, m_0), R) \)

- Is a specific marking always reachable?
  - requires a atomic proposition \( \text{reachable} \)
    
    \[
    \text{reachable}(m) \iff (N_0, m_0) \xrightarrow{*} (N_i, m_j) \text{ s.t. } m \subseteq m_j
    \]
  
    LTL property: \( \mathbf{G} \mathbf{F} \text{reachable}(m) \)

- Is the reconfigurable Petri net reversible?
  - requires the atomic proposition \( \text{reachable} \)
    
    LTL property: \( \mathbf{G} \mathbf{F} \text{reachable}(m_0) \)

- Is a reconfigurable Petri net free of deadlocks?
  - requires a atomic proposition \( \text{enabled} \)
    
    \[
    \text{enabled}(N, m) \iff (N_0, m_0) \xrightarrow{*} (N_i, m_j) \text{ s.t. } (N_0, m_0) = (N, m) \text{ and } \exists t \in T : \text{pre}(t) \subseteq m_j \text{ or } \exists N_i \xrightarrow{r,o} N_{i+1}
    \]
  
    LTL property: \( \mathbf{G} \text{enabled}(m) \)
Outline

- Motivation
- Example: Dynamic Hardware Reconfiguration
- Basic notions
- Reconfiguration of Petri nets
  - Net rules
  - Transformations
  - Instantiation of abstract transformation systems
- Model-checking reconfigurable place/transition nets with Maude
- Conclusion
Net Morphisms

Net morphisms consist of

\textit{structure- and marking preserving maps}

- mapping places to places and transitions to transitions
- preserving pre- and post-domain
- preserving markings
- preserving labels
Definition (PT net morphism)

Given $N_1$ and $N_2$ labeled PT nets over $(C_P, C_T)$ with

$$N_i := (P_i, T_i, \text{pre}_i, \text{post}_i, m_i, \text{pname}_i, \text{tname}_i) \text{ for } i = 1, 2.$$ 

A net morphism $f : N_1 \rightarrow N_2$ is a pair of mappings $f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow T_2)$,

- preserving pre- and post-domain
  1. $f_P \circ \text{pre}_1 = \text{pre}_2 \circ f_T$ und
  2. $f_P \circ \text{post}_1 = \text{post}_2 \circ f_T$

- preserving the marking
  3. $m_1(p) \leq m_2(f_P(p))$ for all $p \in P_1$

- preserving the labels
  4. $\text{pname}_1 = \text{pname}_2 \circ f_P$
  5. $\text{tname}_1 = \text{tname}_2 \circ f_T$

If $m_1(p) = m_2(f_P(p))$ for all $p \in P_1$ the morphisms is called strict.
Net Rules

Definition

Net rules Given $L$, $K$, und $R$ PT nets over $(C_P, C_T)$ the a net rule is given by a span of strict, injective net morphisms

$$r = (L \leftarrow K \rightarrow R)$$
**Definition (Net gluing)**

Let \( I, N_1, N_2 \) be nets with net morphisms \( I \xrightarrow{f} N_1 \) and \( I \xrightarrow{g} N_2 \) with \( g \) being strict injective. Let \( \equiv \) be the smallest equivalence relation on \( P_I \cup T_I \cup P_2 \cup T_2 \) which satisfies \( f(x) \equiv g(x) \) for all \( x \in P_I \cup T_I \).

The gluing \( N = N_1 +_I N_2 \) a net \( N = (P_N, T_N, \text{pre}_N, \text{post}_N, m_N, \text{pname}_N, \text{tname}_N) \) with:

\[
\begin{align*}
P_N &= (P_1 \cup P_2) \equiv \\
T_N &= (T_1 \cup T_2) \equiv
\end{align*}
\]

\[
\begin{align*}
\text{pre}_N([t]_\equiv) &= \begin{cases} 
\text{pre}_1(t) \equiv & \text{if } t \in T_1 \\
\text{pre}_2(t) \equiv & \text{if } t \in T_2
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{post}_N([t]_\equiv) &= \begin{cases} 
\text{post}_1(t) \equiv & \text{if } t \in T_1 \\
\text{post}_2(t) \equiv & \text{if } t \in T_2
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{pname}_N([p]_\equiv) &= \begin{cases} 
\text{pname}_1(p) & \text{if } p \in P_1 \\
\text{pname}_2(p) & \text{if } p \in P_2
\end{cases}
\end{align*}
\]

\[
\begin{align*}
m_N([p]_\equiv) &= \begin{cases} 
\text{m}_2(p) & \text{if } p \in P_1 \\
\text{m}_1(p) & \text{else}
\end{cases}
\end{align*}
\]
Transformation Step

Definition (Net transformation)

Let \( r = (L \leftarrow I \rightarrow R) \) be a rule.

A transformation step from \( N \xrightarrow{r} M \) the net \( N \) to the net \( M \) via \( r \) is given by a net \( C \) (the so-called context) and a net morphism \( I \rightarrow C \) such that:

\[
N \cong L +_I C \quad M \cong R +_I C
\]

depicted by the diagram (double-pushout diagram)

The morphism \( o \) is called the occurrence, \( c \) the co-occurrence.
Results: $\mathcal{M}$-Adhesive Category

Category $\mathcal{C}$ with class $\mathcal{M}$ of specific monomorphisms is $\mathcal{M}$-adhesive, iff

- $\mathcal{M}$ is closed under isomorphisms and (de-)composition
- $\mathcal{C}$ has pushouts and pullbacks preserving $\mathcal{M}$-morphisms
- pushouts along $\mathcal{M}$-morphisms are $\mathcal{M}$-Van-Kampen squares

Shoals of results: Local-Church-Rosser, parallelism, concurrency, embedding, extension, local confluence [EEPT06] nested constraints and application conditions [HP05,HP09] multi-amalgamation of rules (with application conditions) [GEH10, Gol11]
Construction of $M$-adhesive categories \cite{Pra06} Theorem 2.3

If $(C, M_1)$ and $(D, M_2)$ are $M$-adhesive categories, then following categories are also $M$-adhesive categories: [...]

(v) The comma category $(\text{ComCat}(F, G, I), M)$ with $M = (M_1 \times M_2)$ based on the functors $F : C \to X$, $G : D \to X$, where $F$ preserves pushouts along $M_1$-morphisms and $G$ preserves pullbacks along $M_2$-morphisms.

For example

the category $\text{PT}$ of place/transition nets can be constructed as the comma category $(\text{ComCat}(\text{ID}, (\_)^\oplus, \{\text{pre, post}\}), M)$ with $(\_)^\oplus$ preserving pullbacks along injective morphisms.

The category $\text{PT}$ of place/transition nets is a $M$-adhesive category.
Types of Reconfigurable Nets

\(\mathcal{M}\)-adhesive categories

**Low-level**
- P/T nets
- P/Ts with individual tokens
- decorated P/T nets
- P/T with inhibitor arcs and transition priorities
- timed P/T nets
- elementary nets

**High-level**
- AHL nets
- AHL nets with individual tokens
- AHO nets
- AHO nets with individual tokens
- colored Petri nets
- PR/T nets
Outline

- Motivation
- Example: Dynamic Hardware Reconfiguration
- Basic notions
- Reconfiguration of Petri nets
  - Model-checking reconfigurable place/transition nets with Maude
    - Short Intro to Maude
    - Translation to Maude
    - Correctness of MC approach
    - Tools
- Conclusion
Model checking Petri Nets

Lots of approaches, important results, mature tools

- Based on unfoldings/reachbility graph/partial order semantics
- e.g. “A False History of True Concurrency: From Petri to Tools” [Esp10]
- PEP[GB96], CPN [CCM97], Marcie [Tov16], Tina [BV06]...

but only for one net at a time
**Model checking Petri Nets**

Lots of approaches, important results, mature tools

- Based on unfoldings/reachability graph/ partial order semantics
- e.g. ”A False History of True Concurrency: From Petri to Tools”[Esp10]
- PEP[GB96], CPN [CCM97], Marcie [Tov16], Tina [BV06]...

but only for one net at a time
Example

Reconfigurable net with net $N1$ and the rule $\text{switchArc}$
Interleaving Semantics
Short Introduction to Maude

- equational and rewriting logic
- for modelling concurrent state systems
- internal representation as a labelled rewrite theory
- implementations based on one or many modules

**Syntax**

- $\Sigma$ an alphabet of functions,
- a set of equations $E$ over $\Sigma$,
- a set of labels $L$
- rewrite rules $R \subseteq L \times (T_{\Sigma,E}(X)^2)$

**Semantics**

**Labeled Transition System**

- labelled sequence $r : [t]_E \rightarrow [t']_E$.
- to be read as $[t]_E$ becomes $[t']_E$.
- extension with variables $r : [t(\bar{x}^n)]_E \rightarrow [t'(\bar{x}^n)]_E$
Short Introduction to Maude

- equational and rewriting logic
- for modelling concurrent state systems
- internal representation as a labelled rewrite theory
- implementations based on one or many modules

**Syntax**
- \( \Sigma \) an alphabet of functions,
- a set of equations \( E \) over \( \Sigma \),
- a set of labels \( L \)
- rewrite rules \( R \subseteq L \times (T_{\Sigma,E}(X)^2) \)

**Semantics**

**Labeled Transition System**
- labelled sequence \( r : [t]_E \rightarrow [t']_E \).
- to be read as \( [t]_E \) becomes \( [t']_E \).
- extension with variables \( r : [t(\bar{x}^n)]_E \rightarrow [t'(\bar{x}^n)]_E \)
Short Introduction to Maude

- equational and rewriting logic
- for modelling concurrent state systems
- internal representation as a labelled rewrite theory
- implementations based on one or many modules

**Syntax**
- $\Sigma$ an alphabet of functions,
- a set of equations $E$ over $\Sigma$,
- a set of labels $L$
- rewrite rules $R \subseteq L \times (T_{\Sigma,E}(X)^2)$

**Semantics**

**Labeled Transition System**
- labelled sequence $r : [t]_E \rightarrow [t']_E$.
- to be read as $[t]_E$ becomes $[t']_E$.
- extension with variables $r : [t(\vec{x}^n)]_E \rightarrow [t'(\vec{x}^n)]_E$
Short Introduction to Maude

1. sorts Places Transitions Markings.
2. op _ _ : Marking Marking → Marking
   [assoc comm].
3. op initial : → Markings.
4. ops A B : → Markings.
5. eq initial = A.
6. rl [T] : A ⇒ B B.
From Reconfigurable PT Nets to a Maude Modules

- equational logic for the ADT
  - algebraic notion of Petri nets
  - rules and configurations
- term rewriting
  - for the computation of the markings
  - for the transformation of the net
- Maude Modules
  - mod ModelChecker in S. Eker, J. Meseguer, A. Sridharanarayanan 2002
  - mod RPN and mod PROP generic for all reconfigurable PT nets
  - mod RULES and mod NET for each reconfigurable PT net
ADT PT Nets

\begin{verbatim}
mod RPN is

  sort Places .
  subsort Places < Markings .
  sort Transitions .
  sort Pre .
  sort Post .
  sort MappingTuple .
  sort Markings .

  op emptyPlace : \rightarrow Places .
  op \_,_ : Places Places \rightarrow Places
            [ctor assoc comm id: emptyPlace] .

  op (_\rightarrow>_ ) : Transitions Places \rightarrow MappingTuple .
  op pre\{\_\} : MappingTuple \rightarrow Pre .
  op post\{\_\} : MappingTuple \rightarrow Post .
\end{verbatim}
ADT PT Nets

mod RPN is

sort Places .
subsort Places ⊂ Markings .
sort Transitions .
sort Pre .
sort Post .
sort MappingTuple .
sort Markings .

op emptyPlace : → Places .
oper : Places Places → Places
ctor assoc comm id : emptyPlace .

oper : Transitions Places → MappingTuple .
oper pre{} : MappingTuple → Pre .
oper post{} : MappingTuple → Post .

• a set of places $P$
• a set of transitions $T$
• the pre- and post-domain
  \[pre, post : T \rightarrow P^\oplus\]
• markings $m_i \in P^\oplus$
Example Net

mod NET is

net (places{ p("A" | 1), p("A" | 2) },
transitions{ t("X" | 3) : t("X" | 4) },
pre{ (t("X" | 3) --> p("A" | 1)),
     (t("X" | 4) --> p("A" | 1)) },
post{ (t("X" | 3) --> p("B" | 2)),
      (t("X" | 4) --> p("B" | 2)) },
marking{ p("A" | 1); p("A" | 1) })

Julia Padberg
HAW Hamburg
Verification of Reconfigurable Petri Nets
Model-Checking with Maude

Linear Temporal Logic for Rewrite (LTLR) Module

- temporal operators
  - next-operator: $O \phi$
  - until-operator: $\psi U \phi$
  - release-operator: $\psi R \phi$
  - future-operator: $\Diamond \phi$
  - global-operator: $\Box \phi$

- and all the usual operators, such as true, false, conjunction, disjunction and negation

- The correctness of the LTLR model checker has been proven.
From Reconfigurable Petri Nets to Maude Modules

reconfigurable Petri net
given as PNML extension
From reconfigurable Petri nets to Maude modules

- the actual net and its rules as initial state
- properties defined by reachable, t-enabled and enabled
- the actual rules as rewrite rules
- enabling and firing as rewrite rules
From reconfigurable Petri nets to Maude modules

- Maude module **NET** including **PROP** including **MODEL-CHECKER**

  - the actual net and its rules as initial state

- Maude module **PROP** protecting **STRING** including **RULES**

  - properties defined by reachable, t-enabled and enabled

- Maude module **RULES** including **RPN**

  - the actual rules as rewrite rules

- Maude module **RPN** protecting **STRING** protecting **INT**

  - enabling and firing as rewrite rules
From reconfigurable Petri nets to Maude modules

Maude module NET
including PROP
including MODEL-CHECKER

Maude module PROP
protecting STRING
including RULES

Maude module RULES
including RPN

Maude module RPN
protecting STRING
protecting INT

- the actual net and its rules as initial state
- properties defined by reachable, t-enabled and enabled
- the actual rules as rewrite rules
- enabling and firing as rewrite rules
From reconfigurable Petri nets to Maude modules

- the actual net and its rules as initial state
- properties defined by reachable, t-enabled and enabled
- the actual rules as rewrite rules
- enabling and firing as rewrite rules
From reconfigurable Petri nets to Maude modules

- Maude module `NET`
  - **including** `PROP`
  - **including** `MODEL-CHECKER`

- Maude module `PROP`
  - **protecting** `STRING`
  - **including** `RULES`

- Maude module `RULES`
  - **including** `RPN`

- Maude module `RPN`
  - **protecting** `STRING`
  - **protecting** `INT`

- the actual net and its rules as initial state

- properties defined by `reachable, t-enabled` and `enabled`

- the actual rules as rewrite rules

- enabling and firing as rewrite rules
Sketch of Correctness Proof

Results from [Padberg, Schulze 14]

recPN in PNML $\xrightarrow{ReConNetModelChecker} \rightarrow$ Maude modules
Sketch of Correctness Proof

New Results

\[
\text{recPN in PNML} \xrightarrow{\text{ReConNetModelChecker}} \text{Maude modules}\]

\[
(N, R) \xrightarrow{\text{build-functions}} \text{rewrite theory } \text{NET}\]

Julia Padberg
HAW Hamburg
Verification of Reconfigurable Petri Nets
Sketch of Correctness Proof

New Results

$\text{recPN in PNML} \xrightarrow{\text{ReConNetModelChecker}} \text{Maude modules}$

$\text{imp}_{\text{recPN}} \xrightarrow{\text{Conversion}} \text{rewrite theory NET}$

$LTS_{\text{recPN}} \xrightarrow{\text{map}} LTS_{\text{RPN}}$
Sketch of Correctness Proof

Main Result

\[
\text{recPN in PNML} \xrightarrow{\text{ReConNetModelChecker}} \text{Maude modules}
\]

\[
(N, R) \xrightarrow{\text{Conversion}} \text{rewrite theory NET}
\]

\[
\text{LTS for recPN} \xrightarrow{\text{Bisimulation}} \text{LTS}_{RPN} \xrightarrow{\text{Bisimulation}} \text{LTS}_{MNC}
\]

\[
\text{LTS for Maude}
\]
Correctness of MC Approach

Theorem: Bisimulation

$LTS_{RPN}$ and $LTS_{MNC}$ are bisimilar.

Due to the surjective mapping $map$.

Corollary: LTL properties are preserved

For any LTL property $\phi$ we have:

$LTS_{RPN} \models \phi$ iff $LTS_{MNC} \models \phi$

Absence of deadlocks for $(N_{mixer}, R)$ proven by Maude

```
1| Maude 2.7 built: Aug 6 2014 22:54:44
2| Copyright 1997–2014 SRI International
3| Mon Sep 28 19:13:44 2015
4| rewrite in NET: modelCheck(initial, []<> enabled).
5| rewrites: 17601 in 25ms cpu (48ms real) (704040 rewrites/second)
6| result Bool: true
```
Correctness of MC Approach

Theorem: Bisimulation

$LTS_{RPN}$ and $LTS_{MNC}$ are bisimilar.

Due to the surjective mapping $map$.

Corollary: LTL properties are preserved

For any LTL property $\phi$ we have:

$LTS_{RPN} \models \phi$ iff $LTS_{MNC} \models \phi$

Absence of deadlocks for $(N_{mixer}, R)$ proven by Maude

```
1| Maude 2.7 built: Aug 6 2014 22:54:44
2| Copyright 1997–2014 SRI International
3| Mon Sep 28 19:13:44 2015
4| rewrite in NET : modelCheck(initial, []<> enabled).
5| rewrites: 17601 in 25ms cpu (48ms real) (704040 rewrites/second)
6| result Bool: true
```
Correctness of MC Approach

Theorem: Bisimulation

$LTS_{RPN}$ and $LTS_{MNC}$ are bisimilar.

Due to the surjective mapping $map$.

Corollary: LTL properties are preserved

For any LTL property $\phi$ we have:

$LTS_{RPN} \models \phi$ iff $LTS_{MNC} \models \phi$

Absence of deadlocks for $(N_{\text{mixer}}, R)$ proven by Maude

```
Maude 2.7 built: Aug 6 2014 22:54:44
Copyright 1997–2014 SRI International
Mon Sep 28 19:13:44 2015

rewrite in NET : modelCheck(initial, []<> enabled).
rewrites: 17601 in 25ms cpu (48ms real) (704040 rewrites/second)
result Bool: true
```
Semantics

Example in ReConNet
Semantics

State Diagram by Maude

state 0 : P3,P3
state 1 : P2,P3
T,4:P,3->P,2
T,5:P,3->P,2
state 2 : P2,P2
T,4:P,3->P,2
T,5:P,3->P,2
state 3 : P2,P2
T,4:P,3->P,2
state 4 : P2,P2
R1-PNML: d{T[T,4:P,3->P,2]} -> a{T[T,4:P,2->P,3]}
state 5 : P2,P3
T,4:P,2->P,3
state 6 : P2,P2
T,4:P,2->P,3
state 7 : P2,P3
T,5:P,2->P,3
state 8 : P3,P3
T,4:P,2->P,3
state 9 : P2,P3
T,4:P,2->P,3
T,5:P,2->P,3
T,4:P,3->P,2
state 10 : P3,P3
T,5:P,2->P,3
R1-PNML: d{T[T,4:P,2->P,3]} -> a{T[T,4:P,3->P,2]}
state 11 : P3,P3
T,4:P,2->P,3
T,5:P,2->P,3
T,5:P,3->P,2
R1-PNML: d{T[T,4:P,2->P,3]} -> a{T[T,4:P,3->P,2]}
(Potential) Tools for Reconfigurable Petri Nets

**General Graphtransformation Tools**

- AGG
- Groove

**Reconfigurable Net Tools**

- RON-Editor
- ReConNet
(Potential) Tools for Reconfigurable Petri Nets

General Graphtransformation Tools

- Groove

Model Checking supported by

Reconfigurable Net Tools

- ReConNet
Perservation of Invariants

- preservation of invariants [PGE01, Urb03]
- specific construction (reduction) rules, starting with [Mur89] many others
- verification the set of reached configurations [KBD16]
  editing and simulation of rules using RON
  verification of configurations using Tina

Adapt Model-Checking
Outline

- Motivation
- **Example: Dynamic Hardware Reconfiguration**
- Reconfiguration of Petri nets
- Model-checking reconfigurable place/transition nets with Maude
- Conclusion
Ongoing Work

Extensions

• concerning control structures
  negative application conditions, transformation units

• concerning hierarchy concepts
  transition refinement, place fusion

Extensions of the translation to Maude

• using Maude’s conditional rewrite rules

• via a flattening construction
Future Work

- further work on verification
- benchmarking the model checking using Graph transformation systems (e.g., Groove) using Petri nets (e.g., snoopy)
- representation of the Maude’s counterexamples in RECONNET
- extension to reconfigurable algebraic high-level nets data type – algebraic specification – Maude Model Checking reconfigurable AHL nets
Conclusion

Lots of theoretical results

still unused concepts from abstract transformation systems

- critical pair analysis
- nested application conditions
- graph (and net) properties

but not very well recognised, so

- stronger focus on real applications
- better tool support
- better availability

available via github
see https://reconnetblog.wordpress.com/
Conclusion

Lots of theoretical results
still unused concepts from abstract transformation systems
- critical pair analysis
- nested application conditions
- graph (and net) properties
but not very well recognised, so
- stronger focus on real applications
- better tool support
- better availability

available via github
see https://reconnetblog.wordpress.com/
Last but not least...

Thanks to the organizers
Last but not least...

Thank you for your attention
Bernard Berthomieu and François Vernadat.
Time petri nets analysis with TINA.

Allan Cheng, Søren Christensen, and Kjeld Mortensen.


Javier Esparza.
A false history of true concurrency: From petri to tools.

Bernd Grahlmann and Eike Best.
Pep—more than a petri net tool.

Laïd Kahloul, Samir Bourekkache, and Karim Djouani.
Designing reconfigurable manufacturing systems using reconfigurable object Petri nets.

Tadao Murata.
Petri nets: Properties, analysis and applications.

Rule-based refinement of high-level nets preserving safety properties.

Ulrike Prange.
Algebraic high-level nets as weak adhesive HLR categories.
*ECEASST, 2*, 2006.

Alexey A Tovchigrechko.
Marcie’s secrets of efficient model checking.

Milan Urbásek.
Preserving properties in system redesign: Rule-based approach.