Compositional model checking of concurrent systems, with Petri nets

Pawel Sobocinski, ECS, Southampton
DCM 2015

joint work with Julian Rathke and Owen Stephens
Methodology

- Compositional algebra of interconnected systems

- Once we have a **compositional** model, can use it for, amongst other things:
  - algorithmic improvements in verification through divide and conquer
  - parametric verification

- This talk: we focus on Elementary Net Systems (1-safe nets)

- cf. Jan C. Willems, modelling by Tearing, Zooming and Link
Why Petri nets?

• intuitive graphical syntax

• differently from vanilla transition systems, concurrency is **explicit**

  • enables various partial order reduction techniques, e.g. unfolding

• widely used, and conquering other sciences (systems biology, medicine, chemistry, …)
Why not Petri nets?

- often accused of being non-compositional
  - but many algebraic approaches exist, although:
    - sometimes they don’t have a compositional semantics
    - sometimes they are inconvenient for specifying “real systems”
  - exploiting CCS/CSP style operations to compose nets has been very fruitful (eg. Petri Box calculus)
    - the algebra of this talk is a related, but the operations are very different from CCS/CSP composition
Roadmap

• Introduction to Petri nets with boundaries
• Compositional reachability analysis
• Ongoing and future work
Alternative graphical syntax

• places are drawn with an **in-port** (triangle into the place) and an **out-port** (triangle out of the place)

• a transition is simply determined by the collection of ports it is connected to

• transitions are drawn with a small perpendicular mark to help legibility
Petri net with boundaries (PNB)

The most interesting operation on PNBs is synchronisation along a common boundary; we illustrate this operation in Fig. 3. In each of the examples, the size of the right boundary of the first net agrees with the size of the left boundary of the second net—this is a general requirement for composition to be defined: nets that do not agree on the size of their common boundary cannot be synchronised. Given nets $X:(k,l)$ and $Y:(l,m)$, their composition is denoted $X;Y:(k,m)$. In general, transitions of the composed net—called the minimal synchronisations—will be subsets of transitions of the individual component nets. We describe this operation informally with examples because the graphical presentation is quite intuitive. See the appendix for a formal treatment.

intuition: transitions connected to ports are thought of as incomplete-they are a partial view of some global synchronisation.

It is crucial not to think of ports as “input ports” and “output ports”.
The composition operations

• There are two operations for composing PNBs
  • synchronising composition (sometimes called sequential composition)
  • monoidal product

• Both are a kind of parallel composition — the first one with synchronisation, the second one without
Synchronising composition

\[
\begin{align*}
P &: (0, 2) & Q &: (2, 0) & P \cdot Q &: (0, 0) \\
P &: (0, 2) & R &: (2, 0) & P \cdot R &: (0, 0) \\
P &: (0, 2) & S &: (2, 0) & P \cdot S &: (0, 0) \\
P &: (0, 2) & T &: (2, 0) & P \cdot T &: (0, 0)
\end{align*}
\]
Monoidal product

The monoidal product \( \{ u, b \} \) is similar. In the top right quadrant, there are two separate transitions, \( c \) and \( d \), that can synchronize with \( t \). Both the choices are taken into account in the composed net and result in two different transitions \( \{ t, c \} \) and \( \{ t, d \} \), which intuitively mean that the transition \( t \) in the left net can synchronize in two different ways with transitions in the right net. The transition \( e \) does not connect to any places, only to the second boundary port. Thus, the corresponding synchronized transition \( \{ u, e \} \) has precisely the same pre and post set as transition \( u \).

In the bottom left quadrant, the transitions \( t \) and \( u \) are fused into a single transition after composition. In the final example, \( u \) has no complementary transition to synchronize with and thus no composite transition results.

The second operation for composing PNBs is called tensor. Graphically, it can be described as "stacking" one net over the other, and intuitively, it acts as a non-communicating parallel composition. Unlike synchronization along common boundary, any two nets can be tensored: given nets \( X : (k, l) \) and \( Y : (m, n) \), we have \( X \boxtimes Y : (k + m, l + n) \). A simple example is given in Fig. 4.

Petri nets with boundaries are the arrows of a prop.
Fig. 6: A token ring network as a PNB expression

are often cited by researchers and practitioners in support of working with Petri nets, rather than, for example, products of automata. One is qualitative: the graphical syntax results in vivid, intuitive and informative models of real concurrent and distributed systems. A more empirical, quantitative reason is that transition systems have a monolithic statespace that does not contain inherent information about concurrency. Instead, a state of a Petri net, i.e. a marking, has structure from which one can extract useful information. This leads to practical techniques for mitigating state explosion when model checking, e.g. partial order reduction [19] and symmetry-reduction [25], that would not be possible if working with mere transition systems.

Just as Petri nets can be evaluated into a transition system, forgetting the concurrency, a PNB expression can be composed into a Petri net, forgetting the spatial distribution. As we have shown, the close connection between the algebra and net geometry is a qualitative reason for working with PNB expressions. The information can also be exploited quantitatively [27] in order to improve the performance of model checking in suitable examples – the statespace of a PNB expression contains information both about concurrency (because the components are Petri nets) as well as spatial distribution.

2 A Language for Net Composition

In the previous section we demonstrated the algebraic description of Petri Net systems in terms of their component nets with boundaries. We now motivating a Domain Specific Language (DSL), PNBml, that evaluates to the algebra of PNB, but adds expressive high-level functional programming language features.

1.1 Specifying systems algebraically

The examples we have considered thus far have not been of practical interest, having been chosen for their simplicity in order to illustrate the basic operations of PNBs. We now show how a more interesting system can be expressed with the algebra. We will consider other realistic examples in §3.

Consider a model of simple token ring network, taken from [1], and illustrated in Fig. 5. Note that the (1-safe) net contains three identical components that differ only in their "internal state" (the local marking). Initially, only the leftmost component can proceed: after it finishes its internal computation it relinquishes its token, meaning that the next component can proceed. The modular structure of the system is made explicit with the algebra of PNBs, illustrated in Fig. 6, where we show how the system can be expressed formally as a collection of component PNBs, wired together appropriately with simple connector PNBs. Indeed, when the expression \((†)\) is evaluated by composing nets with boundaries, the resulting Petri net is isomorphic to the net in Fig. 5.

The example is an evocative illustration of the fact that the operations for composing PNBs are very closely linked to the underlying geometry of nets – the logical structure of the system can be seen by examining the structure of the algebraic expression.

1.2 Explicit spatial distribution

Using transition systems as a model of concurrency has a long history (see e.g. [3]). Indeed, the semantics of a Petri net is usually a transition system. Two reasons

The algebraic expression reflects the system communication topology

(we can see how the net is wired up from looking at the term!)

\[ D ; ((S ; T ; T) \otimes I) ; E \]

Fig. 6: A token ring network as a PNB expression
LTS semantics

• LTS labels indicate when a transition that is connected to a boundary port has been fired

This assignment is functorial (the LTS of a composed net is the composition of component LTSs)
Composing transition systems

\[
\begin{align*}
  P & \xrightarrow{\bar{a}} Q & R & \xrightarrow{\bar{c}} S \\
  \frac{}{P;R} & \xrightarrow{\bar{a}} Q;S
\end{align*}
\]

(CUT)

\[
\begin{align*}
  P & \xrightarrow{\bar{a}} Q & R & \xrightarrow{\bar{c}} S \\
  \frac{}{P \otimes R} & \xrightarrow{\bar{a}\bar{c}} Q \otimes S
\end{align*}
\]

(TEN)

e.g.

\[
\begin{array}{c}
\begin{array}{ccc}
0/0 & 0/1 & 0/0 \\
\rightarrow & \circ & \rightarrow \\
0 & 1 & 0
\end{array} & ; & \begin{array}{ccc}
0/0 & 0/1 & 0/0 \\
\rightarrow & \circ & \rightarrow \\
0 & 1 & 0
\end{array} = \begin{array}{ccc}
0/0 & 0/1 & 0/0 \\
\rightarrow & \circ & \rightarrow \\
0 & 1 & 0
\end{array}
\end{array}
\]
that these two LTSs are isomorphic. The resulting net is illustrated in Fig. 20a. Its semantics can be obtained directly by firing on the boundaries. Notice that we always have $\text{عش} = 0$ for $\text{عش}$.

Formally, the labelled semantics is defined as follows. Definition 7

Compositionality means reasoning about the behaviour of the net; alternatively, we could compose the er net and its semantics. For example, consider the bu er net composed with itself, as illustrated in Fig. 20b. Compositionality means er net composition and Bu er LTS composed with itself, as illustrated in Fig. 20b. Compositionality means er net composition and Bu er LTS composition as the composition of the er net and its semantics. For example, consider the bu er net composition and Bu er LTS composition.

For example, consider the bu er net composition and Bu er LTS composition. Compositionality means er net composition and Bu er LTS composition as the composition of the er net and its semantics. For example, consider the bu er net composition and Bu er LTS composition as the composition of the er net and its semantics. For example, consider the bu er net composition and Bu er LTS composition as the composition of the er net and its semantics.
Roadmap

• Introduction to Petri nets with boundaries

• Compositional reachability analysis

• Ongoing and future work
PNB Decomposition

Extend graphical syntax to show "target" places where we want a token in the final configuration.

\( \top ; b_1 ; b_1 ; b_1 ; b_1 ; \bot \)
Nondeterministic finite automaton semantics

- NFA captures the **interaction patterns** which allow the component to reach the *desired local marking*

This assignment is **functorial** (the NFA of a composed net is the composition of component NFAs)

**note**: reachability reduces to language emptiness
Theorem

- Weak language equivalence is a congruence wrt to composition operations
  - an internal move is the firing of transitions that are not connected to boundary ports
  - weak means regarding internal moves as tau-moves or epsilon-moves
  - up-to-weak-language-equivalence means that we can aggressively discard irrelevant local state
  - in essence, we only care about how a component net interacts
Example - buffer

The trivial accepting automaton is a fixed point of this process: this can also be seen as a proof of parametrised reachability for the buffer example!
Intuition: why does it work?

$N_0$ can reach desired local marking after firing $t_2$ twice, after which it can be fired an arbitrary additional number of times.

$N_1$ can reach desired local marking and fire $t_2$ an arbitrary number of times.
Implementation details

- Penrose tool: implemented by O. Stephens in Haskell, with almost no optimisation, but:

  - We try to keep automata small


- Memoisation is used to avoid re-minimising and re-compositing weak language equivalent automata


<table>
<thead>
<tr>
<th>name</th>
<th>size</th>
<th>LOLA</th>
<th>CLP</th>
<th>CNA</th>
<th>Penrose</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>8</td>
<td><em>0.001</em></td>
<td>0.003</td>
<td>0.017</td>
<td>0.002</td>
</tr>
<tr>
<td>buffer</td>
<td>32</td>
<td><em>0.001</em></td>
<td>0.013</td>
<td>0.824</td>
<td>0.002</td>
</tr>
<tr>
<td>buffer</td>
<td>512</td>
<td>0.058</td>
<td><em>T</em></td>
<td><em>M</em></td>
<td><em>0.002</em></td>
</tr>
<tr>
<td>buffer</td>
<td>4096</td>
<td><em>T</em></td>
<td><em>T</em></td>
<td><em>M</em></td>
<td><em>0.005</em></td>
</tr>
<tr>
<td>buffer</td>
<td>32768</td>
<td><em>T</em></td>
<td><em>T</em></td>
<td><em>M</em></td>
<td><em>0.029</em></td>
</tr>
</tbody>
</table>
Performance on standard benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size (tokens)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter-choice 8</td>
<td>32768</td>
<td>31.039 1.071 0.003 M T M 0.004 M T M 15.49</td>
</tr>
<tr>
<td>iter-choice 32</td>
<td>32768</td>
<td>15.50</td>
</tr>
<tr>
<td>iter-choice 512</td>
<td>32768</td>
<td>15.52</td>
</tr>
<tr>
<td>iter-choice 4096</td>
<td>32768</td>
<td>16.04</td>
</tr>
<tr>
<td>dac 8</td>
<td>4096</td>
<td>2.462 0.008 T T M 0.053 T T M 24.24</td>
</tr>
<tr>
<td>dac 32</td>
<td>4096</td>
<td>1.824 0.017 0.002 M T M 15.49</td>
</tr>
<tr>
<td>dac 512</td>
<td>4096</td>
<td>15.50</td>
</tr>
<tr>
<td>dac 4096</td>
<td>4096</td>
<td>15.92</td>
</tr>
<tr>
<td>philo 8</td>
<td>512</td>
<td>8.86 0.003 0.016 0.005 M 32768</td>
</tr>
<tr>
<td>philo 32</td>
<td>512</td>
<td>17.34 0.003 0.017 0.005 M 32768</td>
</tr>
<tr>
<td>philo 512</td>
<td>512</td>
<td>33.53 0.004 0.018 0.005 M 32768</td>
</tr>
<tr>
<td>philo 4096</td>
<td>512</td>
<td>17.58 0.004 0.019 0.005 M 32768</td>
</tr>
</tbody>
</table>

J. Rathke, P. Sobocinski and O. Stephens, *Compositional reachability in Petri nets, Reachability Problems ‘14*
Caveats

• Our tool takes in an algebraic decomposition as input

  • some nets do not allow efficient decompositions because of graph theoretic complexity (more on this later)

  • deriving efficient decompositions automatically is highly non-trivial

  • even after choosing a graph decomposition, the syntactic description is important e.g. **associativity matters!**

• But high-level system descriptions are the norm in **real** systems: e.g. our decompositions have followed Corbett’s high-level Ada descriptions very closely
Roadmap

• Introduction to Petri nets with boundaries
• Compositional reachability analysis
• Ongoing and future work
Graph theory

• With A. Chantawibul, working to understand the precise connection between graph theoretical metrics such as rank-width and the existence of “efficient decompositions”

  • rank-width (Seymour and Oum) is a measure that describes the complexity of connectivity in a graph

  • small rank-width of the underlying graph is a necessary condition for the existence of an efficient decomposition

  • it is not sufficient, there are also interesting semantic considerations
Axiomatisations

• All nets can be constructed from a small number of basic PNBs

\[
\begin{array}{cccc}
\Delta:1 \to 2 & \bot:1 \to 0 & \lor:2 \to 1 & \top:0 \to 1 \\
A:1 \to 2 & \downarrow:1 \to 0 & \lor:2 \to 1 & \uparrow:0 \to 1 \\
I:1 \to 1 & X:2 \to 2
\end{array}
\]

(A \bot \lor \top)

(\land \lor \lor)

• Can we obtain a complete axiomatisation in terms of these primitive nets? Complete axiomatisations for useful process equivalences?

• A similar approach has yielded complete axiomatisations for the operational semantics of signal-flow graphs:

• F. Bonchi, Sobocinski, F. Zanasi, A categorical semantics of signal flow graphs, CONCUR `14

• F. Bonchi, Sobocinski, F. Zanasi, Full abstraction for signal flow graphs, PoPL `15
Symmetric monoidal theories

• The algebra behind PNBs is a symmetric monoidal theory.

• roughly speaking, like classical algebraic theories, but linear: variables cannot be copied or discarded

• There is little current support for automated reasoning in symmetric monoidal theories
Parametric verification

- Much attention has recently been devoted to **parametric verification**, where systems that have a formal parameter can be verified for any value of the parameter
  
  - e.g. a token ring network of size $k$ - verify correct for any $k$

- As we’ve seen in the buffer example, the compositional approach seems to be suitable for this kind of analysis
Partial order reduction

- Improving the current tools for PNBs
- Is working with LTS/automata really a good idea?
  - concurrency is ignored…
- Can we combine compositionality with efficient techniques such as unfolding?
- Use SMT-solvers for performance?
Bibliography

P. Sobocinski, *Representations of Petri net interactions*, CONCUR `10


P. Sobocinski, *Nets, relations and linking diagrams*, CALCO `13


F. Bonchi, P. Sobocinski and F. Zanasi, Full abstraction for Signal Flow Graphs, PoPL `15