Unfolding-based Reachability Checking of Petri Nets

César Rodríguez

LIPN, Université Paris 13, France

Séminaire Spécification-Vérification
LIPN, Université Paris 13, October 9, 2014
Unfoldings: Symbolic Representations

- Compact, symbolic representation of concurrent state-space
- Originated from the partial-order semantics of Petri nets, 1970s-1980s
- Ken McMillan [CAV’92]: use them for practical verification
  - Finite, complete unfolding prefix for finite-state Petri nets
- Reachability, deadlock, LTL, . . .
Unfoldings: Symbolic Representations

- Compact, symbolic representation of concurrent state-space
- Originated from the partial-order semantics of Petri nets, 1970s-1980s
- Ken McMillan [CAV’92]: use them for practical verification
  - Finite, complete unfolding prefix for finite-state Petri nets
- Reachability, deadlock, LTL, . . .

Here we focus on

- Three semantics of Petri nets
- Unfolding structure and properties
- Unfolding construction and analysis (briefly)
Model Checking

- System
  - Modelling
    - System model
      - State-space exploration
        - Kripke structure $K$
          - Check whether $K \models \phi$
            - Counterexample / Correct
  - Formalization
    - Property to verify
      - Specification $\phi$
Coping with State-space Explosion

Explosion due to

- Concurrency
- Non-determinism
- Data
- Unsafeness...
Coping with State-space Explosion

Explosion due to
- Concurrency
- Non-determinism
- Data
- Unsafeness...

Alleviating state-space explosion

<table>
<thead>
<tr>
<th>Abstraction:</th>
<th>Aggregate similar states, by throwing away information and possibly repairing inaccuracies e.g., Abstract Interpretation, CEGAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduction:</td>
<td>Discard irrelevant states, by identifying equivalent computations and examining only one representative e.g., Partial-order reduction</td>
</tr>
<tr>
<td>Compression:</td>
<td>Use compact lossless representation, that handles many states at once without losing any of them e.g., BDDs, Unfoldings.</td>
</tr>
</tbody>
</table>
Coping with State-space Explosion

Explosion due to
- Concurrency
- Non-determinism
- Data
- Unsafleness...

Alleviating state-space explosion

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstraction:</strong></td>
<td>Aggregate similar states, by throwing away information and possibly repairing inaccuracies</td>
</tr>
<tr>
<td></td>
<td>e.g., Abstract Interpretation, CEGAR</td>
</tr>
<tr>
<td><strong>Reduction:</strong></td>
<td>Discard irrelevant states, by identifying equivalent computations and examining only one representative</td>
</tr>
<tr>
<td></td>
<td>e.g., Partial-order reduction</td>
</tr>
<tr>
<td><strong>Compression:</strong></td>
<td>Use compact lossless representation, that handles many states at once without losing any of them</td>
</tr>
<tr>
<td></td>
<td>e.g., BDDs, Unfoldings.</td>
</tr>
</tbody>
</table>

- **BDDs:** exploit regularity of homogeneous components
- **Unfoldings:** exploit concurrency of components
Check whether $K \models \phi$

Counterexample / Correct
Model Checking with Net Unfoldings

**Concurrent system**

Modelling

Petri Net

Unfolding construction

Complete prefix

Unfolding analysis

Counterexample / Correct

Property to verify

Formalization

Reachability / LTL
### Unfolding construction

- Initially proposed by Ken McMillan [McMillan 92]
- Size of the prefix reduced [Esparza, Römer, Vogler 96]
- Canonical prefixes [Khomenko, Koutny, Vogler 02]
- Comprehensive account [Esparza, Heljanko 08]

### Unfolding analysis

- Reachability and deadlock [McMillan 92], [Melzer, Römer 97], [Heljanko 99], [Khomenko, Koutny 00]
- LTL-X [Esparza, Heljanko 01]
Outline

1 Petri Nets

2 Non-sequential Semantics

3 Unfolding Semantics

4 Finite, Complete Prefixes

5 Summary
Petri Nets — Example

The circles are places
The shaded rectangles are transitions
The dots are tokens
The arrows are arcs
Petri Nets — Example

The

are places

The

are transitions

The

are tokens

The

are arcs

Allowed patterns:

Forbidden patterns:
Petri Nets — Example

The

are places

The

are transitions

The

are tokens

The

are arcs

Allowed patterns:

Forbidden patterns:
Petri Nets — Example

The \( \bigcirc \) are **places**
The \( \square \) are **transitions**
The \( \bullet \) are **tokens**
The \( \longrightarrow \) are **arcs**

**Allowed patterns:**

**Forbidden patterns:**
Petri Nets — Example

The are places
The are transitions
The are tokens
The are arcs

Allowed patterns:

Forbidden patterns:
Petri Nets — Example

The circles are places.
The rectangles are transitions.
The dots are tokens.
The arrows are arcs.

Allowed patterns:

Forbidden patterns:
A Petri net is a tuple $N := \langle P, T, F, m_0 \rangle$ such that

- $P$: finite set of places
- $T$: finite set of transitions
- $F \subseteq P \times T \cup T \times P$: flow relation
- $m_0: P \to \{0, 1\}$: initial marking

The preset and postset of a transition or place $x$ are:

**Preset:** $\bullet x := \{ y \in P \cup T : (y, x) \in F \}$

**Postset:** $x^\bullet := \{ y \in P \cup T : (x, y) \in F \}$
A marking of $N$ is a function $m : P \rightarrow \mathbb{N}$ that maps places to the number of tokens they contain.

- $m(\text{idle}_1) = 1$
- $m(\text{idle}_2) = 1$
- $m(\text{mutex}) = 1$
- $m(p) = 0$ for any other $p \in P$
A marking of $N$ is a function $m : P \rightarrow \mathbb{N}$ that maps places to the number of tokens they contain.

\[ m(\text{mutex}) = 2 \]
\[ m(\text{cs}_2) = 3 \]
\[ m(p) = 0 \text{ for any other } p \in P \]
A transition $t$ is enabled at a marking $m$ iff

$$m(p) \geq 1 \text{ for all } p \in \bullet t,$$

i.e., if the marking covers the preset of $t$.
A transition \( t \) is **enabled** at a marking \( m \) iff

\[
m(p) \geq 1 \quad \text{for all} \quad p \in \bullet t,
\]

i.e., if the marking covers the preset of \( t \).

**Diagram:**

Only \( \text{start}_1 \) and \( \text{start}_2 \) are enabled.
A transition $t$ enabled at marking $m$ can fire, producing a new marking $m'$, denoted as

$$m \xrightarrow{t} m'$$

where $m'$ is defined as

$$m'(p) = m(p) + \begin{cases} 
1 & \text{if } p \in t^\bullet \setminus \bullet t \\
-1 & \text{if } p \in \bullet t \setminus t^\bullet \\
0 & \text{otherwise}
\end{cases}$$

for all $p \in P$. 
A transition $t$ enabled at marking $m$ can fire, producing a new marking $m'$, denoted as

$$m \xrightarrow{t} m'$$

where $m'$ is defined as

$$m'(p) = m(p) + \begin{cases} 
1 & \text{if } p \in t^* \setminus t \\
-1 & \text{if } p \in t^* \setminus t \\
0 & \text{otherwise}
\end{cases}$$

for all $p \in P$. 
Let $N := \langle P, T, F, m_0 \rangle$ be a Petri net,

**Definition: operational semantics**

The operational semantics of $N$ is the edge-labelled transition system $M_N := \langle S, \Delta, s_0 \rangle$ defined as

- $S := \text{set of markings } m : P \rightarrow \mathbb{N}$ of $N$
- $\Delta := \{ \langle m, t, m' \rangle : \text{there is } t \in T \text{ such that } m \xrightarrow{t} m' \}$
- $s_0 := m_0$, the initial marking of $N$
Let \( N := \langle P, T, F, m_0 \rangle \) be a Petri net,

**Definition: operational semantics**

The operational semantics of \( N \) is the edge-labelled transition system

\[
M_N := \langle S, \Delta, s_0 \rangle
\]

defined as

- \( S := \) set of markings \( m: P \to \mathbb{N} \) of \( N \)
- \( \Delta := \{ \langle m, t, m' \rangle : \text{there is } t \in T \text{ such that } m \xrightarrow{t} m' \} \)
- \( s_0 := m_0, \) the initial marking of \( N \)

**Definition**

The reachability set of \( N \) is the smallest set \( \text{reach}(N) \) satisfying

1. \( m_0 \in \text{reach}(N) \)
2. if \( m \in \text{reach}(N) \) and \( m \xrightarrow{t} m' \), for any \( t \in T \), then \( m' \in \text{reach}(N) \).
Petri Nets — Operational Semantics: Example

mutex

start_1

idle_1

start_2

idle_2

waiting_1

enter_1

exit_1

waiting_2

enter_2

exit_2

cs_1

start

enter

start

start

idle

idle

enter

enter

exit

exit

waiting

waiting
Petri Nets — Operational Semantics: Example

\[
\begin{array}{c}
\text{exit} \quad (id_1 \text{ mut } id_2) \\
\text{start}_1 \quad (wa_1 \text{ mut } id_2) \quad (id_1 \text{ mut } wa_2) \\
\text{start}_2 \quad (wa_1 \text{ mut } wa_2) \quad (id_1 \text{ cs}_2) \\
\text{enter}_1 \quad (cs_1 \text{ id}_2) \quad (wa_1 \text{ cs}_2) \\
\text{start}_2 \quad (cs_1 \text{ wa}_2) \\
\text{exit}_1 \quad \text{exit}_2
\end{array}
\]
A run, or firing sequence of $N$ is any sequence of transitions

$$t_1 t_2 t_3 \ldots \in T^* \cup T^\omega$$

which labels at least one path

$$m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \xrightarrow{t_3} \ldots$$

in $M_N$ starting from the initial marking $m_0$. The set of runs of $N$ is denoted by $\text{runs}(N)$. 
## Petri Nets — Decidability and Complexity

<table>
<thead>
<tr>
<th></th>
<th>Bounded net</th>
<th>Unbounded net</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reachability</strong></td>
<td>PSPACE-complete</td>
<td>EXPSPACE-hard</td>
</tr>
<tr>
<td><strong>Coverability</strong></td>
<td>PSPACE-complete</td>
<td>EXSPACE-complete</td>
</tr>
<tr>
<td><strong>LTL model checking</strong></td>
<td>PSPACE-complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td><strong>Boundedness</strong></td>
<td>N/A</td>
<td>EXPSPACE-complete</td>
</tr>
</tbody>
</table>
Communicating Automata
### Concurrent Boolean Programs

L0:  
\[
a := 1;  \\
\text{while } (a) \ b := 0;  \\
goto \text{L0};
\]

L1:  
\[
b := 1;  \\
\text{while } (b) \ a := 0;  \\
goto \text{L1};
\]

---

#### Transition System

![Transition System Diagram](#)
Counter Abstractions

\[ x_1, x_2 \]

Shared variables (finite state)

Program (finite state, unbounded replication)

\[ x_2 := 0 \]
\[ x_1 := 0 \]
\[ [x_1 = 1] \]
\[ x_1 := \neg x_1 \]
\[ [x_2 = 1] \]
Outline

1 Petri Nets

2 Non-sequential Semantics

3 Unfolding Semantics

4 Finite, Complete Prefixes

5 Summary
State-Explosion: Concurrency

2^3 reachable markings
State-Explosion: Concurrency

- $2^3$ reachable markings
- And $2^n$ if $n$ processes instead of 3
Processes (or configurations) of a Petri Net

Events and conditions
Labelled, acyclic, and safe

Represents multiple interleavings of the same concurrent behaviour
Processes (or configurations) of a Petri Net

![Petri Net Diagram]

- `start_1`, `start_2`, `enter_1`
- `start_2`, `start_1`, `enter_1`
Processes (or configurations) of a Petri Net

\[
\begin{array}{c}
\text{start}_1, \text{start}_2, \text{enter}_1 \\
\text{start}_2, \text{start}_1, \text{enter}_1
\end{array}
\]
Processes (or configurations) of a Petri Net

start$_1$, start$_2$, enter$_1$
start$_2$, start$_1$, enter$_1$

Events and conditions
Labelled, acyclic, and safe
Represents multiple interleavings of the same concurrent behaviour
Processes (or configurations) of a Petri Net

Events and conditions

Labelled, acyclic, and safe
Represents multiple interleavings of the same concurrent behaviour

start$_1$, start$_2$, enter$_1$
start$_2$, start$_1$, enter$_1$
Processes (or configurations) of a Petri Net

\[
\text{start}_1, \text{start}_2, \text{enter}_1 \\
\text{start}_2, \text{start}_1, \text{enter}_1 \\
\text{start}_1, \text{enter}_1, \text{start}_2
\]
Processes (or configurations) of a Petri Net

Events and conditions
- Labelled, acyclic, and safe
- Represents multiple interleavings of the same concurrent behaviour
Structure of Processes

- Processes are acyclic, i.e., partial orders
- Associated to a (set of) run
- Every two events $e, e'$ are either
  1. Concurrent, denoted $e \parallel e'$, as copies of $\text{start}_1$ and $\text{start}_2$
  2. Causally related, denoted $e < e'$, as $\text{start}_1$ and $\text{enter}_1$
Petri nets — Non-sequential Semantics

The non-sequential semantics of $N$ is the set $conf(N)$ of all processes associated to the runs of $N$, i.e., $conf(N) := \{C_\sigma : C_\sigma$ is the process of some $\sigma \in runs(N)\}$

Each process is a Mazurkiewicz trace or a labelled partial order or...
Non-sequential Semantics

The non-sequential semantics of $N$ is the set $\text{conf}(N)$ of all processes associated to the runs of $N$, i.e.,

$$\text{conf}(N) := \{ C_\sigma : C_\sigma \text{ is the process of some } \sigma \in \text{runs}(N) \}$$

- Each process is a Mazurkiewicz trace or a labelled partial order or . . .
Outline

1. Petri Nets
2. Non-sequential Semantics
3. Unfolding Semantics
4. Finite, Complete Prefixes
5. Summary
What if we fuse common parts of multiple processes?
What if we fuse common parts of multiple processes?

- We get a branching process or unfolding prefix.
- Events may now be in conflict, denoted by $e \not\equiv e'$, as enter$_1$ and enter$_2$. 
What if we fuse common parts of multiple processes?

We get a branching process or unfolding prefix

Events may now be in conflict, denoted by $e \not= e'$, as enter$_1$ and enter$_2$
**Unfolding Semantics**

The unfolding $U_N$ is the net that results from fusing together the common parts of all configurations in $\text{conf}(N)$.

- **Acyclic and safe**
- **Labelling is a homomorphism**
- **Infinite in general**
**Remarks**

- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Inductive Definition — Example

Remarks
- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Inductive Definition — Example

Remarks

- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Inductive Definition — Example

Remarks

- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Inductive Definition — Example

Remarks

- $U_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Inductive Definition — Example

Remarks
- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Inductive Definition — Example

Remarks
- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Remarks

- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Inductive Definition — Example

Remarks

- $\mathcal{U}_N$ is acyclic, 1-safe
- Labelling is a homomorphism
- Infinite in general
- Finite, complete unfolding prefix
Let $N := \langle P, T, F, m_0 \rangle$ be a safe Petri net. The unfolding

$$\mathcal{U}_N := \langle B, E, G, D, \tilde{m}_0 \rangle$$

is the safe, acyclic net defined by:

- $p \in m_0$
  - $c = \langle \bot, p \rangle \in B$
  - $h(c) = p$
  - $c \in \tilde{m}_0$

- $t \in T$, $X \subseteq B$
  - $h(X) = \bullet t$
  - $X$ is coverable
  - $e = \langle X, t \rangle \in E$
  - $\bullet e = X$
  - $h(e) = t$

- $e \in E$
  - $h(e) = t$
  - $t^\bullet = \{p_1, \ldots, p_n\}$
  - $c_i = \langle e, p_i \rangle \in B$
  - $e^\bullet = \{c_1, \ldots, c_n\}$
  - $h(c_i) = p_i$

$h$ is a Petri net homomorphism.
Structural Relations

**Definition**

**Causality:** \( e < e' \) iff \( e' \) occurs \( \Rightarrow \) \( e \) occurs before

**Conflict:** \( e \# e' \) iff \( e \) and \( e' \) never occur in the same run

**Concurrency:** \( e \parallel e' \) iff not \( e < e' \) and not \( e' < e \) and not \( e \# e' \)
Configurations

A set of events \( C \) is a configuration iff:

1. \( e \in C \land e' < e \Rightarrow e' \in C \) (causally closed)
2. \( \neg e \neq e' \) for all \( e, e' \in C \) (conflict free)

Intuition: \( C \) configuration iff all its events can be sorted to form a run.
A set of events \( C \) is a configuration iff:

1. \( e \in C \land e' < e \Rightarrow e' \in C \) (causally closed)
2. \( \neg e \# e' \) for all \( e, e' \in C \) (conflict free)

Intuition: \( C \) configuration iff all its events can be sorted to form a run.
Outline

1. Petri Nets
2. Non-sequential Semantics
3. Unfolding Semantics
4. Finite, Complete Prefixes
5. Summary
\( \mathcal{U}_N \) is the result of unfolding ‘as much as possible’

Finite unfolding prefix \( \mathcal{P}_N \) results if you stop construction
\( U_N \) is the result of unfolding ‘as much as possible’

Finite unfolding prefix \( P_N \) results if you stop construction

If \( N \) has finitely many reachable markings...
\( U_N \) is the result of unfolding ‘as much as possible’

Finite unfolding prefix \( \mathcal{P}_N \) results if you stop construction

**Definition**

Prefix \( \mathcal{P}_N \) is marking-complete if:

for all marking \( m \) reachable in \( N \), there is marking \( \tilde{m} \) reachable in \( \mathcal{P}_N \) such that

\[
h(\tilde{m}) = m.
\]

If \( N \) has finitely many reachable markings...
Verification with Unfoldings: Finite, Complete Prefixes

- $U_N$ is the result of unfolding ‘as much as possible’
- Finite unfolding prefix $P_N$ results if you stop construction

**Definition**

Prefix $P_N$ is marking-complete if:

For all marking $m$ reachable in $N$, there is marking $\tilde{m}$ reachable in $P_N$ such that $h(\tilde{m}) = m$.

If $N$ has finitely many reachable markings...

- Some finite and marking-complete $P_N$ exists
- $P_N$: symbolic representation of reachability graph
- Reachability of $N$ is:
  - PSPACE-complete in $N$
  - NP-complete in $P_N$
  - Linear in reachability graph
Unfoldings Cope with Concurrency

- $2^3$ reachable markings
- And $2^n$ if $n$ processes
Unfoldings Cope with Concurrency

- $2^3$ reachable markings
- And $2^n$ if $n$ processes
- Unfolding is of linear size
Cutoff Events

Pruning the unfolding

An event $e$ is a cutoff if either there is an event $e'$ such that

- $|[e']| < |[e]|$ and
- $\text{mark}([e]) = \text{mark}([e'])$.

Remarks

- Requires building prefixes breadth-first
- Cutoff criteria relates to completeness
- Proposed by McMillan; improved by Esparza et al., among others
Let $P_N$ be a complete unfolding prefix of $N$:

- The reachability problem in $P_N$ can be solved in polynomial time
- Every reachable marking of $P_N$ is labelled by a marking reachable in $N$
- All markings of $N$ are represented in $P_N$

So given $P_N$ and a marking $m$ of $N$, checking whether $m$ is reachable in $N$ is NP-complete in $P_N$

- Reductions to SAT, linear programming, stable models, ... 
- Analysis time generally much smaller than unfolding time
Given a set of places $M$ of the net, generate

- $\phi^{{reach, M}}$ satisfiable iff places $M$ reachable in $N$
- Encodes existence of a configuration (partially-ordered run) that marks $M$
### Partial-order Reduction vs Unfoldings

<table>
<thead>
<tr>
<th>Underlying structure</th>
<th>Partial-order reduction</th>
<th>Unfoldings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interleavings</td>
<td>Partial order</td>
</tr>
<tr>
<td>Idea</td>
<td>Discard equivalent states</td>
<td>Compress equivalent states</td>
</tr>
<tr>
<td>Independence</td>
<td>Conditional, unconditional</td>
<td>Unconditional</td>
</tr>
<tr>
<td>Local reachability</td>
<td>Linear time</td>
<td>Linear time</td>
</tr>
<tr>
<td>Global reachability</td>
<td>in NP</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Deadlock</td>
<td>Linear time</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Mainstream</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Unfoldings applicable to other models of concurrency:

- Process algebras
- Communicating automata
- Concurrent boolean programs
- High-level nets
- Unbounded nets
- Nets with read arcs
- Time Petri nets
- Programs
- ...

César Rodríguez (Paris 13)
Summary

- Compact representation of a finite, **concurrent** state spaces
- Structure, properties, and construction of unfoldings
- Reachability analysis: based on SAT or on-the-fly
- Applicable to **other formalisms** with notion of independence
Summary

- Compact representation of a finite, concurrent state spaces
- Structure, properties, and construction of unfoldings
- Reachability analysis: based on SAT or on-the-fly
- Applicable to other formalisms with notion of independence

Unfoldings do not address other sources of explosion:
- Non-deterministic choices (→ merged processes)
- Concurrent read access (→ contextual unfoldings)
- Data (→ abstract interpretation, work with Oxford)