True Concurrency and Net Unfoldings

Daniil Frumin

December 9, 2013
Talk overview

1. Introduction
2. Unfoldings
3. Verification with unfoldings
4. Other developments in the area
5. Beyond unfoldings & conclusion
6. References and bibliography
1. Introduction
2. Unfoldings
3. Verification with unfoldings
4. Other developments in the area
5. Beyond unfoldings & conclusion
6. References and bibliography
Q: What is true concurrency?

A: True concurrency is a concurrency that we cannot represent using interleavings. True concurrency semantics are those that respect true concurrency.
Q: What is true concurrency?
A: It’s a concurrency that we can’t represent using interleavings.
Q: What is *true concurrency semantics*?
Q: What is true concurrency?
A: It’s a concurrency that we can’t represent using interleavings.

Q: What is true concurrency semantics?
A: It is semantics that respect true concurrency.
True concurrency semantics (CCS)

Interleaving world:

\[ a \parallel b \approx a.b + b.a \]

Non-interleaving world:

\[ a \parallel b \not\approx a.b + b.a \]

Figure 1: \( a.b + b.a \)
True concurrency semantics (CCS)

Calculus of communicating systems [Milner, 1989, Aceto et al., 2005]

Usual process calculi semantics

\[
\begin{align*}
    P & \xrightarrow{a} P' \\
    & \quad \Rightarrow \\
    P \parallel Q & \xrightarrow{a} P' \parallel Q
\end{align*}
\]

\[
\begin{align*}
    Q & \xrightarrow{a} Q' \\
    & \quad \Rightarrow \\
    P \parallel Q & \xrightarrow{a} P \parallel Q'
\end{align*}
\]

Non-interleaving semantics

Additional rule breaks strong bisimulation:

\[
\begin{align*}
    P & \rightarrow P' \\
    Q & \rightarrow Q' \\
    & \quad \Rightarrow \\
    P \parallel Q & \rightarrow P' \parallel Q'
\end{align*}
\]
Issues that programmers/users are facing

Problems that arise in (true) concurrent environments

Race conditions
- Bad interleavings
- Data races

Real-world example

“Multicore CPUs move attack from theoretical to practical” by Peter Bright
Topics in true concurrency

- True concurrency semantics of process algebras
- Axiomatic concurrency theory
- Trace theory
- Simulation relations in the presence of true concurrency
- Logics for true concurrency
- Unfoldings theory
- Partial order model checking

“A False History of True Concurrency” [Esparza, 2010]
Topics in true concurrency

- True concurrency semantics of process algebras
- Axiomatic concurrency theory
- Trace theory
- Simulation relations in the presence of true concurrency
- Logics for true concurrency
- Unfoldings theory
- Partial order model checking

“A False History of True Concurrency” [Esparza, 2010]
Talk overview

1. Introduction
2. Unfoldings
3. Verification with unfoldings
4. Other developments in the area
5. Beyond unfoldings & conclusion
6. References and bibliography
Net unfoldings is a popular true concurrency semantics for many computational models. Original development due to [Nielsen et al., 1981] (the term used: "event structures"). The authors also established a connection between true concurrency semantics for Petri nets and Scott’s domain theory. More information on event structures, domain theory and relations to other models of concurrency: [Winskel and Nielsen, 1993].
Unfolding a transition system

We can “unfold” a finite state machine into a computational tree.

Figure 2: State machine $SM_1$

Figure 3: Unfoldings of the state machine $SM_1$
Figure 4: P/T net $N_1$

Figure 5: Unfoldings of the net $N_1$
Occurrence nets (relations on nodes)

Let $N = (P, T, F)$ be a Petri net. We call the set $P \cup T$ the set of nodes. Abusing the notation we will write $x \in N$ to denote $x \in P \cup T$.

- $<$ — the causal relation: irreflexive transitive closure of $F$;
- $\#$ — the conflict relation:
  \[ x \# y \iff \exists t, t' \in E. t \neq t', \operatorname{pre}(t) \cap \operatorname{pre}(t') \neq \emptyset \land t \leq x \land t' \leq y; \]
- $co$ — the concurrency relation:
  \[ x \ co y \iff \neg(x < y) \land \neg(y < x) \land \neg(x \# y). \]
Figure 6: Causally dependent nodes
Relations on nodes: conflict

Figure 7: Nodes in conflict
Relations on nodes: concurrency

Figure 8: Concurrent nodes
Occurrence nets (definition)

Occurrence net $N = (B, E, F)$ ($B$ – conditions, $E$ – events)

- $N$ is acyclic;
- $\forall p \in B, |\text{pre}(p)| \leq 1$;
- $\forall x \in N$ the set $\{x' | x' < x\}$ is finite (it is said that every node has a finite number of predecessors);
- $\forall x \in N, \neg (x \# x)$, e.g. no node is in self-conflict.
Occurrence nets (properties of relations)

Some properties of the mentioned relations:\(^1\):

### 3 relations “cover” the whole net
Each to nodes are either concurrent, xor causally dependend, xor in conflict.

### General properties
- \(\leq\) is a (partial) order;
- \# and co are symmetric;
- \# “plays well” with \(<\): if \(x\#y\) and \(x \leq x' \land y \leq y'\) then \(x'\#y'\).

---

\(^1\)Some formalized proofs can be found at [http://me.hskll.org/repos/coq/OccurrNet.html](http://me.hskll.org/repos/coq/OccurrNet.html)
Let $N_1 = (B, E, \text{pre}_1, \text{post}_1)$, $N_2 = (P, T, \text{pre}_2, \text{post}_2)$ be Petri nets. 
$h : N_1 \rightarrow N_2$ is called a net morphism iff

1. $h(B) \subseteq P$, $h(E) \subseteq T$;
2. For each $e \in E$: $h(\text{pre}_1(e)) = \text{pre}_2(h(e))$ and $h(\text{post}_1(e)) = \text{post}_2(h(e))$.

Additionally, for nets with initial markings (sometimes referred to as net systems) we require that $h$ preserves initial markings. It is possible to check that this definition is “sound” (composition of two morphisms is a morphism; nets with morphisms form a category $\text{Petri}$).
A branching process (originally due to [Engelfriet, 1991]) for a net $N$ is a tuple $BP = (O, h)$ where

1. $O = (B, E, pre, post)$ — occurrence net;
2. $h : O \rightarrow N$ — net morphism;
3. Additionally for an initial marking $M_I$ of $N$ we identify a set of *starter/initial conditions* of $I \subseteq B$ s.t. $I$ is an initial marking of $O$ (consequently $h(I) = M_I$) and $I$ is the set of *causally minimal*, i.e. $\forall s \in I. |pre(s)| = 0$;
4. For all $e, e' \in E$ if $pre(e) = pre(e')$ and $h(e) = h(e')$ then $e = e'$.
Branching processes (inductive definition)

Alternatively, we can give a constructive definition.  

A set of branching processes (for a net $N$) is the smallest set satisfying the following conditions:

1. Let $I = \{i_p \mid p \in M_0\}$, $h(i_p) = p$. $((I, \emptyset, \emptyset), h)$ is a branching process; (induction base, a net with only a handful of conditions and no events)

2. Let $BP = ((B, E, F), h)$ be a branching process. Let $t$ be a new* transition of $N$, s.t. for some $P \subseteq B$, $h(P) = pre(t)$. Then $BP' = ((B', E', F'), h')$ is a branching process, where
   - $E' = E \cup \{e_t\}$
   - $B' = B \cup \{b_p \mid p \in post(t)\}$ (where each of $b_p$ is “fresh”)
   - $h'$ is an extension of $h$, s.t. $h(e_t) = t$, $h(b_p) = p$

*new meaning that there are no events in $BP$ that satisfy $pre(e) = P$. This is also called a redundancy rule, same as item 4 in the previous definition.

3. Let $S$ be a (finite or infinite) set of branching processes. Then $\bigcup S$ is a branching process if all branching processes in $S$ can be composed in “good” way (e.g. union of two does no introduce redundancies, initial conditions coincide).

---

2Slightly modified version of what is presented in [Esparza and Heljanko, 2008].
Examples

Figure 9: P/T net $N_1$

Figure 10: Branching process $BP_1$ for the net $N_1$
Examples

Figure 11: P/T net $N_1$

Figure 12: Branching process $BP_2$ for the net $N_1$
Examples

Figure 13: P/T net $N_1$

Figure 14: Branching process $BP_3$ for the net $N_1$
Branching processes are subject to *prefix relation*: $A \sqsubseteq B$ if there is an injective homomorphism from $A$ to $B$ (we can view it as if $A$ is a prefix/subnet of $B$ up to isomorphism\(^3\)). A $\sqsubseteq$-maximal\(^4\) branching process is called an *unfolding* of a net and denoted as $U(N)$.

---

\(^3\)Intuitively, “up to renaming”

\(^4\)Existence guaranteed by Zorn’s lemma
Net unfoldings (uniqueness)

**Theorem**

*Net unfoldings are unique (up to isomorphism).*

**Proof sketch.**

It can be shown that branching processes form a complete lattice wrt to \(\sqsubseteq\) by picking up a *canonical representation* of branching processes for a particular net. In that setting \(\sqsubseteq\) coincides with \(\subseteq\) and union of a family of branching processes in a canonical representation is itself a branching process in a canonical representation. The upper bound of a set of branching processes \(Bs = \{S_i \mid i \in \text{Ind}\}\) then is simply \(\bigcup Bs\). See [Engelfriet, 1991] for more details.
Net unfoldings (fundamental property)

Theorem (Fundamental property of unfoldings)

Let $N$ be a $P/T$-net, let $M$ be a reachable marking of $U(N)$, s.t. $h(M) = \mu$ then

1. If $M \xrightarrow{a} M'$ in $U(N)$, then $\mu \xrightarrow{h(a)} h(M')$ in $N$;
2. If $\mu \xrightarrow{t} \mu'$ in $N$, then $M \xrightarrow{a} M'$ in $N$ where $h(M') = \mu'$ and $h(a) = t$.

Intuitively, this means that unfolding possesses the same behavioral properties that original net has.
Proof sketch.

The theorem can be proved using induction on the length of the fireable sequence $\sigma$.

1. In case of $\sigma = \epsilon$ – obvious

2. In case of $\sigma = \sigma't$ we have (by the induction hypothesis) $\mu_0[\sigma']\mu_1$, $M_0[\psi]M_1$, $h(\psi) = \sigma' \land h(M_1) = \mu_1$. Since $t$ is active $\text{pre}(t) \subseteq \mu_1 \implies \text{pre}(t) \subseteq h(M_1)$. Then $\text{pre}(t) = h(M_1')$ for some $M_1' \subseteq M_1$. Then $U(N)$ contains an event $e$ s.t. $\text{pre}(e) = M_1'$ and $h(e) = t$. If it wasn’t the case, than $U(N)$ wouldn’t be the maximal branching process.
Talk overview

1. Introduction
2. Unfoldings
3. Verification with unfoldings
4. Other developments in the area
5. Beyond unfoldings & conclusion
6. References and bibliography
Finite prefixes: battling the state space explosion problem

Figure 15: State space explosion, common in highly concurrent systems
Verification with finite prefixes

We can use *finite prefixes* of unfoldings to solve a number of verification problems:

- Reachability
- Coverability
- Fireability of a transition
- Deadlock freedom
- Mutex
- Etc
Preliminaries: configurations and cuts

Definition

A *configuration* of a branching process is a set $C \subseteq E$ s.t. for all $e \in C$

- $\forall e' < e . e' \in C$, i.e. $C$ is *downwards closed* w.r.t. $<$;
- $\forall e' \in C . \neg(e' \# e)$, i.e. $C$ is *conflict-free*.

For each event $e$ we can define a *local configuration* $\text{Conf}(e) = \{ e' \mid e' \leq e \}$
Preliminaries: configurations and cuts

Definition

A configuration of a branching process is a set $C \subseteq E$ s.t. for all $e \in C$

- $\forall e' < e . e' \in C$, i.e. $C$ is downwards closed w.r.t. $<$;
- $\forall e' \in C . \neg (e' \# e)$, i.e. $C$ is conflict-free.

For each event $e$ we can define a local configuration $\text{Conf}(e) = \{ e' | e' \leq e \}$

Definition

A set $B'$ is called a cut if it’s a maximal (w.r.t $\subseteq$) set of conditions that satisfies $\forall x, y \in B'. x \text{ co } y$.

- Cuts characterizes reachable markings;
- Each configuration induces a cut: $\text{Cut}(C) = (\text{Min} \cup \text{post}(C)) \setminus \text{pre}(C)$, where Min is the set of $<$-minimal nodes of a branching process (i.e. the initial marking, starting nodes, $h(M_0)$).
A prefix of the unfolding of a net $N$ is said to be *marking-complete* if for every reachable marking $M$ of $N$ there exists a configuration $C$, s.t. $h(Cut(C)) = M$. 
Constructing finite prefixes, McMillan algorithm

Constructing a finite prefix for the net $N$ (originally by [McMillan, 1993]).

1. Start with an net $U$, that contains only the initial marking of $N$ and an empty set of terminal events $T$.

2. Create a queue $Q$ that contains possible extensions of $U$, i.e. events $e$ such that $\text{pre}(e)$ is already in $U$ and elements of $\text{pre}(e)$ are pairwise concurrent.

3. Grab an element $t$ from the queue, prioritized by the size of the local configuration. Add $t$ and $\text{post}(t)$ to the branching process $U$. If $t$ is a cut-off point, then add $t$ to the set $T$ of terminal events/cut-off nodes.

4. Generate more possible extensions, ignoring nodes $x$ s.t. $\exists t \in T. t < x$. Add possible extensions to the queue.

5. Repeat while $Q$ is non-empty.

Node $e$ is called a cut-off point iff there is another event $e'$ such that $h(\text{Cut}(e')) = h(\text{Cut}(e))$ (i.e. they transition to the same markings) and $|\text{Cut}(e')| < |\text{Cut}(e)|$. 
A net $N$ contains a deadlock$^5$ iff $U(N)$ has a deadlock; 

$U(N)$ contains a deadlock iff a marking-complete prefix of $U(N)$ contains a configuration from which it is impossible to reach a configuration, containing a cut-off point; 

i.e. if there is a configuration which is in conflict with every cut-off node in the prefix.
Checking for deadlock with SAT-solvers

We can produce the formula $\psi$ that corresponds to the configurations of a (complete) prefix $BP$. Each satisfactory assignment of $\psi$ determines a valid configuration in $BP$. Variable $e$ is true iff the event $e$ has occurred in $BP$. $\psi$ consists of formulae $\psi_e$ for each event $e$:

$$\psi_e = \bigwedge_{f \in \text{pre}(\text{pre}(e))} (e = \Rightarrow f) \land \bigwedge_{f \neq e} (\neg e \lor \neg f) \land \bigwedge_{e \text{ is a cut-off event}} (\neg e)$$
Checking for deadlock with SAT-solvers

We can produce the formula $\psi$ that corresponds to the configurations of a (complete) prefix $BP$.
Each satisfactory assignment of $\psi$ determines a valid configuration in $BP$. Variable $e$ is true iff the event $e$ has occurred in $BP$. $\psi$ consists of formulae $\psi_e$ for each event $e$:

$$
\psi_e = \bigwedge_{f \in \text{pre}(\text{pre}(e))} (e \implies f) \land \bigwedge_{f \neq e} (\neg e \lor \neg f) \land \bigwedge_{e \text{ is a cut-off event}} (\neg e)
$$
Checking for deadlock with SAT-solvers

\[ \psi_e = \bigwedge_{f \in \text{pre}(\text{pre}(e))} (e \implies f) \land \bigwedge_{f \neq e} (\neg e \lor \neg f) \land \bigwedge_{e \text{ is a cut-off event}} (\neg e) \]

Figure 16: \( e \implies f \)

Figure 17: \( \neg e \lor \neg f \)
Checking for deadlock with SAT-solvers

A place $p$ is marked (where $e' = pre(p)$):

$$marked(p) = (\bigwedge_{e \in post(p)} \neg e) \land e'$$

We can construct a formula $enables(t)$ for each transition $t$ in the original net that is true iff the configuration enables a transition labeled with $t$.

$$enables(t) = \bigwedge_{p \in pre(t)} \bigvee_{h(b) = p} marked(b)$$
Checking for deadlock with SAT-solvers

A place $p$ is marked (where $e' = pre(p)$):

$$marked(p) = (\bigwedge_{e \in post(p)} \neg e) \land e'$$

We can construct a formula $enables(t)$ for each transition $t$ in the original net that is true iff the configuration enables a transition labeled with $t$.

$$enables(t) = \bigwedge_{p \in pre(t) \ h(b) = p} \bigvee \text{marked}(b)$$

Finally, we can construct a formula that is satisfiable iff there is no deadlock in the net

$$\psi \iff (enables(a) \lor \cdots \lor enables(z))$$

where $\{a, \ldots, z\}$ is the set of transitions of the net $N$. 
The problem of generating possible extensions of a branching process is NP-complete (can be proved via reduction from SAT) [Esparza and Heljanko, 2008, Heljanko, 1999].

Figure 18: Synchronized product for (a) variable $x_1$ (b) literal $x_1$ in clause $x_1 \lor x_2$ (c) clause $x_1 \lor x_2$ in formula $(x_1 \lor x_2) \land \overline{x_1}$; taken from from [Esparza and Heljanko, 2008]
Deadlock checking is NP-complete (in the size of the prefix; [McMillan, 1995], also see previous case), marking reachability using finite prefixes is also NP-complete.
Model checking is PSPACE-complete. [Heljanko, 2000]
It has been noted that McMillan’s algorithm can generate prefixes bigger than needed.

**Figure 19:** Net $N_2$

**Figure 20:** Finite prefix of $N_2$ according to the McMillan’s algorithm
It has been noted that McMillan’s algorithm can generate prefixes bigger than needed.

Figure 19: Net $N_2$

Figure 20: Finite prefix of $N_2$ according to the McMillan’s algorithm
Cut-Off criterion and adequate orders are used to abstract the way we handle terminal/cut-off events.

**Definition (Cut-off event)**

We define $Mark(C) = h(Cut(C))$. Event $e$ is called a cut-off event iff there is a configuration $C$ already present in a branching process, such that $Mark(C) = Mark([e])$ and $C \prec [e]$, where $\prec$ is an adequate order.
Definition (Adequate order)

A partial order $\prec$ on the set of configurations of an unfolding is called adequate [Esparza et al., 1996] iff

- $\prec$ is well-founded (i.e. for each set of configurations there exists a $\prec$-minimal one);
- $\prec$ refines set inclusion: $C \subseteq C' \implies C \prec C'$;
- $\prec$ is preserved by finite extensions: if $\text{Mark}(C) = \text{Mark}(C')$ and $C \prec C'$ then $C \oplus E \prec C \oplus I(E)$ where $E$ is a suffix of $C$, $\oplus$ is a net concatenation operator, and $I(E)$ is an image of $E$ under “natural” isomorphism.
Old algorithm

Constructing a finite prefix for the net $N$.

1. Start with an net $U$, that contains only the initial marking of $N$ and an empty set of *terminal events* $T$.

2. Create a queue $Q$ that contains *possible extensions* of $U$, i.e. events $e$ such that $\text{pre}(e)$ is already in $U$ and elements of $\text{pre}(e)$ are pairwise concurrent.

3. Grab an element $t$ from the queue, prioritized by the size of the local configuration. Add $t$ and $\text{post}(t)$ to the branching process $U$. If $t$ is a *cut-off point*, then add $t$ to the set $T$ of terminal events/cut-off nodes.

4. Generate more possible extensions, ignoring nodes $x$ s.t. $\exists t \in T. t < x$. Add possible extensions to the queue.

5. Repeat while $Q$ is non-empty.
New algorithm

Constructing a finite prefix for the net $N$.

1. Start with an net $U$, that contains only the initial marking of $N$ and an empty set of terminal events $T$.

2. Create a queue $Q$ that contains possible extensions of $U$, i.e. events $e$ such that $\text{pre}(e)$ is already in $U$ and elements of $\text{pre}(e)$ are pairwise concurrent.

3. Grab an element $t$ from the queue, prioritized by the relation on events induced by $\prec$, i.e. choose $e$ over $e'$ if $[e] \prec [e']$. Add $t$ and $\text{post}(t)$ to the branching process $U$. If $t$ is a cut-off point according to $\prec$, then add $t$ to the set $T$ of terminal events/cut-off nodes.

4. Generate more possible extensions, ignoring nodes $x$ s.t. $\exists t \in T. t \prec x$. Add possible extensions to the queue.

5. Repeat while $Q$ is non-empty.
The completeness of the algorithm

The algorithm is correct in the sense that for every adequate order $\prec$ it produces a marking-complete prefix. Good explanation is presented in [Esparza and Heljanko, 2008].
Examples of adequate orders

- McMillan’s original order: \( C \prec C' \iff |C| < |C'| \)

- ERV order: Defined as following. Let \( \prec_{lex} \) be a lexicographical order on set of sequences of transitions; we can “lift” \( \prec_{lex} \) to the set of configurations by declaring \( C \prec_{lex} C' \) iff \( \text{flat}(C) \prec_{lex} \text{flat}(C) \) where \( \text{flat}(C) \) is a sequence of transitions ordered by \( \prec_{lex} \) and contains transition \( t \) as often as there are events in \( C \) labeled with \( t \).

  \( C \prec C' \iff \)

  - \( |C| < |C'| \);
  - or if \( |C| = |C'| \) and \( C \prec_{lex} C' \);
  - or if \( |C| = |C'| \), \( \text{flat}(C) = \text{flat}(C') \), and
    - \( \text{Min}(C) \prec_{lex} \text{Min}(C') \);
    - or \( \text{flat}(\text{Min}(C)) \prec_{lex} \text{flat}(\text{Min}(C')) \) and \( C \setminus \text{Min}(C) \prec C \setminus \text{Min}(C') \)

\( \text{Min}(C) \) – the set of minimal (wrt the causal ordering) nodes of \( C \).
Examples of adequate orders

- McMillan’s original order: \( C \prec C' \iff |C| < |C'| \)
  Is not a total order.

- ERV order: Defined as following. Let \( <_{\text{lex}} \) be a lexicographical order on set of sequences of transitions; we can “lift” \( <_{\text{lex}} \) to the set of configurations by declaring \( C <_{\text{lex}} C' \iff \text{flat}(C) <_{\text{lex}} \text{flat}(C) \) where \( \text{flat}(C) \) is a sequence of transitions ordered by \( <_{\text{lex}} \) and contains transition \( t \) as often as there are events in \( C \) labeled with \( t \). \( C \prec C' \iff \)
  \begin{itemize}
  \item \( |C| < |C'| \);
  \item or if \( |C| = |C'| \) and \( C <_{\text{lex}} C' \);
  \item or if \( |C| = |C'| \), \( \text{flat}(C) = \text{flat}(C') \), and
    \begin{itemize}
    \item \( \text{Min}(C) <_{\text{lex}} \text{Min}(C') \);
    \item or \( \text{flat}(\text{Min}(C)) <_{\text{lex}} \text{flat}(\text{Min}(C')) \) and \( C \setminus \text{Min}(C) \prec C \setminus \text{Min}(C') \)
    \end{itemize}
  \end{itemize}

\( \text{Min}(C) \) – the set of minimal (wrt the causal ordering) nodes of \( C \).
Is a total order for 1-safe nets [Esparza et al., 1996].

Total orders are good, allow us to have more cut-off events.
Talk overview

1. Introduction
2. Unfoldings
3. Verification with unfoldings
4. Other developments in the area
5. Beyond unfoldings & conclusion
6. References and bibliography
Infinite executability problem

Many problems can be solved using the complete finite prefixes that were presented

- Reachability
- Coverability
- Fireability of a transition
- Deadlock freedom
- Mutex
- Etc
Infinite executability problem

Many problems can be solved using the complete finite prefixes that were presented

- Reachability
- Coverability
- Fireability of a transition
- Deadlock freedom
- Mutex
- Etc

Some problems still cannot be solved using such prefix. Infinite executability problem?
Let \( \#_r(C) \) denote the number of events from \( C \) labeled by transition \( r \).

**Definition (Cut-off criterion for repeated executability problem)**

Event \( e \) is considered to be *terminal* iff there exists an event \( e' \prec e \) such that \( \text{Mark}([e']) = \text{Mark}([e]) \) and either

1. \( e' < e \) or
2. \( \#_r([e']) \geq \#_r([e]). \)
Arbitrary properties (expressed in LTL) can be checked using unfoldings: [Couvreur et al., 2000, Esparza and Heljanko, 2001].
Cutting context [Khomenko, 2003] is a generalization that allows us to preserve only the properties we want when constructing a finite prefix.

$$\Theta = (\approx, \prec, \{C_e\}_{e \in E})$$

1. $\prec$ – adequate order;
2. $\{C_e\}_{e \in E}$ – family of (finite) configurations of the unfolding (usually only local configurations);
3. $\approx$ – equivalence relation on the set of finite configurations of the unfolding.
4. $\approx$ and $\prec$ preserves finite extensions.

An event is cut-off iff there exists a configuration $C \in C_e$ s.t. $C \prec [e]$ and $C \approx [e]$.

In usual setting: $C \approx C' \iff \text{Mark}(C) \approx \text{Mark}(C')$
Talk overview

1. Introduction
2. Unfoldings
3. Verification with unfoldings
4. Other developments in the area
5. Beyond unfoldings & conclusion
6. References and bibliography
More topics in true concurrency I

- True concurrency semantics of process algebras
  Complete finite prefixes for a model similar to branching processes + adequate order on calculus formulae: [Langerak and Brinksma, 1999]. Summary of older work: [Boudol et al., 2008].

- Axiomatic concurrency theory
  Project started by Carl Petri himself.
  http://www.informatik.uni-hamburg.de/TGI/forschung/projekte/concurrency_eng.html

- Trace theory
  Mazurkiewicz traces – another formalism for true concurrency semantics.
  LTrL [Thiagarajan and Walukiewicz, 2002] is a logic for communicating multi-agent systems. LTrL is to Mazurkiewicz traces/event structures as LTL is for computational trees.
Theorem (Kamp’s theorem)

$LTL$ is equivalent to the first-order theory of (infinite) sequences

Theorem

$LTrL$ is equivalent to the first-order theory of traces
Topic in true concurrency: relations

Figure 21: Illustration from “A logic for true concurrency” by Silvia Crafa
Any questions?

Thank you for listening!
Talk overview

1. Introduction
2. Unfoldings
3. Verification with unfoldings
4. Other developments in the area
5. Beyond unfoldings & conclusion
6. References and bibliography


*Communication and concurrency.*
PHI Series in computer science. Prentice Hall.

Petri nets, event structures and domains, part I.

An expressively complete linear time temporal logic for Mazurkiewicz traces.

Models for concurrency.
*DAIMI Report Series, 22*(463).