True Concurrency and Net Unfoldings

Daniil Frumin

December 9, 2013

Talk overview

Talk overview

True concurrency?

Q: What is true concurrency?

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*?

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*?
A: It is semantics that respect true concurrency.

True concurrency semantics (CCS)

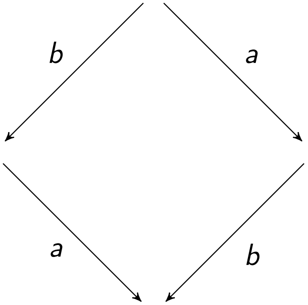

Figure 1: $a.b + b.a$

Interleaving world:

$$a \parallel b \approx a.b + b.a$$

Non-interleaving world:

$$a \parallel b \not\approx a.b + b.a$$

True concurrency semantics (CCS)

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

Usual process calculi semantics

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q}$$

$$\frac{Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P \parallel Q'}$$

Non-interleaving semantics

Additional rule breaks strong bisimulation:

$$\frac{P \to P' \qquad Q \to Q'}{P \parallel Q \to P' \parallel Q'}$$

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
http://arstechnica.com/security/2010/05/multicore-cpus-move-attack-from-theoretical-to-practical/

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

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.

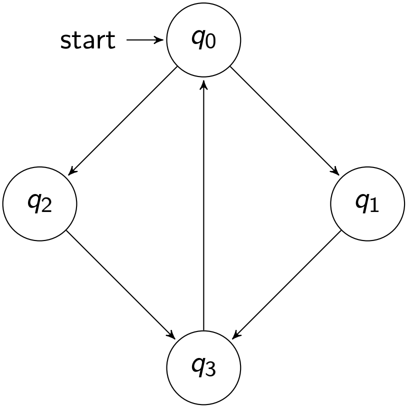

Figure 2: State machine SM_1

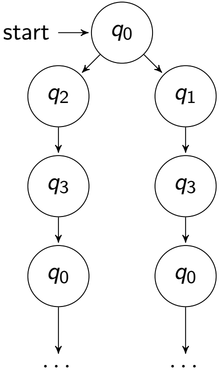

Figure 3: Unfoldings of the state machine SM_1

Unfolding a Petri net

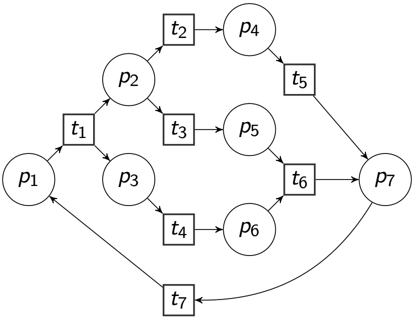

Figure 4: P/T net N_1

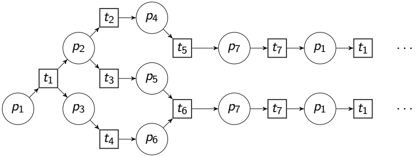

Figure 5: Unfoldings of the net N_1

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*:
  $x \# y \iff \exists t, t' \in E. t \neq t', pre(t) \cap pre(t') \neq \emptyset \wedge t \leq x \wedge t' \leq y$;
- *co* – the *concurrency* relation: $x \, co \, y \iff \neg(x < y) \wedge \neg(y < x) \wedge \neg(x \# y)$.

Relations on nodes: causality

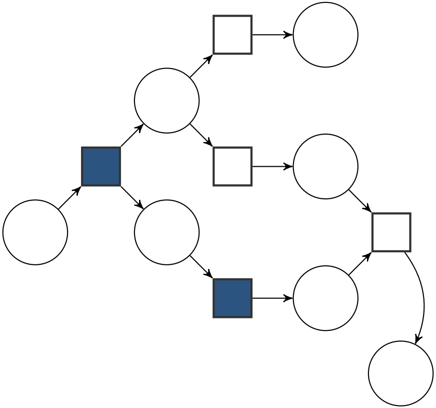

Figure 6: Causally dependent nodes

Relations on nodes: conflict

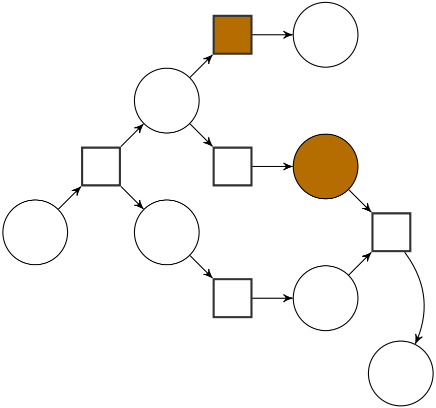

Figure 7: Nodes in conflict

Relations on nodes: concurrency

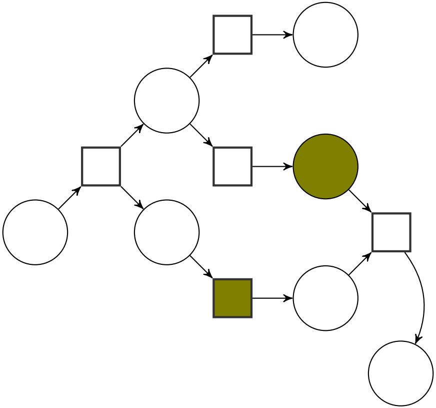

Figure 8: Concurrent nodes

Occurrence nets (definition)

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

- N is acyclic;
- $\forall p \in B, |pre(p)| \leq 1$;
- $\forall x \in N$ 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 $x \leq x' \wedge y \leq y'$ then $x' \# y'$.

[1] Some formalized proofs can be found at `http://me.hskll.org/repos/coq/OccurrNet.html`

Net morphisms

Let $N_1 = (B, E, pre_1, post_1), N_2 = (P, T, pre_2, post_2)$ be Petri nets.
$h : N_1 \to N_2$ is called a *net morphism* iff

1. $h(B) \subseteq P$, $h(E) \subseteq T$;
2. For each $e \in E$: $h(pre_1(e)) = pre_2(h(e))$ and $h(post_1(e)) = post_2(h(e))$.

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 morpishm; nets with morphisms form a category **Petri**).

Branching processes (definition)

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. $h : O \to N$ – 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[2].

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).

[2] Slightly modified version of what is presented in [E... ...2008]

Examples

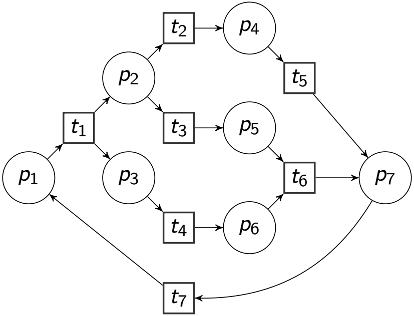

Figure 9: P/T net N_1

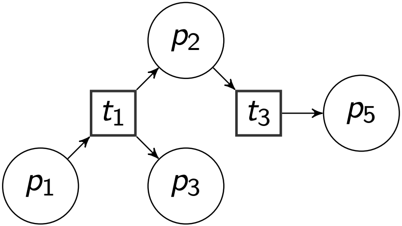

Figure 10: Branching process BP_1 for the net N_1

Examples

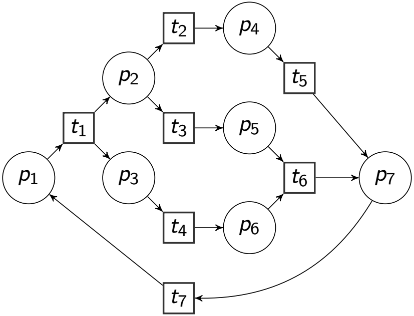

Figure 11: P/T net N_1

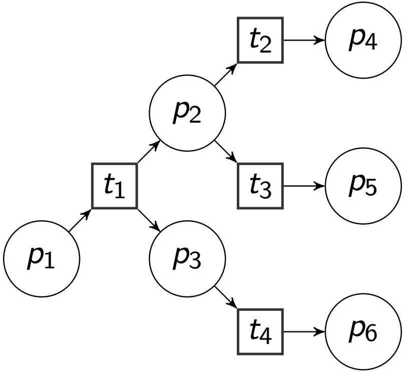

Figure 12: Branching process BP_2 for the net N_1

Examples

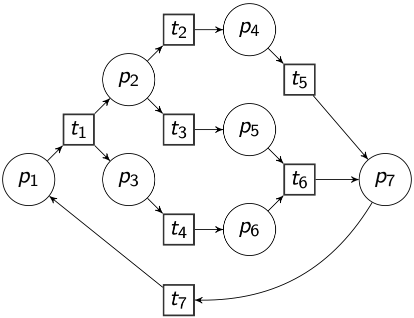

Figure 13: P/T net N_1

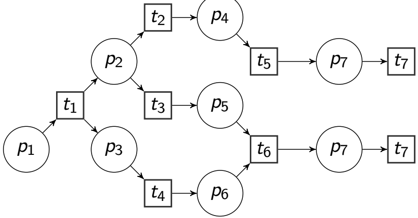

Figure 14: Branching process BP_3 for the net N_1

Net unfolding (definition)

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 isomoprhism).

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 $Bs = \{S_i \mid i \in Ind\}$ then is simply $\bigcup Bs$. See [Engelfriet, 1991] for more details. $\square$

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 posses the same behavioral properties that original net has.

Net unfoldings (fundamental property)

Proof sketch.

The theorem can be proved using induction on the length of the fireable sequence σ.

1. In case of $\sigma = \epsilon$ – obvious
2. In case of $\sigma = \sigma' t$ we have (by the induction hypothesis) $\mu_0[\sigma'\rangle\mu_1$, $M_0[\psi\rangle M_1$, $h(\psi) = \sigma' \wedge h(M_1) = \mu_1$. Since t is active $pre(t) \subseteq \mu_1 \implies pre(t) \subseteq h(M_1)$. Then $pre(t) = h(M_1')$ for some $M_1' \subseteq M_1$. Then $U(N)$ contains an event e s.t. $pre(e) = M_1'$ and $h(e) = t$. If it wasn't the case, than $U(N)$ wouldn't be the maximal branching process.

□

Talk overview

Finite prefixes: battling the state space explosion problem

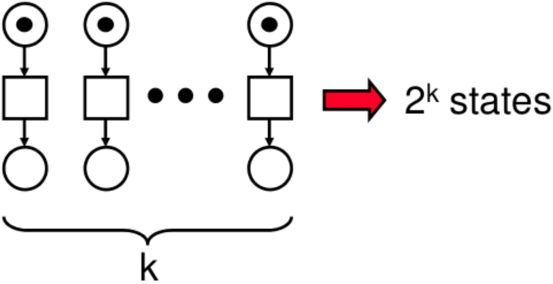

Figure 15: State space explosion, common in highly concurrent systems

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*
$Conf(e) = \{e' \mid e' \leq e\}$

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*
$Conf(e) = \{e' \mid e' \leq e\}$

Definition

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

- Cuts characterizes reachable markings;
- Each configuration induces a cut: $Cut(C) = (Min \cup post(C)) \setminus pre(C)$, where Min is the set of $<$-minimal nodes of a branching process (i.e. the initial marking, starting nodes, $h(M_0)$).

Marking-complete finite prefixes

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 $pre(e)$ is already in U and elements of $pre(e)$ are pairwise concurrent.
3. Grab an element t from the queue, prioritized by the size of the local configuration. Add t and $post(t)$ 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 $h(Cut(e')) = h(Cut(e))$ (i.e. they transition to the same markings) and $|Cut(e')| < |Cut(e)|$.

Checking for properties (deadlock)

- 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.

[5] N has a reachable marking M such that no transition can be fired

We can produce the formula ψ that corresponds to the configurations of a (complete) prefix BP.
Each satisfactory assignment of ψ determines a valid configuration in BP.

Checking for deadlock with SAT-solvers

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

$$\psi_e = \bigwedge_{f \in pre(pre(e))} (e \implies f) \wedge \bigwedge_{f \# e} (\neg e \vee \neg f) \wedge \bigwedge_{e \text{ is a cut-off event}} (\neg e)$$

Checking for deadlock with SAT-solvers

$$\psi_e = \bigwedge_{f \in pre(pre(e))} (e \implies f) \wedge \bigwedge_{f \# e} (\neg e \vee \neg f) \wedge \bigwedge_{e \text{ is a cut-off event}} (\neg e)$$

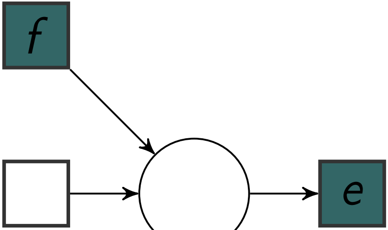

Figure 16: $e \implies f$

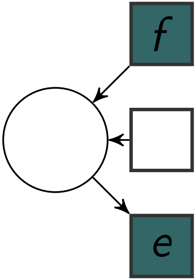

Figure 17: $\neg e \vee \neg f$

Checking for deadlock with SAT-solvers

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

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

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 $e' = pre(p)$):

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

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.

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

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

$$\psi \implies (enables(a) \vee \cdots \vee enables(z))$$

where $\{a, \ldots, z\}$ is the set of transitions of the net N.

Sidenote: complexity issues I

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].

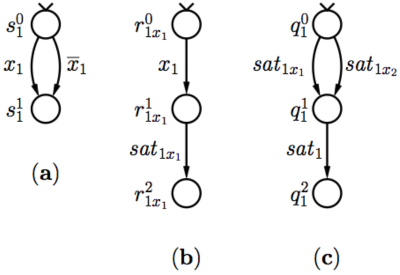

Figure 18: Synchronized product for (a) variable x_1 (b) literal x_1 in clause $x_1 \vee x_2$ (c) clause $x_1 \vee x_2$ in formula $(x_1 \vee x_2) \wedge \overline{x_1}$; taken from from [Esparza and Heljanko, 2008]

Sidenote: complexity issues II

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.

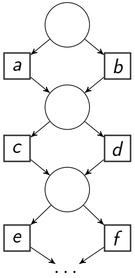

Figure 19: Net N_2

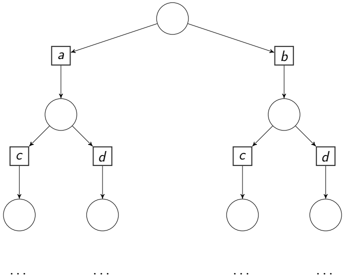

Figure 20: Finite prefix of N_2 according to the McMillan's algorithm

Generalization: room for improvement

It has been noted that McMillan's algorithm can generate prefixes bigger than needed.

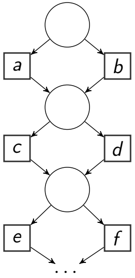

Figure 19: Net N_2

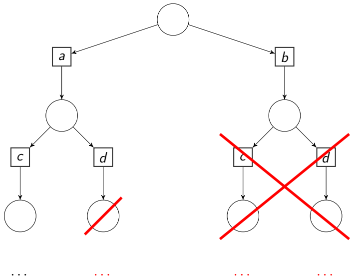

Figure 20: Finite prefix of N_2 according to the McMillan's algorithm

Generalization: adequate orders I

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*.

Generalization: adequate orders II

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: $C \subsetneq C' \implies C \prec C'$;
- $\prec$ is preserved by finite extensions: if $Mark(C) = Mark(C')$ 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 $pre(e)$ is already in U and elements of $pre(e)$ are pairwise concurrent.
3. Grab an element t from the queue, prioritized by the size of the local configuration. Add t and $post(t)$ 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 $pre(e)$ is already in U and elements of $pre(e)$ 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 $post(t)$ 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. $\exists t \in T . t < x$. 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 $<_{lex}$ be a lexicographical order on set of sequences of transitions; we can "lift" $<_{lex}$ to the set of configurations by declaring $C <_{lex} C'$ iff $\mathit{flat}(C) <_{lex} \mathit{flat}(C)$ where $\mathit{flat}(C)$ is a sequence of transitions ordered by $<_{lex}$ 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 $C <_{lex} C'$;
  - or if $|C| = |C'|$, $\mathit{flat}(C) = \mathit{flat}(C')$, and
    - $Min(C) <_{lex} Min(C')$;
    - or $\mathit{flat}(Min(C)) <_{lex} \mathit{flat}(Min(C'))$ and $C \setminus Min(C) \prec C \setminus Min(C')$

  $Min(C)$ – 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 $<_{lex}$ be a lexicographical order on set of sequences of transitions; we can "lift" $<_{lex}$ to the set of configurations by declaring $C <_{lex} C'$ iff $\mathit{flat}(C) <_{lex} \mathit{flat}(C)$ where $\mathit{flat}(C)$ is a sequence of transitions ordered by $<_{lex}$ 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 $C <_{lex} C'$;
  - or if $|C| = |C'|$, $\mathit{flat}(C) = \mathit{flat}(C')$, and
    - $Min(C) <_{lex} Min(C')$;
    - or $\mathit{flat}(Min(C)) <_{lex} \mathit{flat}(Min(C'))$ and $C \setminus Min(C) \prec C \setminus Min(C')$

  $Min(C)$ – 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

Infinite excecutability 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 excecutability 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 can not be solved using such prefix.
Infinite excecutability problem?

Cut-off criterion for 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 $Mark([e']) = Mark([e])$ and either

1. $e' < e$ or
2. $\#_r([e']) \geq \#_r([e])$.

Model checking using unfoldings

Arbitrary properties (expressed in LTL) can be checked using unfoldings: [Couvreur et al., 2000, Esparza and Heljanko, 2001].

More generalizations

Cutting contex [Khomenko, 2003] is a generalization that allows us to preserve only the properties we want when constructing a finite prefix.

$$\Theta = (\approx, \prec, \{\mathcal{C}_e\}_{e \in E})$$

1. $\prec$ – adequate order;
2. $\{\mathcal{C}_e\}_{e \in E}$ – 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 $C \in \mathcal{C}_e$ s.t. $C \prec [e]$ and $C \approx [e]$.
In usual setting: $C \approx C' \iff Mark(C) \approx Mark(C')$

Talk overview

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.

More topics in true concurrency II

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

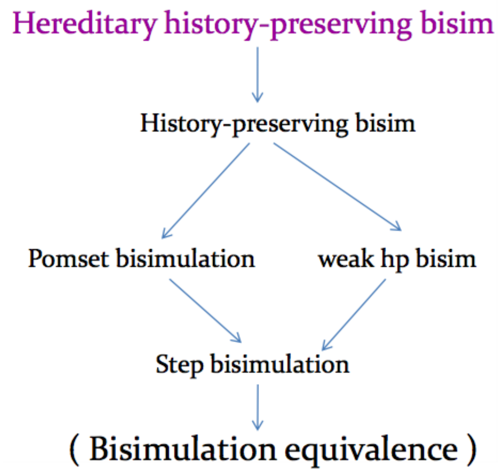

Figure 21: Illustration from "A logic for true concurrency" by Silvia Crafa

The end

Thank you for listening!

Any questions?

Talk overview

References I

Aceto, L., Larsen, K. G., and Ingolfsdottir, A. (2005).
An introduction to Milner's CCS.
http://www.cs.auc.dk/~luca/SV/intro2ccs.pdf.

Boudol, G., Castellani, I., Hennessy, M., Nielsen, M., and Winskel, G. (2008).
Twenty years on: Reflections on the CEDISYS project. combining true concurrency with process algebra.
In Degano, P., Nicola, R., and Meseguer, J., editors, *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 757–777. Springer Berlin Heidelberg.

Couvreur, J.-M., Grivet, S., and Poitrenaud, D. (2000).
Designing a LTL model-checker based on unfolding graphs.
In Nielsen, M. and Simpson, D., editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 123–145. Springer Berlin Heidelberg.

Engelfriet, J. (1991).
Branching processes of Petri nets.
Acta Inf., 28(6):575–591.

Esparza, J. (2010).
A false history of true concurrency: From Petri to tools.
In *Proceedings of the 17th International SPIN Conference on Model Checking Software*, SPIN'10, pages 180–186, Berlin, Heidelberg. Springer-Verlag.

Esparza, J. and Heljanko, K. (2001).
Implementing LTL model checking with net unfoldings.
In *Proceedings of the 8th International SPIN Workshop on Model Checking of Software*, SPIN '01, pages 37–56, New York, NY, USA. Springer-Verlag New York, Inc.

Esparza, J. and Heljanko, K. (2008).
Unfoldings: a partial-order approach to model checking.
Springer.

Esparza, J., Römer, S., and Vogler, W. (1996).
An improvement of McMillan's unfolding algorithm.
In Margaria, T. and Steffen, B., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106. Springer Berlin Heidelberg.

References IV

Heljanko, K. (1999).
Deadlock and reachability checking with finite complete prefixes.
Technical report, Helsinki University of Technology, Laboratory for Theoretical Computer Science.

Heljanko, K. (2000).
Model checking with finite complete prefixes is PSPACE-complete.
In *Proceedings of the 11th International Conference on Concurrency Theory*, CONCUR '00, pages 108–122, London, UK, UK. Springer-Verlag.

Khomenko, V. (2003).
Model Checking Based on Prefixes of Petri Net Unfoldings.
Ph.D. Thesis, School of Computing Science, Newcastle University.

Langerak, R. and Brinksma, E. (1999).
A complete finite prefix for process algebra.
In Halbwachs, N. and Peled, D., editors, *Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 184–195. Springer Berlin Heidelberg.

McMillan, K. L. (1993).
Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits.
In *Computer Aided Verification*, pages 164–177. Springer.

McMillan, K. L. (1995).
A technique of state space search based on unfolding.
Form. Methods Syst. Des., 6(1):45–65.

References VI

Milner, R. (1989).
Communication and concurrency.
PHI Series in computer science. Prentice Hall.

Nielsen, M., Plotkin, G., and Winskel, G. (1981).
Petri nets, event structures and domains, part I.
Theoretical Computer Science, 13(1):85–108.

Thiagarajan, P. S. and Walukiewicz, I. (2002).
An expressively complete linear time temporal logic for Mazurkiewiczr traces.
Information and Computation, 179(2):230–249.

Winskel, G. and Nielsen, M. (1993).
Models for concurrency.
DAIMI Report Series, 22(463).