ASR : an introduction to Distributed Systems and Algorithms (Algorithmes et Systèmes Répartis)

4/4

M. Raynal¹, E. Fabre²

¹ Prof., ASAP team
² DR INRIA, DistribCom team

Master 2, research in Computer Science, ’12
Download the lecture slides from my web-page:
http://people.rennes.inria.fr/Eric.Fabre/
Previously on ASR...

Applications of FCPs
- Reachability analysis
- Deadlock detection

Distributed computations with BPs
- Morphism
- Combination of nets
- Combination of occurrence nets
- A central theorem
- Diagnosis
- Projection of a branching process
- Distributed diagnosis

Conclusion
Summary of last time

Modeling a distributed application

- essential feature: the concurrency of events
- a simple formalism to model concurrent systems: (safe) Petri nets
- run of a PN in a true concurrency semantics = configuration
Sets of runs

- Occurrence net $\mathcal{O}$
  - partially ordered net (well founded)
  - conditions (places) have at most one cause (predecessor)
  - causality, conflict, concurrency

Branching process $(\mathcal{O}, \phi)$ of a net $\mathcal{N}$

- $\phi : \mathcal{O} \rightarrow \mathcal{N}$ is a labeling on events and transitions
- parsimony: $\forall e, e', \quad \bullet e = \bullet e', \quad \phi(e) = \phi(e') \implies e = e'$
- configurations of $\mathcal{O}$ represent by $\phi$ the runs of $\mathcal{N}$
- maximal BP of $\mathcal{N} =$ the unfolding of $\mathcal{N}$
Sets of runs

- Occurrence net $O$
  - partially ordered net (well founded)
  - conditions (places) have at most one cause (predecessor)
  - causality, conflict, concurrency

- Branching process $(O, \phi)$ of a net $N$
  - $\phi : O \to N$ is a labeling on events and transitions
  - parsimony: $\forall e, e', \cdot e = \cdot e', \phi(e) = \phi(e') \Rightarrow e = e'$
  - configurations of $O$ represent by $\phi$ the runs of $N$
  - maximal BP of $N$ = the unfolding of $N$
Finite and complete prefixes

- obtained by stopping the unfolding algorithms at cut-off events
- necessity of an adequate order, when comparing events in the unfolding algorithm, to define consistent cut-offs
- represent all reachable markings of the PN
- can be used for reachability analysis and for deadlock detection (today)
Outline

1. Previously on ASR...

2. Applications of FCPs
   - Reachability analysis
   - Deadlock detection

3. Distributed computations with BPs
   - Morphism
   - Combination of nets
   - Combination of occurrence nets
   - A central theorem
   - Diagnosis
   - Projection of a branching process
   - Distributed diagnosis

4. Conclusion
Reachability analysis

Reachability analysis with a finite complete prefix

Is the (sub-)marking $M$ reachable in net $\mathcal{N}$?
By completeness, it suffices to check this on a FCP of $\mathcal{N}$.

**Sub-marking:**
Try to reach $(Q_1, Q_0)$, i.e. a marking where places in $Q_1$ are marked, and places in $Q_0$ are empty.
Remark 1: By introducing complementary places $p^c$ to $N$, $\forall p \in P$, becomes equivalent to assuming $Q_0 = \emptyset$.

- exactly one of $\{p, p^c\}$ is marked at each time
- $t \in \cdot p \iff t \in p^c\cdot$ and $t \in p\cdot \iff t \in \cdot p^c$
- excepted for $t \in \cdot p \cap p^c$

Exercise: express a safe PN with complementary places as a product of elementary components, each one being an automaton (single token)
Remark 2:

- Looking for the submarking \((Q_1, \emptyset)\) amounts to checking that some extra transition \(t^f\) with \(\bullet t^f = Q_1\) is firable in \(\mathcal{N}\).
- This is equivalent to looking for a co-set on which \(t^f\) can be fired in the unfolding algorithm.
- So reachability and co-set construction have identical complexities.

Proposition

The reachability test (co-set construction) is NP-hard.
Remark 2:

- Looking for the submarking \((Q_1, \emptyset)\) amounts to checking that some extra transition \(t^f\) with \(\bullet t^f = Q_1\) is firable in \(\mathcal{N}\).
- This is equivalent to looking for a co-set on which \(t^f\) can be fired in the unfolding algorithm.
- So reachability and co-set construction have identical complexities.

**Proposition**

*The reachability test (co-set construction) is NP-hard.*
Proof: it is equivalent to solving a SAT problem.

Example:

encoding of the SAT problem \((x_1 \lor x_2 \lor \bar{x}_3) \land (\bar{x}_1 \lor x_2)\)

in a safe Petri net

The complexity of moving to the unfolding is polynomial. So finding a co-set where \(t_f\) is firable is NP-hard.
Algorithm: to build a co-set $X$ with $f_N(X) = Q_1 = \{p_1, p_2, \ldots\}$

- **Method**
  - take a condition $c_1$ with $f_N(c_1) = p_1$
  - extend it with a condition $c_2$ with $f_N(c_2) = p_2$, and $c_2 \perp c_1$
  - try to add $c_3$ with $f_N(c_3) = p_3$, and $c_3 \perp \{c_1, c_2\}$
  - etc., backtracking when failing

- **Concurrency test for $c_1 \perp c_2$**
  - build $[c_1]$ and $[c_2]$
  - check $[c_1] \cap [c_2] \cap C = \emptyset$ i.e. no conflict
  - check $c_1 \not\in [c_2]$ , i.e. not $c_1 \rightarrow^* c_2$
  - check $c_2 \not\in [c_1]$ , i.e. not $c_2 \rightarrow^* c_1$

- The best implementations use SAT solvers for these properties.

- These ideas also used to build unfoldings/prefixes. Modern implementations use and update a list of possible extensions (= event + co-set).
Algorithm: to build a co-set $X$ with $f_{\mathcal{N}}(X) = Q_1 = \{p_1, p_2, \ldots\}$

- **Method**
  - take a condition $c_1$ with $f_{\mathcal{N}}(c_1) = p_1$
  - extend it with a condition $c_2$ with $f_{\mathcal{N}}(c_2) = p_2$, and $c_2 \perp c_1$
  - try to add $c_3$ with $f_{\mathcal{N}}(c_3) = p_3$, and $c_3 \perp \{c_1, c_2\}$
  - etc., backtracking when failing

- **Concurrency test for** $c_1 \perp c_2$
  - build $[c_1]$ and $[c_2]$
  - check $[c_1] \cap [c_2] \cap C = \emptyset$ i.e. no conflict
  - check $c_1 \notin [c_2]$, i.e. not $c_1 \rightarrow^* c_2$
  - check $c_2 \notin [c_1]$, i.e. not $c_2 \rightarrow^* c_1$

- The best implementations use SAT solvers for these properties.

- These ideas also used to build unfoldings/prefixes. Modern implementations use and update a list of possible extensions ($= \text{event} + \text{co-set}$).

Another approach: using the marking equation

- let \( \omega = t^1, ..., t^n \) be a firable sequence in net \( N \)
- the Parikh vector \( \vec{\omega} = [\nu(\omega, t)]_{t \in T} \) counts occurrences of each transition \( t \) of \( T \) in \( \omega \)
- the incidence matrix \( N \in \mathbb{N}^{\|P\| \times \|T\|} \) is defined by
  \[
  N(p, t) = \mathbf{1}_{t \rightarrow p} - \mathbf{1}_{p \rightarrow t}
  \]
- the marking \( M \) reached by \( \omega \) is given by
  \[
  M = P_0 + N \cdot \vec{\omega}
  \]

Proposition

\( M \) is reachable only if there exists a solution to \( M = P_0 + N \cdot X \), \( X \in \mathbb{N}^{\|T\|} \). If the net \( N \) is acyclic, this condition is also sufficient.

Can be used on a finite complete prefix, assuming it is already available. Resolution by integer linear programming.
Another approach: using the marking equation

- let $\omega = t^1, \ldots, t^n$ be a firable sequence in net $N$
- the Parikh vector $\vec{\omega} = [\nu(\omega, t)]_{t \in T}$ counts occurrences of each transition $t$ of $T$ in $\omega$
- the incidence matrix $N \in \mathbb{N}^{\mid P \mid \times \mid T \mid}$ is defined by
  \[
  N(p, t) = 1_{t \rightarrow p} - 1_{p \rightarrow t}
  \]
- the marking $M$ reached by $\omega$ is given by
  \[
  M = P_0 + N \cdot \vec{\omega}
  \]

**Proposition**

$M$ is reachable only if there exists a solution to $M = P_0 + N \cdot X$, $X \in \mathbb{N}^{\mid T \mid}$. If the net $N$ is acyclic, this condition is also sufficient.

Can be used on a finite complete prefix, assuming it is already available. Resolution by integer linear programming.
Deadlock checking

**Deadlock** = marking of $\mathcal{N}$ where no more transition can be fired.

**Proposition (Mc Millan)**

Let $O$ be a FCP of $\mathcal{U}_\mathcal{N}$. There exists no deadlock in $\mathcal{N}$ iff every configuration $\kappa$ of $O$ can be extended into a configuration $\kappa' \sqsupseteq \kappa$ that contains a cut-off event.

This is simply because there is no deadlock iff every configuration can be made arbitrarily large in $\mathcal{U}_\mathcal{N}$ (recall the notion of $e$-shift).

**Corollary**

Equivalently, there is a deadlock in $\mathcal{N}$ iff there exists some $\kappa$ that can’t be extended to reach a cut-off.
Deadlock checking

**Deadlock** = marking of $\mathcal{N}$ where no more transition can be fired.

**Proposition (Mc Millan)**

Let $O$ be a FCP of $\mathcal{U}_\mathcal{N}$. There exists no deadlock in $\mathcal{N}$ iff every configuration $\kappa$ of $O$ can be extended into a configuration $\kappa' \sqsupseteq \kappa$ that contains a cut-off event.

This is simply because there is no deadlock iff every configuration can be made arbitrarily large in $\mathcal{U}_\mathcal{N}$ (recall the notion of e-shift).

**Corollary**

Equivalently, there is a deadlock in $\mathcal{N}$ iff there exists some $\kappa$ that can’t be extended to reach a cut-off.
Corollary

Equivalently, there is a deadlock in $N$ iff there exists some $\kappa$ that can’t be extended to reach a cut-off.

Proof of $\Leftarrow$

- extend $\kappa$ into a maximal config. $\kappa'$, that contains no cut-off
- no more transition can be fired after $\kappa'$ (otherwise one of its maximal events would be a cut-off)
- so $Mark(\kappa')$ is a deadlock

Proof of $\Rightarrow$

- let $M$ be a deadlock and $\kappa \in O$ such that $Mark(\kappa) = M$
- there exists one such $\kappa$ that contains no cut-off (remove cut-offs one by one, as in the completeness proof of a FCP)
- this $\kappa$ can’t be extended at all (otherwise $M$ wouldn’t be a deadlock)
Corollary

Equivalently, there is a deadlock in $N$ iff there exists some $\kappa$ that can’t be extended to reach a cut-off.

Proof of $\iff$

- extend $\kappa$ into a maximal config. $\kappa'$, that contains no cut-off
- no more transition can be fired after $\kappa'$ (otherwise one of its maximal events would be a cut-off)
- so $Mark(\kappa')$ is a deadlock

Proof of $\Rightarrow$

- let $M$ be a deadlock and $\kappa \in O$ such that $Mark(\kappa) = M$
- there exists one such $\kappa$ that contains no cut-off (remove cut-offs one by one, as in the completeness proof of a FCP)
- this $\kappa$ can’t be extended at all (otherwise $M$ wouldn’t be a deadlock)
Corollary

Let $\kappa$ be a configuration in conflict with all cut-off events of $O$. Let $\kappa' \supseteq \kappa$ be a maximal extension of $\kappa$ in $O$. Then $\text{Mark}(\kappa')$ is a deadlock.

Homework: does this characterize all deadlocks?

- checking the existence of deadlocks is also NP hard
- there exist “graphical” methods on the FCP $O$
- there exist as well methods based on the marking equation
Distributed computations with branching processes

1. Previously on ASR...

2. Applications of FCPs
   - Reachability analysis
   - Deadlock detection

3. Distributed computations with BPs
   - Morphism
   - Combination of nets
   - Combination of occurrence nets
   - A central theorem
   - Diagnosis
   - Projection of a branching process
   - Distributed diagnosis

4. Conclusion
Distributed computations with branching processes

1. Previously on ASR...

2. Applications of FCPs
   - Reachability analysis
   - Deadlock detection

3. Distributed computations with BPs
   - Morphism
   - Combination of nets
   - Combination of occurrence nets
   - A central theorem
   - Diagnosis
   - Projection of a branching process
   - Distributed diagnosis

4. Conclusion
Morphism of (labeled) PN:

\[ \phi : \mathcal{N}_1 \rightarrow \mathcal{N}_2 \quad \text{with} \quad \mathcal{N}_i = (P_i, T_i, \rightarrow_i, P_{i,0}, \lambda_i, \Lambda_i) \]

- \( \phi : T_1 \rightarrow T_2 \) partial function on transitions,
  - label preserving: \( t_2 = \phi(t_1) \Rightarrow \lambda_2(t_2) = \lambda_1(t_1) \)
  - \( \Lambda_2 \subseteq \Lambda_1 \) and \( \text{Dom}(\phi) = \lambda_1^{-1}(\Lambda_2) \)
- \( \phi \) relation between place sets \( P_1 \) and \( P_2 \)
  - \( \phi(P_{1,0}) = P_{2,0} \)
  - \( \forall p_2 \in P_{2,0}, \exists! p_1 \in P_{1,0}, \quad p_1 \overset{\phi}{\rightarrow} p_2 \)
  - \( \phi \) defined at \( p_1 \) \( \Rightarrow \) \( \phi \) defined at \( \cdot p_1 \) and \( p_1 \).
- flow preservation: \( t_2 = \phi(t_1) \) implies
  - \( \phi^{\text{op}} : \cdot t_2 \rightarrow \cdot t_1 \) is a total function
  - \( \phi^{\text{op}} : t_2 \cdot \rightarrow t_1 \cdot \) is a total function
Example 1:

Example 2: the foldings \( f_N : U_N \rightarrow N \)
(recall: a folding is a total function)

Lemma
Net morphisms preserve runs, i.e. a firable sequence of \( N_1 \) is mapped by \( \phi \) into a firable sequence of \( N_2 \).
Example 1:

Example 2: the foldings $f_N : \mathcal{U}_N \rightarrow N$
(recall: a folding is a total function)

Lemma

Net morphisms preserve runs, i.e. a firable sequence of $N_1$ is mapped by $\phi$ into a firable sequence of $N_2$. 
Previously on ASR...

Applications of FCPs
- Reachability analysis
- Deadlock detection

Distributed computations with BPs
- Morphism
- Combination of nets
- Combination of occurrence nets
- A central theorem
- Diagnosis
- Projection of a branching process
- Distributed diagnosis

Conclusion
Combination of nets

**Product of PN** (recall): \( N = N_1 \times N_2 = (P, T, \rightarrow, P_0, \lambda, \Lambda) \)

- **Places:** \( P = P_1 \cup P_2 \) *disjoint union*
- **Transitions:**

\[
T = \{(t_1, t_2) : \lambda_1(t_1) = \lambda_2(t_2)\} \quad \text{synchro. by shared labels}
\]
\[
\cup \{(t_1, \star) : \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} \quad \text{private to } N_1
\]
\[
\cup \{(*, t_2) : \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} \quad \text{private to } N_2
\]

- **Flow:** \( \rightarrow \) defined by

\[
(t_1, t_2)^* = t_1^* \cup t_2^*
\]
\[
(t_1, t_2)^* = t_1^* \cup t_2^*
\]
Where morphisms appear...

- The canonical mappings $\psi_i : N_1 \times N_2 \rightarrow N_i$ are morphisms.
- Products of nets satisfy a universal property (in the category theory sense).

\[\begin{array}{c}
\text{a} \\
\text{t}_2 \beta \\
\text{b} \\
\end{array} \\
\xrightarrow{\psi_1} \\
\begin{array}{c}
\text{a} \\
\text{t}_1 \alpha \\
\end{array}
\]

\[\begin{array}{c}
\text{a} \\
\text{t}_2 \beta \\
\text{b} \\
\end{array} \\
\xrightarrow{\psi_2} \\
\begin{array}{c}
\text{a} \\
\text{t}_2 \beta \\
\text{b} \\
\end{array} \\
\xrightarrow{(t_1, t_3)} \\
\begin{array}{c}
\text{c} \\
\text{t}_4 \gamma \\
\end{array} \\
\xrightarrow{\psi_2} \\
\begin{array}{c}
\text{c} \\
\text{t}_3 \alpha \\
\text{d} \\
\end{array} \\
\xrightarrow{t_4 \gamma} \\
\begin{array}{c}
\text{c} \\
\text{t}_4 \gamma \\
\end{array}
\]

Composition by “pullback” (also called fibered product)

- Corresponds to the case where \( \mathcal{N}_1, \mathcal{N}_2 \) have a common sub-net \( \mathcal{N}_0 \), by morphisms \( \phi_i : \mathcal{N}_i \rightarrow \mathcal{N}_0 \)
- Specific case of interest:
  \( \mathcal{N}_0 \) is an interface between \( \mathcal{N}_1 \) and \( \mathcal{N}_2 \) iff \( \Lambda_1 \cap \Lambda_2 \subseteq \Lambda_0 \)
- This means that all interactions between \( \mathcal{N}_1, \mathcal{N}_2 \) are captured by \( \mathcal{N}_0 \).

![Diagram of nets and transitions]

\( t_3 \rightarrow t_2 \rightarrow t_1 \rightarrow t_4 \) in \( \mathcal{N}_1 \)

\( \phi_1 \) and \( \phi_2 \) map \( \mathcal{N}_1 \) and \( \mathcal{N}_2 \) to \( \mathcal{N}_0 \) respectively.
Construction: \( \mathcal{N} = \mathcal{N}_1 \wedge \mathcal{N}_2 \) or simply \( \mathcal{N} = \mathcal{N}_1 \wedge \mathcal{N}_2 \) [assumes the morphisms \( \phi_i \) are functions on places]

- **Places:**
  \[
P = \{(p_1, p_2) : \phi_1(p_1) = \phi_2(p_2)\} \quad \text{shared places}
  \cup \{(p_1, \star) : p_1 \notin \text{Dom}(\phi_1)\} \quad \text{places private to} \ \mathcal{N}_1
  \cup \{(*, p_2) : p_2 \notin \text{Dom}(\phi_2)\} \quad \text{places private to} \ \mathcal{N}_2
  \]

- **Transitions:**
  \[
  T = \{(t_1, t_2) : \phi_1(t_1) = \phi_2(t_2)\} \quad \text{shared transitions}
  \cup \{(t_1, t_2) : \lambda_1(t_1) = \lambda_2(t_2), \ t_i \notin \text{Dom}(\phi_i)\} \quad \text{sync. outs.} \ \mathcal{N}_0
  \cup \{(t_1, \star) : \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} \quad \text{private to} \ \mathcal{N}_1
  \cup \{(*, t_2) : \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} \quad \text{private to} \ \mathcal{N}_2
  \]

- **Flow, etc.:** same as before
Example:

- corresponds to a product when $N_0$ is empty
- the canonical $\psi_i : N_1 \land N_2 \to N_i$ are morphisms
- the diagram is commutative: $\phi_1 \circ \psi_1 = \phi_2 \circ \psi_2$
- a universal property is attached to this construction as well
Previously on ASR...

Applications of FCPs
- Reachability analysis
- Deadlock detection

Distributed computations with BPs
- Morphism
- Combination of nets
- Combination of occurrence nets
- A central theorem
- Diagnosis
- Projection of a branching process
- Distributed diagnosis

Conclusion
Combination of occurrence nets

Product of occurrence nets:

- same as the product of nets...
  ...but must yield an occurrence net!
- $O_1 \times O_2$ is generally not an occurrence net
  (Exercise: build a counter-example)
- Definition:

  \[
  O_1 \times^O O_2 \triangleq \mathcal{U}(O_1 \times O_2)
  \]

- This formula allows to recycle the unfolding algorithm to compute products on ONs.
Algorithmic construction of $O = O_1 \times^O O_2$

- **Init**
  - $C = C_{1,0} \uplus C_{2,0}$, and $\psi_i : C \rightarrow C_i$ canonical projections
  - $E = \emptyset$, $\rightarrow = \emptyset$, ...

- **Repeat until stability**
  - connect a **shared event**
    - select co-set $X \subseteq C$ and events $e_1, e_2$ in $E_1, E_2$
    - s.t. $\lambda_1(e_1) = \lambda_2(e_2)$ and $\cdot e_i = \psi_i(e_i)$
    - add $e = (e_1, e_2)$ to $E$, with $\cdot e = X$
    - create conditions $X' = e^*$ in $C$, such that $\psi_i(X') = e_i^*$
  - connect a **private event** of $O_1$
    - select co-set $X \subseteq C$ and events $e_1 \in E_1$
    - s.t. $\lambda_1(e_1) \in \Lambda_1 \setminus \Lambda_2$ and $\cdot e_1 = \psi_1(e_1)$
    - add $e = (e_1, \star)$ to $E$, with $\cdot e = X$
    - create conditions $X' = e^*$ in $C$, such that $\psi_1(X') = e_1^*$
  - connect a **private event** of $O_2$ : symmetrical
Algorithmic construction of $O = O_1 \times^O O_2$

- **Init**
  - $C = C_{1,0} \uplus C_{2,0}$, and $\psi_i : C \rightarrow C_i$ canonical projections
  - $E = \emptyset$, $\rightarrow = \emptyset$, ...

- **Repeat until stability**
  - connect a shared event
    - select co-set $X \subseteq C$ and events $e_1, e_2$ in $E_1, E_2$
      s.t. $\lambda_1(e_1) = \lambda_2(e_2)$ and $\cdot e_i = \psi_i(e_i)$
    - add $e = (e_1, e_2)$ to $E$, with $\cdot e = X$
    - create conditions $X' = e^\ast$ in $C$, such that $\psi_i(X') = e_i^\ast$

  - connect a private event of $O_1$
    - select co-set $X \subseteq C$ and events $e_1 \in E_1$
      s.t. $\lambda_1(e_1) \in \Lambda_1 \setminus \Lambda_2$ and $\cdot e_1 = \psi_1(e_1)$
    - add $e = (e_1, \star)$ to $E$, with $\cdot e = X$
    - create conditions $X' = e^\ast$ in $C$, such that $\psi_1(X') = e_1^\ast$

  - connect a private event of $O_2$: symmetrical
Algorithmic construction of $O = O_1 \times^O O_2$

- **Init**
  - $C = C_{1,0} \uplus C_{2,0}$, and $\psi_i : C \to C_i$ canonical projections
  - $E = \emptyset$, $\rightarrow = \emptyset$, ...

- **Repeat until stability**
  - connect a **shared event**
    - select co-set $X \subseteq C$ and events $e_1, e_2$ in $E_1, E_2$
    - s.t. $\lambda_1(e_1) = \lambda_2(e_2)$ and $e_i = \psi_i(e_i)$
    - add $e = (e_1, e_2)$ to $E$, with $e = X$
    - create conditions $X' = e^*$ in $C$, such that $\psi_i(X') = e_i^*$

  - connect a **private event** of $O_1$
    - select co-set $X \subseteq C$ and events $e_1 \in E_1$
    - s.t. $\lambda_1(e_1) \in \Lambda_1 \setminus \Lambda_2$ and $e_1 = \psi_1(e_1)$
    - add $e = (e_1, \star)$ to $E$, with $e = X$
    - create conditions $X' = e^*$ in $C$, such that $\psi_1(X') = e_1^*$

  - connect a **private event** of $O_2$ : symmetrical
Example:
Example:
Example:
Combination of occurrence nets

Example:
Example:

Combination of occurrence nets
Combination of occurrence nets

Example:
Example:
Example:
Example:
Example:

Combination of occurrence nets
Example:
Combination of occurrence nets

Example:
Example:
Combination of occurrence nets

**Pullback of BP:**

- $O_1 \land^O O_2$ defined as $\mathcal{U}(O_1 \land O_2)$
- computable by a similar algorithm

**Example:**
| 1 | Previously on ASR... |
| 2 | Applications of FCPs |
|   | Reachability analysis |
|   | Deadlock detection |
| 3 | Distributed computations with BPs |
|   | Morphism |
|   | Combination of nets |
|   | Combination of occurrence nets |
|   | A central theorem |
|   | Diagnosis |
|   | Projection of a branching process |
|   | Distributed diagnosis |
| 4 | Conclusion |
Theorem (product is preserved by unfolding)

Let $\mathcal{N} = \times_{i \in I} \mathcal{N}_i$, then $\mathcal{U}_\mathcal{N} = \times_{i \in I}^{\mathcal{O}} \mathcal{U}_{\mathcal{N}_i}$.

- as for languages, where $L(\times_i \mathcal{A}_i) = \times_i^L L(\mathcal{A}_i)$ ...
- in the same way, combinations by pullback are preserved by unfolding
Example:
Distributed computations with branching processes

1. Previously on ASR...

2. Applications of FCPs
   - Reachability analysis
   - Deadlock detection

3. Distributed computations with BPs
   - Morphism
   - Combination of nets
   - Combination of occurrence nets
   - A central theorem
   - Diagnosis
     - Projection of a branching process
     - Distributed diagnosis

4. Conclusion
Centralized diagnosis

- **observation**: let $\Lambda_o \subseteq \Lambda$ be the set of observable labels
- a hidden run $\kappa_h$ of $\mathcal{N}$ is produced
- one observes only $\lambda(\kappa_h) \cap \Lambda_o$
- these labels are collected either as a sequence or as a partial order $\text{Obs}$ (i.e. an occurrence net)
Definition

\textbf{Diagnosis} = \textit{the set of runs of } \mathcal{N} \textit{ that explain the observations.}

\textit{Obtained from } \mathcal{D} = \mathcal{U}_{\mathcal{N}} \times^{O} \text{Obs} = \mathcal{U}(\mathcal{N} \times \text{Obs}).

- the product synchronizes possible runs of \mathcal{N} with observations \text{Obs}
- the hidden run \kappa_h is present in \mathcal{D}
- Caution: not all configurations of \mathcal{D} explain entirely observations \text{Obs}
- they may only explain a prefix of it
Distributed computations with branching processes

1. Previously on ASR...

2. Applications of FCPs
   - Reachability analysis
   - Deadlock detection

3. Distributed computations with BPs
   - Morphism
   - Combination of nets
   - Combination of occurrence nets
   - A central theorem
   - Diagnosis
   - Projection of a branching process
   - Distributed diagnosis

4. Conclusion
Projection of a branching process

Projection of BP:

- let $O \subseteq \mathcal{U}_N$ and $N = \times_{i \in I} N_i$
- from $\psi_i : \mathcal{U}_N \rightarrow \mathcal{U}_{N_i}$ we can define the projection of $O$ on $N_i$

$$\Pi_i O \triangleq \psi_i(O) \subseteq \mathcal{U}_{N_i}$$

Example:
Projection of a branching process

Projection of BP:

- let $O \subseteq \mathcal{U}_N$ and $N = \times_{i \in I} N_i$
- from $\psi_i : \mathcal{U}_N \rightarrow \mathcal{U}_{N_i}$ we can define the projection of $O$ on $N_i$
  \[
  \Pi_i O \triangleq \psi_i(O) \subseteq \mathcal{U}_{N_i}
  \]

Example:
Projection of BP:

Let $O \subseteq U_N$ and $N = \times_{i \in I} N_i$

From $\psi_i : U_N \rightarrow U_{N_i}$ we can define the projection of $O$ on $N_i$

$$\Pi_i O \triangleq \psi_i(O) \subseteq U_{N_i}$$

Example:
Projection of a branching process

- projection = restriction (to desired nodes) + trimming
- projections may lose some conflict or causality relations and make them look as fake concurrency
Projection of a branching process

Minimal product covering

Proposition

Assume $N = \times_{i \in I} N_i$, then

$$U_N = \times_{i \in I} O_{U_{N_i}} = \times_{i \in I} \Pi_i(U_N)$$

where $\Pi_i(U_N) \subseteq U_{N_i}$ is the minimal factor of $U_N$ in component $N_i$.

- configurations $\kappa_i$ of $\Pi_i(U_N)$ represent runs of $N_i$ that remain possible in $N$.
- minimality: taking a strict prefix $O_i \subseteq \Pi_i(U_N)$ prevents reconstructing the whole $U_N$.
- the same result holds with pullbacks instead of products.

Can we compute the $\Pi_i(U_N)$ without computing first $U_N$?
Projection of a branching process

Minimal product covering

**Proposition**

Assume $\mathcal{N} = \times_{i \in I} \mathcal{N}_i$, then

$$\mathcal{U}_\mathcal{N} = \times_{i \in I} \mathcal{U}_{\mathcal{N}_i} = \times_{i \in I} \Pi_i(\mathcal{U}_\mathcal{N})$$

where $\Pi_i(\mathcal{U}_\mathcal{N}) \subseteq \mathcal{U}_{\mathcal{N}_i}$ is the **minimal factor** of $\mathcal{U}_\mathcal{N}$ in component $\mathcal{N}_i$.

- configurations $\kappa_i$ of $\Pi_i(\mathcal{U}_\mathcal{N})$ represent runs of $\mathcal{N}_i$ that remain possible in $\mathcal{N}$
- minimality: taking a strict prefix $O_i \supseteq \Pi_i(\mathcal{U}_\mathcal{N})$ prevents reconstructing the whole $\mathcal{U}_\mathcal{N}$
- the same result holds with pullbacks instead of products

Can we compute the $\Pi_i(\mathcal{U}_\mathcal{N})$ without computing first $\mathcal{U}_\mathcal{N}$?
Previously on ASR...

Applications of FCPs
- Reachability analysis
- Deadlock detection

Distributed computations with BPs
- Morphism
- Combination of nets
- Combination of occurrence nets
- A central theorem
- Diagnosis
- Projection of a branching process
- Distributed diagnosis

Conclusion
Key to message passing algorithms

Theorem

Let $N = N_1 \land N_2$, with $N_0$ an interface automaton between $N_1$ and $N_2$. Let $O = O_1 \land^O O_2$ be a BP of $N$. Then

$$\Pi_1(O) = O_1 \land^O \Pi_0(O_2)$$
Meaning of these equations

\[ \Pi_1(O) = O_1 \land^O \Pi_0(O_2) \]

- \( \Pi_0(O_2) \) is a message sent from \( N_2 \) to \( N_1 \) about its behaviors on the interface net \( N_0 \) (and symmetrically)
- the computations involve small and local BP, \( O \) never appears
- this message-passing principle extends to networks of components
Distributed diagnosis

**Assumptions**

- \( \mathcal{N} = \mathcal{N}_1 \land \mathcal{N}_2 \), interface net \( \mathcal{N}_0 \)
- visible labels \( \Lambda_{o,i} \subseteq \Lambda_i \) in each \( \mathcal{N}_i \)
- a hidden run \( \kappa_h \) is performed by \( \mathcal{N} \)
- \( Obs_i = \lambda_i(\kappa_h) \cap \Lambda_{o,i} \)

represents the observations collected on component \( \mathcal{N}_i \)
Problem

- compute the projection on each $N_i$ of the global explanations
  \[ D = U_N \times^O Obs \]  where \[ Obs = Obs_1 \times^O Obs_2 \]
- recover runs of each component that contribute to explaining all observations (local views of the global diagnosis)

Formally

\[
D = (U_{N_1} \times^O Obs_1) \land^O (U_{N_2} \times^O Obs_2) \\
D_1 = \Pi_1(D) \\
D_1 = (U_{N_1} \times^O Obs_1) \land^O \Pi_0(U_{N_2} \times^O Obs_2) \\
D = D_1 \land^O D_2
\]
Problem

- compute the projection on each $N_i$ of the global explanations
  \[ D = \mathcal{U}_N \times^O \text{Obs} \quad \text{where} \quad \text{Obs} = \text{Obs}_1 \times^O \text{Obs}_2 \]
- recover runs of each component that contribute to explaining all observations (local views of the global diagnosis)

Formally

\[
\begin{align*}
D &= (\mathcal{U}_{N_1} \times^O \text{Obs}_1) \wedge^O (\mathcal{U}_{N_2} \times^O \text{Obs}_2) \\
D_1 &= \Pi_1(D) \\
&= (\mathcal{U}_{N_1} \times^O \text{Obs}_1) \wedge^O \Pi_0(\mathcal{U}_{N_2} \times^O \text{Obs}_2) \\
D &= D_1 \wedge^O D_2
\end{align*}
\]
Example:
Example:
Example:
Example:
Example:
Example:
Example:
Outline

1. Previously on ASR...
2. Applications of FCPs
   - Reachability analysis
   - Deadlock detection
3. Distributed computations with BPs
   - Morphism
   - Combination of nets
   - Combination of occurrence nets
   - A central theorem
   - Diagnosis
   - Projection of a branching process
   - Distributed diagnosis
4. Conclusion
Conclusion & Perspectives

The essential:
- concurrency = essential feature of distributed systems
- runs as partial orders $\Rightarrow$ less runs!
- best representation of runs for weakly coupled components
- distributed computations for verification purposes $\Rightarrow$ less work!

Some open issues:
1. What about quantitative aspects (time, cost, probabilities) ?
2. Approximate distributed algorithms to deal with (very) large systems.
3. Distributed systems with dynamic structures...
Conclusion & Perspectives

The essential:
- concurrency = essential feature of distributed systems
- runs as partial orders ⇒ less runs!
- best representation of runs for weakly coupled components
- distributed computations for verification purposes ⇒ less work!

Some open issues:
1. What about quantitative aspects (time, cost, probabilities)?
2. Approximate distributed algorithms to deal with (very) large systems.
3. Distributed systems with dynamic structures...