Partial order reductions using compositional confluence detection

Frédéric Lang

Radu Mateescu

INRIA Rhône-Alpes / VASY

http://www.inrialpes.fr/vasy

Context (1/2)

- **Explicit state verification of concurrent systems**
  - Parallel composition of asynchronous processes
  - Synchronisation or interleaving of communication actions
  - Systematic exploration of the behaviour graph
- **Several techniques to palliate state explosion**
  - *Compositional verification* : apply property preserving reductions to the graphs of the composed processes
  - *Partial order reductions* : avoid interleavings that are useless with respect to the properties under verification
  - *On-the-fly verification* : only explore states when necessary to evaluate the property under verification

Context (2/2)

- Those techniques can be combined
  - **CADP** toolbox (**http://www.inrialpes.fr/vasy/cadp**)
  - **Open/Caesar** environment
  - **Exp.Open** tool
- This talk presents two variants of a *new partial order reduction technique*, one preserving deadlocks and one preserving branching equivalence, based on a *compositional analysis* of the composed processes

Partial order reductions

persistent sets family [Godefroid, Valmari, Peled]

- Roots in communicating automata theory
- Operations are *dependent* if there can be some state in which they do not commute
- Find a subset S of the *operations* enabled in the current state such that every operation $\notin$ S and *dependent* on an operation $\in$ S cannot be enabled before an operation $\in$ S is fired
- *Deadlocks* are preserved if operations $\notin$ S are postponed
- *Visible traces* or *branching equivalence* can be preserved under additional conditions

Partial order reductions

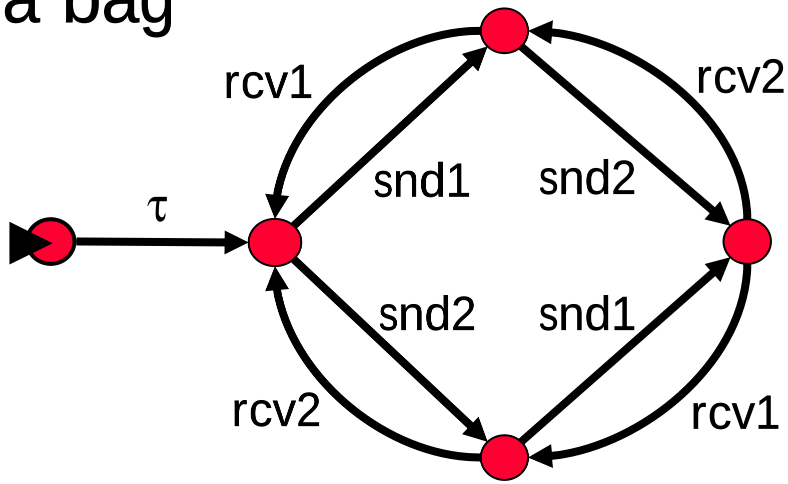
τ-confluence family [Groote, van de Pol, Ying]

- Roots in process algebra theory
- Find invisible (τ) transitions commuting with all other transitions
- *Branching equivalence* is preserved if transitions in choice with τ-confluent transitions are postponed
- Symbolic and/or (on-the-fly) explicit state detection tools exist

This talk combines persistent sets and τ-confluence

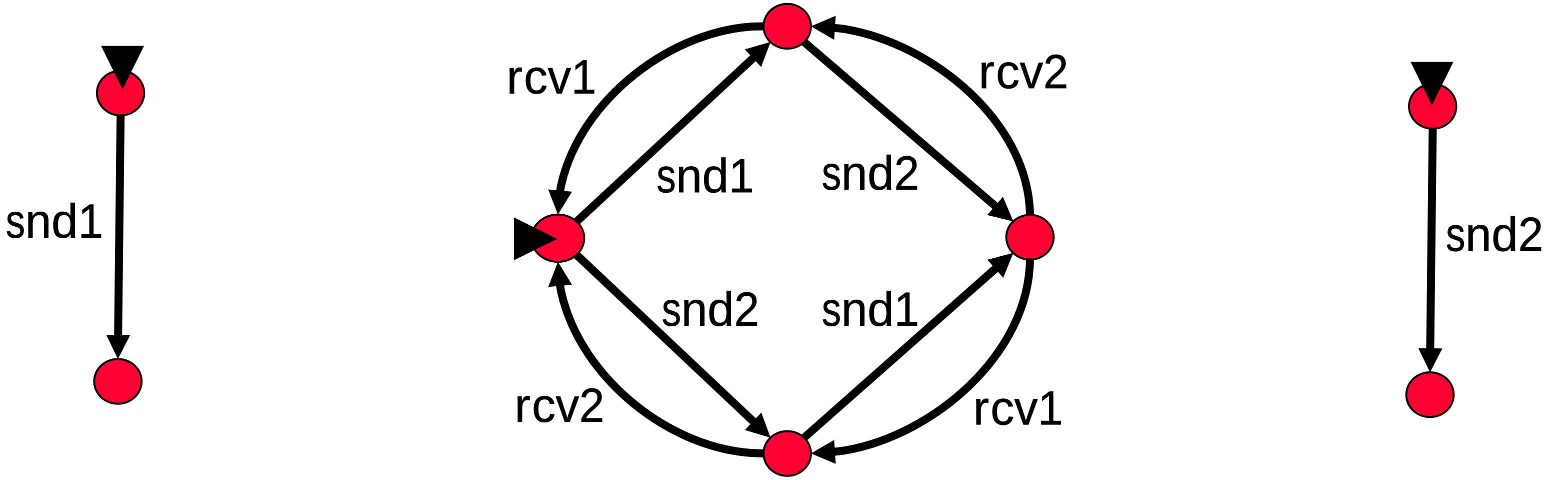
The network model (1/3)

- The model we use to represent concurrent systems
- **Each process is described by a graph**
- Each transition is labeled by a *visible communication action* or an *invisible action* τ
- **Example**: a bag



The network model (2/3)

- Graphs are composed using *synchronization rules*
- **Example**: Network N

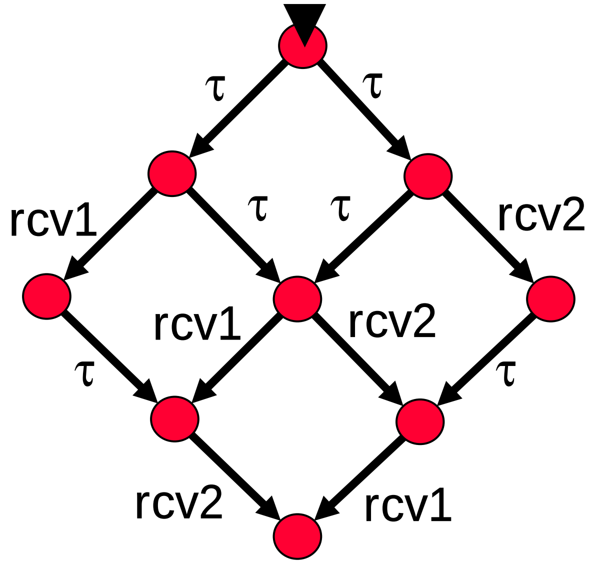


Rules: $(\bullet,\ \text{rcv1},\ \bullet) \rightarrow \text{rcv1}$ $(\text{snd1},\ \text{snd1},\ \bullet) \rightarrow \tau$

$(\bullet,\ \text{rcv2},\ \bullet) \rightarrow \text{rcv2}$ $(\bullet,\ \text{snd2},\ \text{snd2}) \rightarrow \tau$

The network model (3/3)

- Network semantics = product of composed graphs
- **Example**: semantics of N (previous slide)



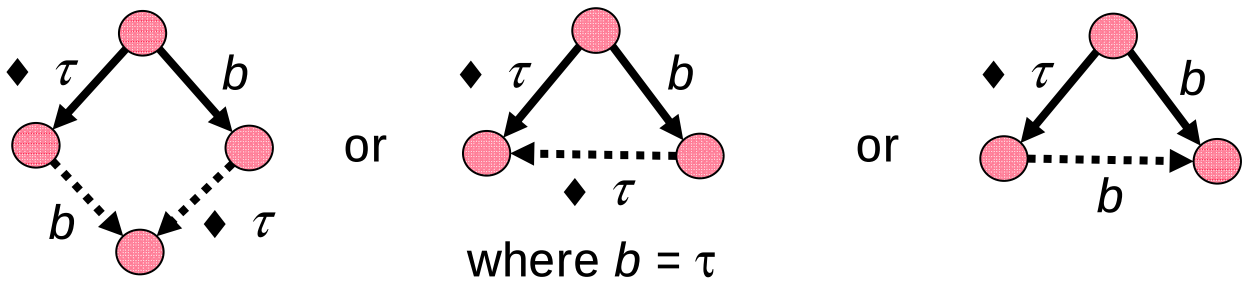
- Reasonable restrictions on τ actions guarantee that branching equivalence is a congruence for networks (no synchronisation, no cut, and no renaming of τ actions)

Persistent sets for networks

- Two operations are *dependent* if there is some state in which they may not commute
  - For networks, *operation* = *synchronization rule*
  - Two rules $(a_1, \ldots, a_n) \rightarrow a$ and $(b_1, \ldots, b_n) \rightarrow b$ are *dependent* if $(\exists i \in 1..n)\ a_i \neq \bullet \wedge b_i \neq \bullet$
  - Indeed, *in this case and only in this case*, there can be a state where one rule disables the other
- Persistent set construction for networks is described in [Lang-05]

τ-confluence

- Definition of *partial strong τ-confluence* by Groote & van de Pol (τ-confluence for short in this talk)
- A transition is *τ-confluent* ($\xrightarrow{\blacklozenge\,\tau}$) if:



- τ-confluent transitions can be *prioritized* as long as they do not close a circuit
- This preserves branching equivalence

τ-confluence for networks

- τ-confluence can be eliminated in composed graphs
  - **Correct** because τ-confluence elimination preserves branching equivalence
  - But **useless** if graphs are minimized for branching
- τ-confluence can be eliminated on-the-fly while computing the product graph
  - Efficient tools exist (**EXP.OPEN**/**REDUCTOR** tools of **CADP**)
  - But **cost increases non-linearly** with the size of the product graph

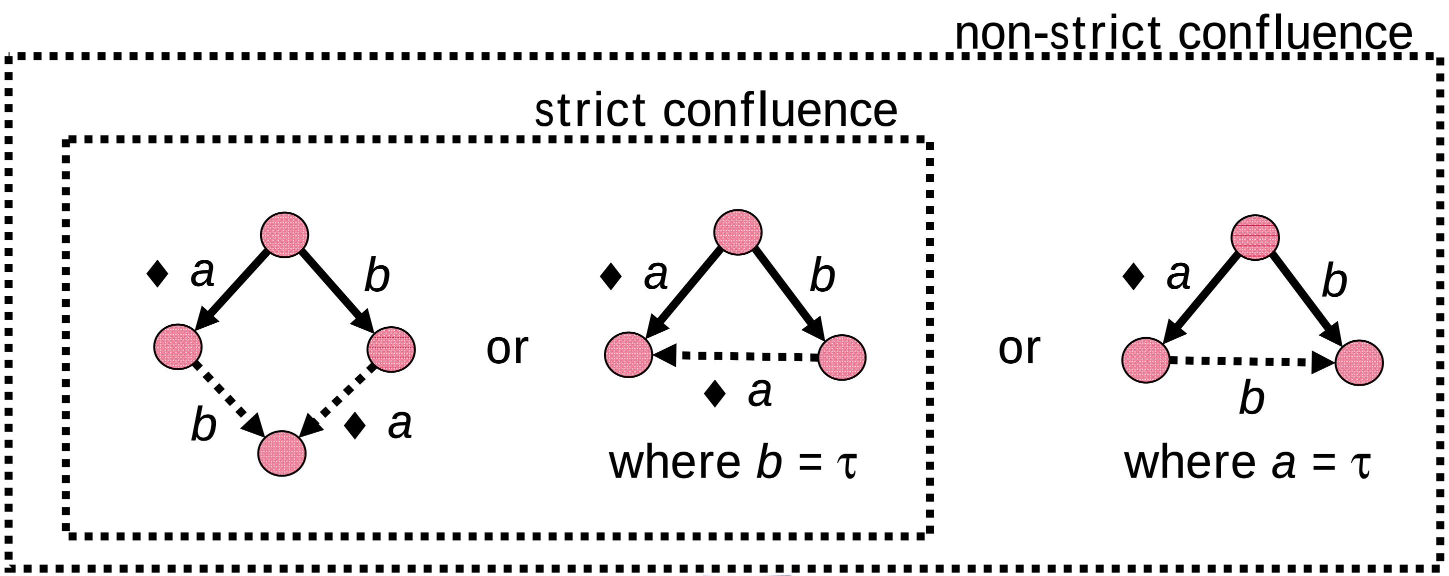
Compositional confluence detection

We present *Compositional Confluence Detection* (CCD)

- CCD removes some τ-confluent transitions that:
  - **Are obtained by synchronisation**, then hiding, of locally visible actions and thus cannot be removed beforehand in the composed graphs
  - **Are not detected by persistent set methods**
- CCD is **less resource consuming** than on-the-fly τ-confluence elimination in the product graph
- CCD **can be combined** with compositional verification and persistent set methods

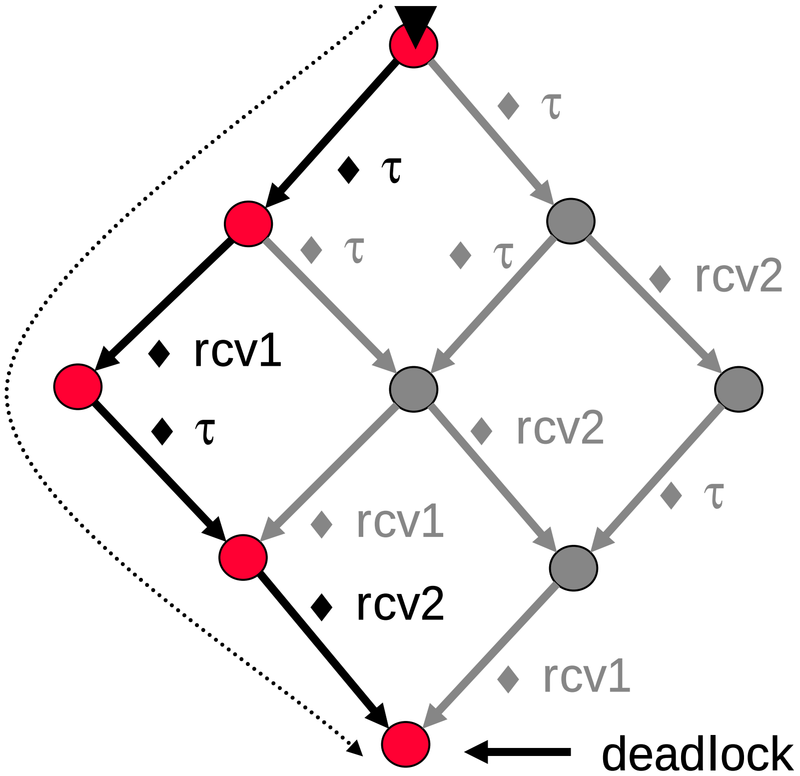
Confluence

- CCD requires a more general notion of *confluence*
  - Generalizes τ-confluence for visible actions
  - Is analogous to "confluent processes" (Milner) and lifted to transitions as Groote & van de Pol's τ-confluence
  - Has a strict and a non-strict variants
- A transition is [*strictly*] *confluent* ($\xrightarrow{\blacklozenge\, a}$) if:



Strict confluence theorem

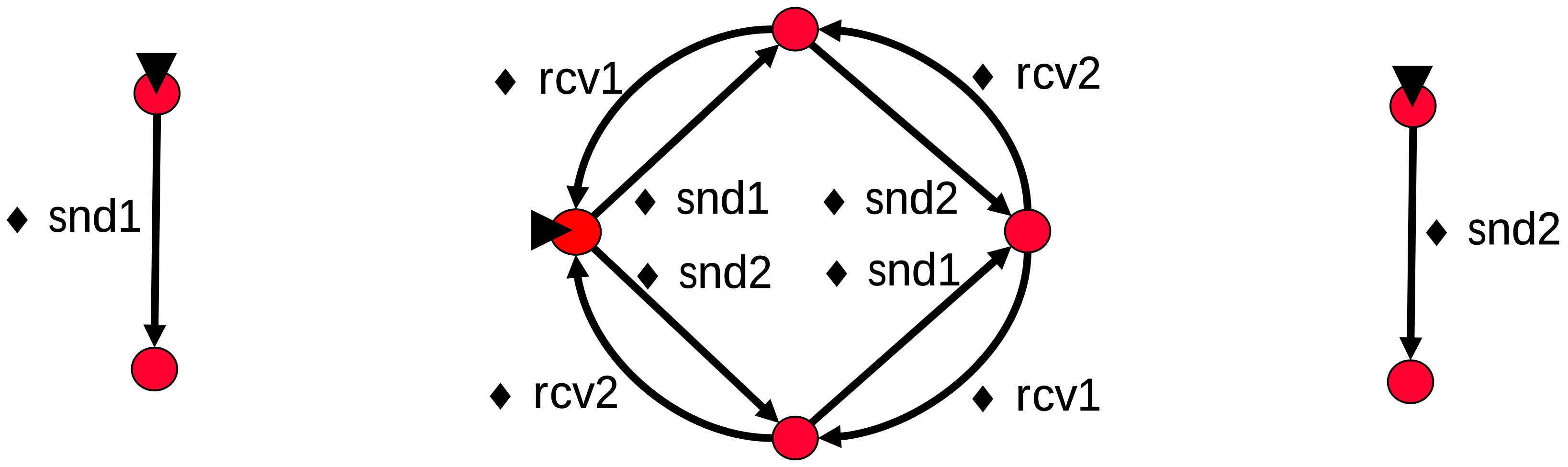
- **Theorem**: Prioritization of strictly confluent transitions preserves deadlocks
- Formal proof available in INRIA RR-7078
- **Example**:



Compositional confluence theorem

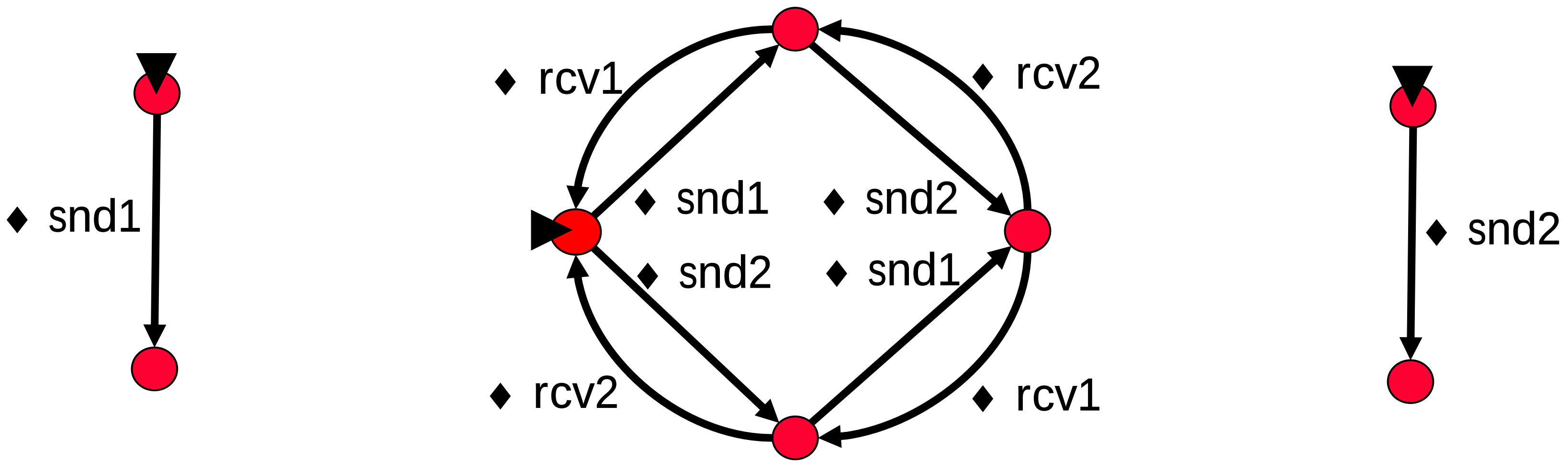
- **Theorem**: Transitions obtained by synchronisation of [strictly] confluent transitions are [strictly] confluent
- Formal proof available in INRIA RR-7078
- **Corollaries**:
  - Prioritizing transitions obtained by synchronization of **strictly confluent** transitions preserves **deadlocks**
  - Prioritizing τ-transitions obtained by synchronization of **confluent** transitions preserves **branching equivalence**, as long as they do not close a circuit

Example (1/2)



(snd1, snd1, •) $\rightarrow \tau$ yields a τ-confluent transition in init state as **both snd1-transitions are confluent**

Example (2/2)



- S = {(snd1, snd1, •) → τ} is not persistent in init state
  - S persistent if each operation ∉ S dependent on a operation ∈ S cannot be enabled before an operation ∈ S is fired
  - ((•, snd2, snd2), τ) ∉ S dependent on ((snd1, snd1, •), τ) ∈ S
  - Both rules are enabled in init state
- Same for S = { (•, snd2, snd2) → τ }

Confluence detection

- Encode the problem as the resolution of a maximal fixed point Boolean Equation System (BES):

$$\{ X_{s1,a,s2} =_\nu \bigwedge s_1 \rightarrow_b s_3 \Big($$

$$\bigvee s_2 \rightarrow_a s_4\ X_{s3,a,s4} \vee (b{=}\tau \wedge \bigvee s_3 \rightarrow_a s_2 \text{ true})$$

strict confluence

$$\vee$$

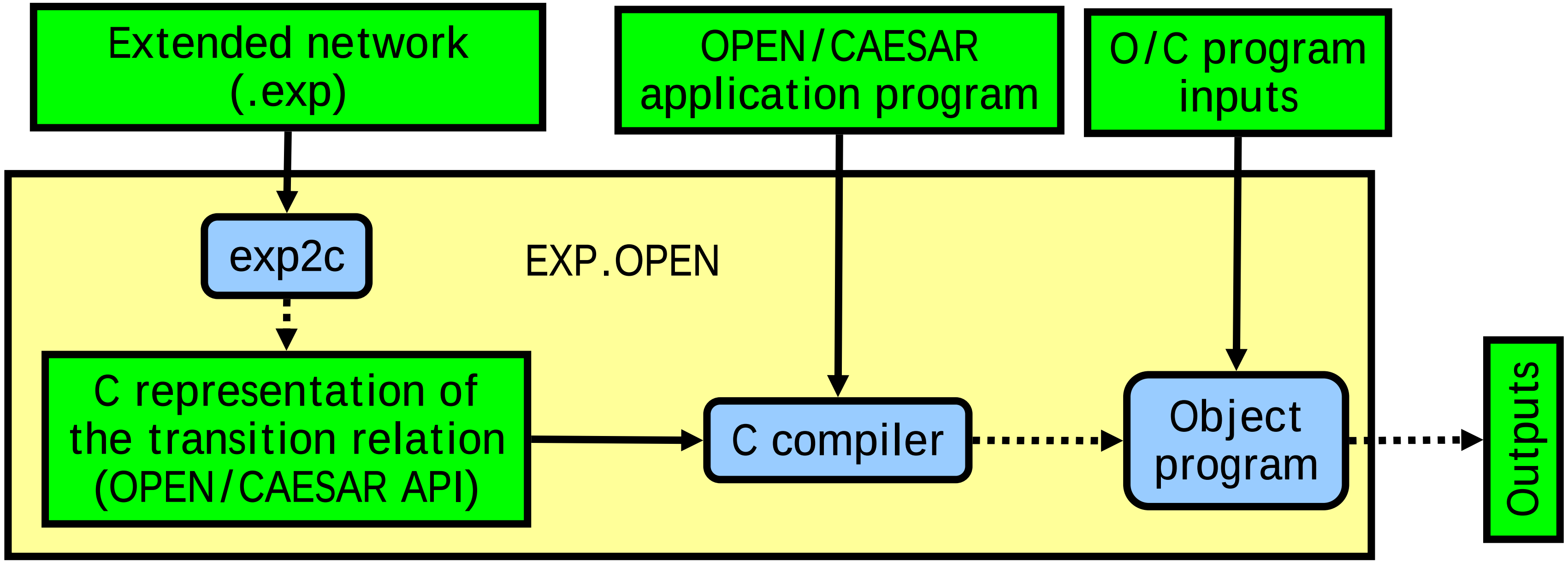
$$(a{=}\tau \wedge s_3{=}s_4)$$

$$\Big) \}$$

non-strict confluence

- $X_{s1,a,s2}$ true iff $s_1 \rightarrow_a s_2$ confluent
- BES resolution carried out using a global linear-time algorithm [Andersen-94, Mateescu-00]

The EXP.OPEN 2.0 tool of CADP



- New option **-confluence**
  - Combined with persistent set methods (**-deadpreserving**, **-weaktrace**, or **-branching** options)
  - Search [strictly] confluent transitions in composed graphs
  - Use confluence information to prioritize transitions

Experimental results
branching (1/2)

- CADP demos available at
  http://www.inrialpes.fr/vasy/cadp/demos
- ODP (Open Distributed Processing) trader (demo 37)
  - **22 K st. / 158 K trans.** using compositional verification
  - no reduction using persistent sets
  - **0,5 K st. / 2,8 K trans.** using CCD
- Asynchronous circuit for Data Encryption (demo 38)
  - **1,4 K st. / 3,5 K trans.** using compositional verification
  - no reduction using persistent sets
  - **0,3 K st. / 0,6 K trans.** using CCD

Experimental results
branching (2/2)

- Examples provided by **ST Microelectronics** (critical part of a multiprocessor system on chip)
- ST example 1:
  - **5,4 M st. / 37,6 M trans.** using compositional verification
  - no reduction using persistent sets
  - **5,1 M st. / 24,7 M trans.** using persistent sets + CCD
- ST example 2:
  - **789 M st. / 8104 M trans.** using compositional verification
  - no reduction using persistent sets
  - **710 M st. / 6143 M trans.** using persistent sets + CCD

Experimental results
deadlocks

- ODP trader
  - **22 K st. / 158 K trans.** using compositional verification
  - no reduction using persistent sets
  - **0,08 K st. / 0,1 K trans.** using persistent sets + CCD
- ST example 1:
  - **5,4 M st. / 37,6 M trans.** using compositional verification
  - **5,2 M st. / 34,2 M trans.** using persistent sets
  - **0,39 M st. / 1,3 M trans.** using persistent sets + CCD

Conclusion

- CCD (Compositional Confluence Detection) is **a new partial order reduction method**
  - It **works compositionally** by searching confluence in the composed graphs to detect confluence in the product
  - It can **improve the reductions obtained using persistent set methods**
- **CADP** (http://www.inrialpes.fr/vasy/cadp) supports CCD combined with **persistent sets**, **on-the fly verification** and **compositional verification**
- In the future, CCD could also be combined with **distributed graph generation**