Refined interfaces for compositional verification

Frédéric Lang

INRIA Rhône-Alpes / VASY

655, avenue de l'Europe

F-38330 Montbonnot Saint Martin

France

Context

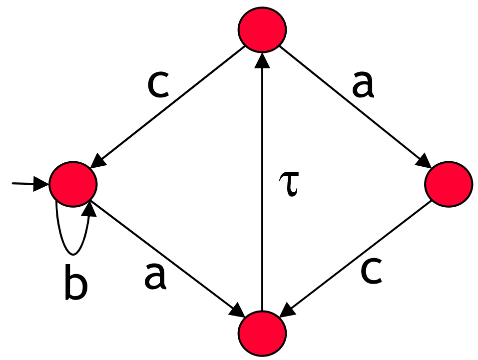
- Verification of concurrent systems
  - Processes running asynchronously in parallel
  - Formal descriptions (e.g. LOTOS)
  - Action-based models (state/transition graphs)
- Enumerative ("explicit state") verification methods
  - Systematic exploration of the state/transition graph
  - Example: model checking, equivalence checking, ...
- State explosion problem
  - Exponential growth of the state/transition graph
  - Several methods can be used to palliate state explosion
- Tool support: the CADP toolbox

Compositional Verification

- Compositional verification: "*divide and conquer*"
  - Partition the system into subsystems
  - Minimize each subsystem modulo a strong or weak bisimulation preserving the properties to verify
  - Recombine subsystems to get a system equivalent to the initial one
- Compositional verification may fail
  - Concurrent processes constrain each others
  - Separating tightly-coupled processes $\rightarrow$ explosion
- Solution: use interfaces
  - [Graf-Steffen-91], [Cheung-Kramer-93], [Krimm-Mounier-97]
  - Use interfaces to model the environment
- This talk: automated interface generation

State/transition graphs

- Semantic model of processes, also called Labelled Transition System (LTS)
- Transitions between states are labelled by events
  - Synchronizable/observable events
  - Non-synchronizable/hidden event τ

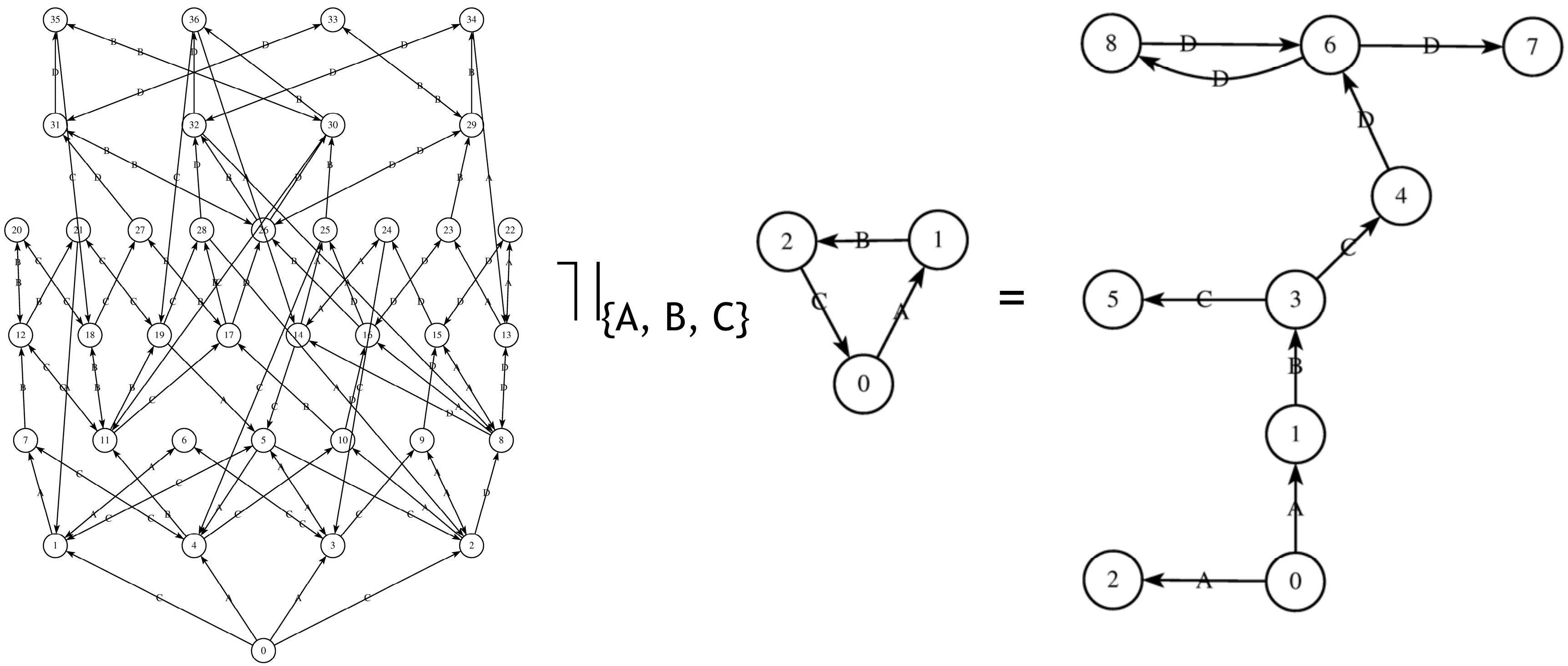


- CADP toolbox allows on-the-fly exploration of state/transition graphs (OPEN/CAESAR)

Using interface constraints

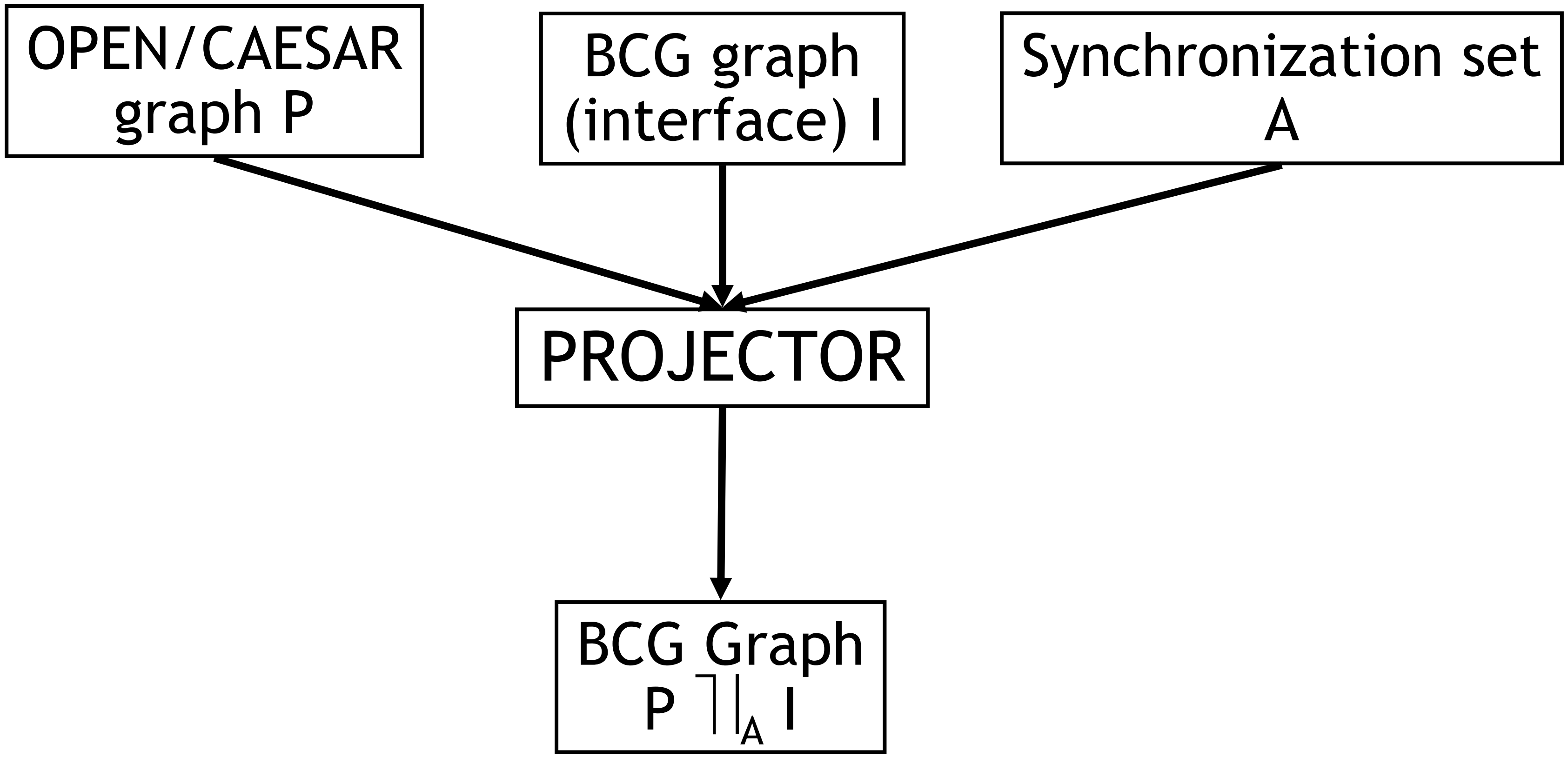
- A big graph P can be reduced using *interface constraints*, represented as a graph I and a set of labels A through which P and I interact
- Projection operator $P \rceil||_A I$ (Graf & Steffen, Krimm & Mounier)
  - Computes the sub-graph of P reachable in $P \; ||_A \; I$
  - I can be reduced modulo safety equivalence after hiding all labels outside A
- A similar approach exists for CSP (Cheung & Kramer)
  - Normal parallel composition instead of projection
  - Requires tau elimination and determinization (expensive) in I to ensure *context transparency*

Example of projection



The PROJECTOR tool of CADP

Software implementation of projection (Krimm & Mounier 1997)



Computing the interface constraints

- **Solution 1:** User-specified interface
  - The user provides an interface
  - A correct interface is hard to guess
  - But correctness can be checked afterwards
- **Solution 2:** Synthesized interface
  - A correct interface is computed automatically from the environment
  - Krimm & Mounier give an algorithm based on the analysis of a LOTOS expression describing the system as a parallel composition
    - The interface I is a process of the composition
    - The synchronization set A is derived automatically

Limitation 1 of K&M algorithm

The method to compute the synchronization set A is specific to LOTOS parallel composition

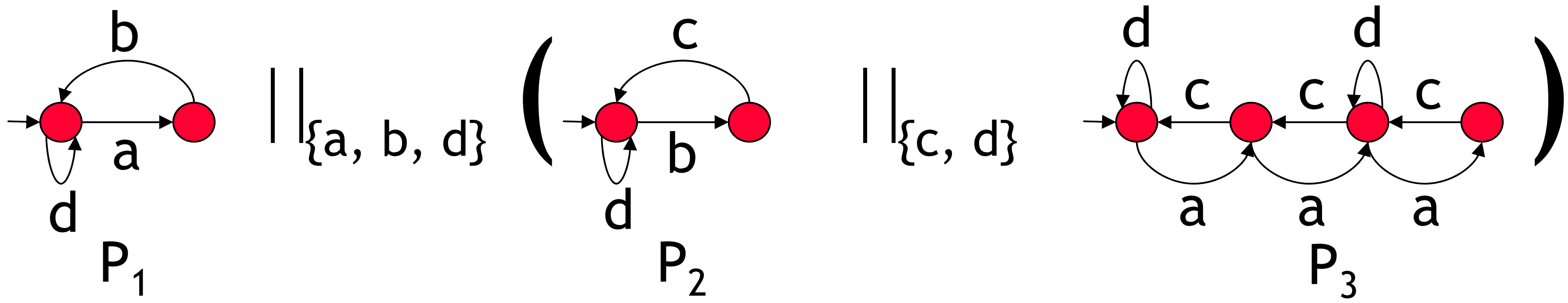
How can we synthesize interfaces in expressions that use different and/or more general operators?

Limitation 2 of K&M algorithm

It is impossible to compute interface constraints induced by **several** (not necessarily close) processes

Sometimes, only such constraints allow reductions

Example of limitation 2



- Restricting P_3 w.r.t. P_1 or P_2 yields no reduction:

$$P_3 \rceil\|_{\{a,\ d\}} P_1 = P_3 \rceil\|_{\{c,\ d\}} P_2 = P_3$$

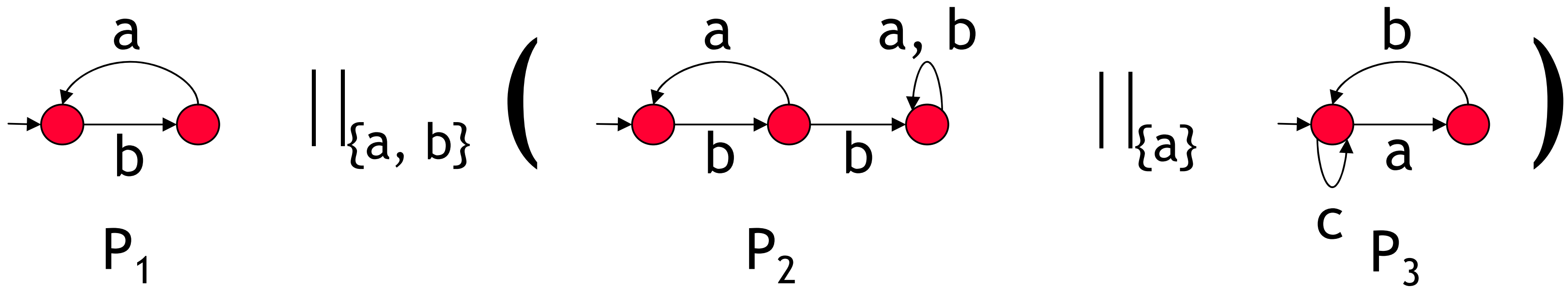
- Restricting P_3 w.r.t. both P_1 and P_2 (synchronized on b) would yield better reductions

Limitation 3 of K&M algorithm

Interfaces may be not precise enough when nondeterministic synchronization is involved

(i.e., a given transition may nondeterministically choose to synchronize or not with another)

Example of limitation 3



- When P_1, P_2, and P_3 are ready for a b transition, P_1 must choose to synchronize with either P_2 or P_3
- Restricting P_2 w.r.t. P_1 yields no reduction:

$$P_2 \;\rceil|_{\{a\}}\; P_1 = P_2$$

- However P_1 implies that two successive b actions cannot be reached without an a in between

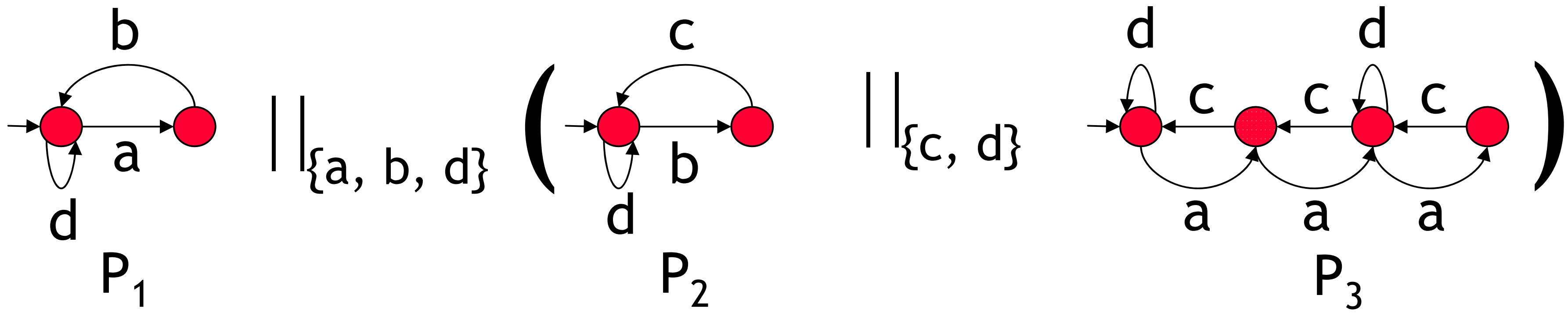
Refined interfaces

- We propose a new algorithm which solves the limitations of K&M algorithm
- Our algorithm works in three phases
  1. Translation of the composition of processes into a general model called "synchronization networks"
  2. Extraction of an "interface network" from the network model
  3. Generation of the interface graph corresponding to the interface network

Phase 1: synchronization networks

- A general synchronization model for an arbitrary number of processes $P_1, \ldots, P_n$
- Synchronization vectors of the form $L_1, \ldots, L_n \rightarrow L$ where each L_i is either a label or the symbol • (inaction)
- Semantics of $L_1, \ldots, L_n \rightarrow L$
  - L_i transitions such that $L_i \neq \bullet$ execute synchronously in the respective P_i's
  - L is the label resulting from synchronization in the product graph
- Constraints are added so that τ transitions cannot be renamed nor cut
- Most equivalences (strong, branching, observational, ...) are congruences for synchronization networks

Example 1



can be represented by the set of synchronization vectors

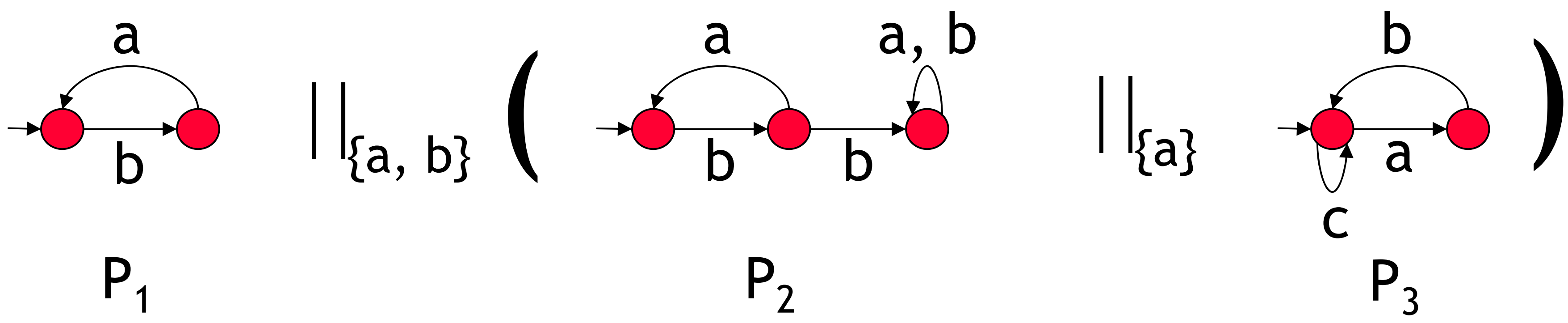
$$a, \bullet, a \rightarrow a$$

$$b, b, \bullet \rightarrow b$$

$$\bullet, c, c \rightarrow c$$

$$d, d, d \rightarrow d$$

Example 2



can be represented by the set of synchronization vectors

$a, a, a \rightarrow a$

$b, b, \bullet \rightarrow b$

$b, \bullet, b \rightarrow b$

} nondeterministic synchronization on b

$\bullet, \bullet, c \rightarrow c$

Phase 2: Interface network extraction

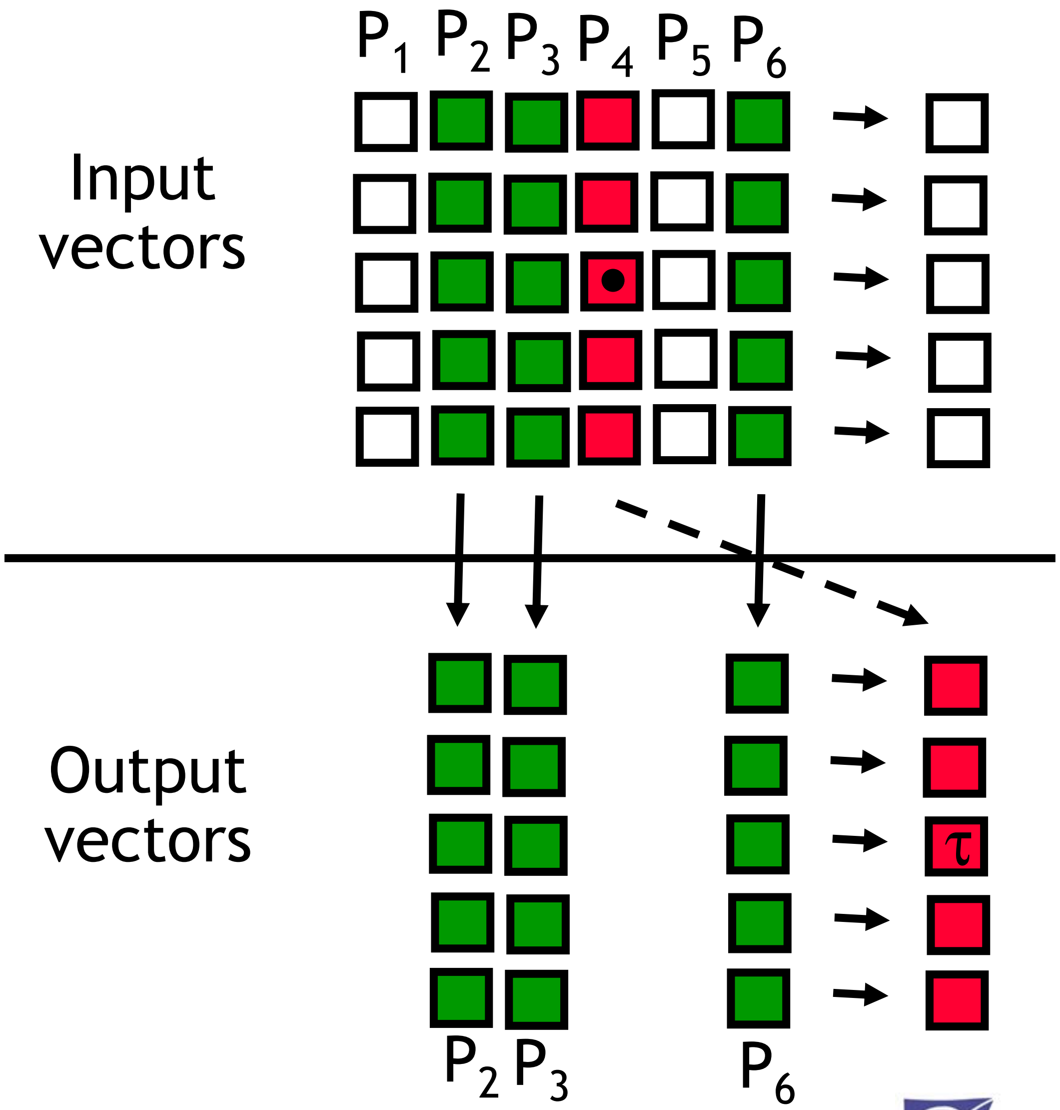
- Extraction of a network N' representing a subset of the environment of a process to be constrained
- Inputs:
  - The synchronization network N of a system $P_1, \ldots, P_n$
  - The index i of the process P_i to be constrained
  - A (user-given) set of indices $\{j_1, \ldots, j_m\}$, representing a subset $P_{j1}, \ldots, P_{jm}$ of the processes in the environment of P_i
- Algorithm: for each vector v in N, create in N' a vector

$$v[j_1], \ldots, v[j_m] \rightarrow v_i$$

where $v_i = v[i]$ if $v[i] \neq \bullet$ (P_i active)

$v_i = \tau$ otherwise (P_i inactive)

Illustration



Interface network extraction for restricting P_4 w.r.t. P_2, P_3, P_6

Example

- P_1, P_2, P_3 synchronized by the vectors

  $a, a, a \to a$

  $b, b, \bullet \to b$

  $b, \bullet, b \to b$

  $\bullet, \bullet, c \to c$

- The interface network for restricting P_2 w.r.t. P_1 is:

  $a \to a$

  $b \to b$

  $b \to \tau$

  ~~$\bullet \to \tau$~~ (This last one can be removed : only •'s in left-hand side)

Phase 3: interface graph generation

- Generate the graph corresponding to N'
- Thanks to congruence, P_{j1}, ..., P_{jm} can be reduced modulo safety equivalence beforehand
- Partial order reduction allows to avoid useless interleavings

Using the generated interface

- The (possibly large) graph of P_i can be replaced by (smaller) graph of $P_i \rceil|_A I$ where I is an interface obtained by our algorithm

- A formal proof is provided in the FORTE'2006 paper

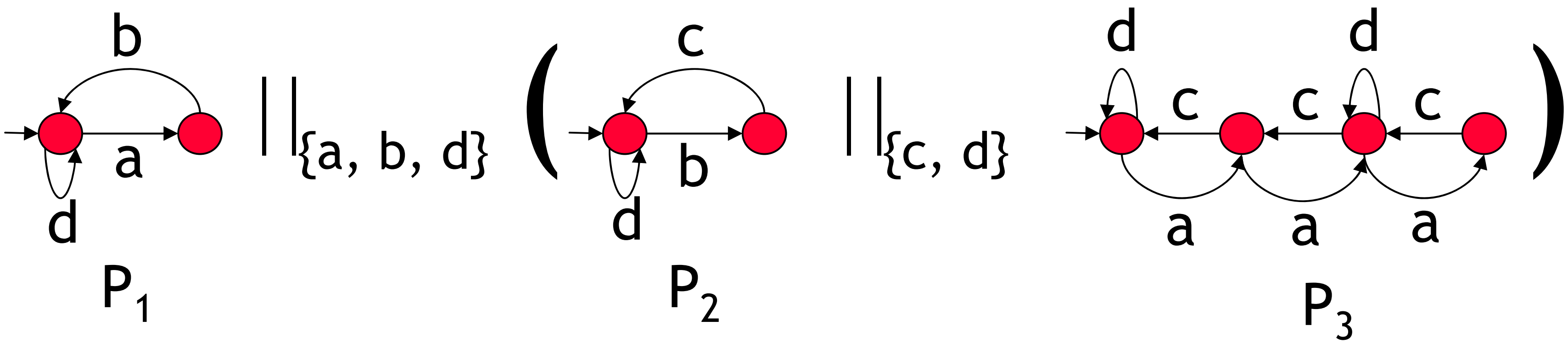
Limitation 1 solved

- Our algorithm can handle synchronization networks, a general model similar to MEC and FC2 networks
- We implemented the translation into networks for
  - CCS, CSP, LOTOS, mCRL parallel composition
  - E-LOTOS generalized parallel composition and *m* among *n* synchronization
- The translation can still be done for other operators

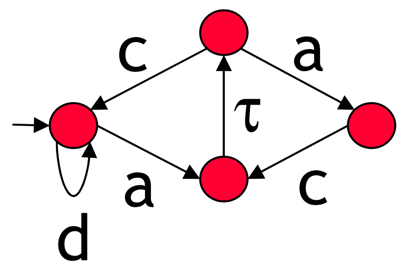
Limitation 2 solved

- Interface constraints restricting a processes w.r.t. **several** processes of its environment can be synthesized

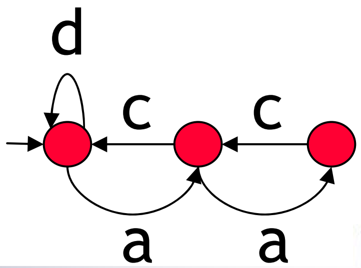
Example



Interface I for restricting P_3 w.r.t. P_1 and P_2:



$P_3 \rceil||_{\{a,\ c,\ d\}} I =$

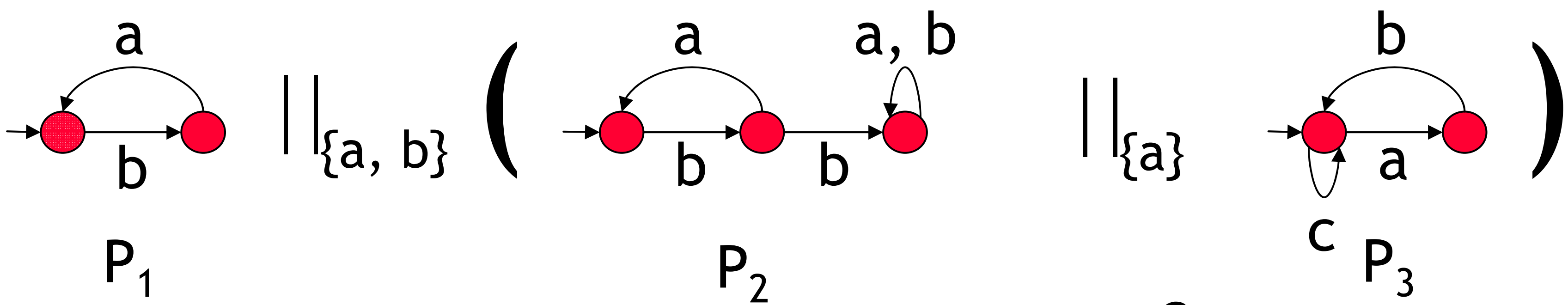


is smaller than P_3

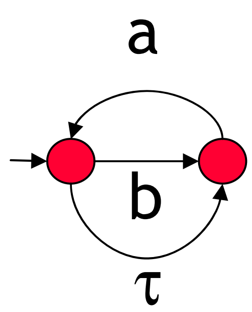
Limitation 3 solved

- Interfaces are precise even in presence of nondeterministic synchronization

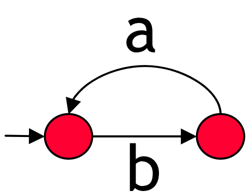
Example



Interface I for restricting P_2 w.r.t. P_1:



$P_2 \rceil||_{\{a,\ b\}} I =$



is smaller than P_2

Implementation in CADP

- Algorithm implemented in Exp.Open 2.0 (-interface option)
- Example: odp.exp

```
hide all but WORK in
    par EXPORT, IMPORT in
        par WORK #2 in
                "object_1.bcg"
        ||      "object_2.bcg"
        ||      "object_3.bcg"
        ||      "object_4.bcg"
        end par
    ||  "trader.bcg"
    end par
end hide
```

Implementation in CADP

Command:

exp.open -weaktrace -interface "5: 1 2 3" "odp.exp" \
generator "trader_interface.bcg"

- Generates an interface graph "trader_interface.bcg" for restricting the 5th graph ("trader.bcg") w.r.t. the 1st, 2nd and 3rd graphs ("object_{1,2,3}.bcg") in "odp.exp"
- Partial order reduction preserving observable traces is applied (-weaktrace)

Applications (1/3)

Philips' HAVi Home Audio-Video leader election

- Modeled in LOTOS by J. Romijn (Eindhoven)
- Largest process (404,477 states) was:
  - Reduced downto 365,923 states (182s, 46Mb) using interface obtained by K&M algorithm
  - Reduced downto 645 states (11s, 8.5Mb) using a refined interface

http://www.inrialpes.fr/vasy/cadp/demos/demo_27.html

Applications (2/3)

ODP (Open Distributed Processing) Trader

- Modeled in E-LOTOS by Garavel & Sighireanu (INRIA)
- Uses *m* among *n* synchronization to model the dynamicity of object exchanges
- Trader reduced from 1 M states without interface downto 256 states using a refined interface

http://www.inrialpes.fr/vasy/cadp/demos/demo_37.html

Applications (3/3)

Cache Coherency Protocol

- Modeled in LOTOS by M. Zendri (Bull)
- 5 agents accessing a remote directory concurrently
- No reduction using interface obtained by K&M algorithm
- Remote directory reduced from 1 M states downto 60 states using refined interface
- Directory generated for a configuration with 7 agents (81 states)

http://www.inrialpes.fr/vasy/cadp/demos/demo_28.html

Refined abstraction in SVL

- SVL: Scripting language for verification in CADP
- SVL already contained an operator written "P -|[A]| I" or "**abstraction** I **sync** A **of** P" corresponding to "$P \rceil|_A I$"
- SVL now has a new "**refined abstraction**" operator, which
  - generates the interface automatically using EXP.OPEN, and
  - restricts the process using PROJECTOR

SVL refined abstraction example

```
"cache.bcg" = root leaf strong reduction of
    (
      (AGENT_1 ||| AGENT_2 ||| AGENT_3)
      |[GET_LINE_STATUS, PUT_LINE_STATUS]|
      (refined abstraction AGENT_1, AGENT_2
              using DIR_ABSTRACT of DIRECTORY)
    );
```

Conclusions

- We provided a new algorithm to synthesize interface constraints automatically
- Our algorithm solves the 3 limitations of K&M's algorithm
  - It does not depend on a particular input language
  - It permits to take into account constraints induced by several processes
  - It permits a finer analysis of synchronization patterns between processes, thus yielding better reductions
- The method is fully implemented in CADP
- It is easy to use thanks to the SVL scripting language
- Experiments indicate possible reductions by several orders of magnitude