# State Space Reduction for Process Algebra Specifications

Hubert Garavel & Wendelin Serwe

INRIA Rhône-Alpes / VASY

655, avenue de l'Europe

F-38330 Montbonnot Saint-Martin

http://www.inrialpes.fr/vasy

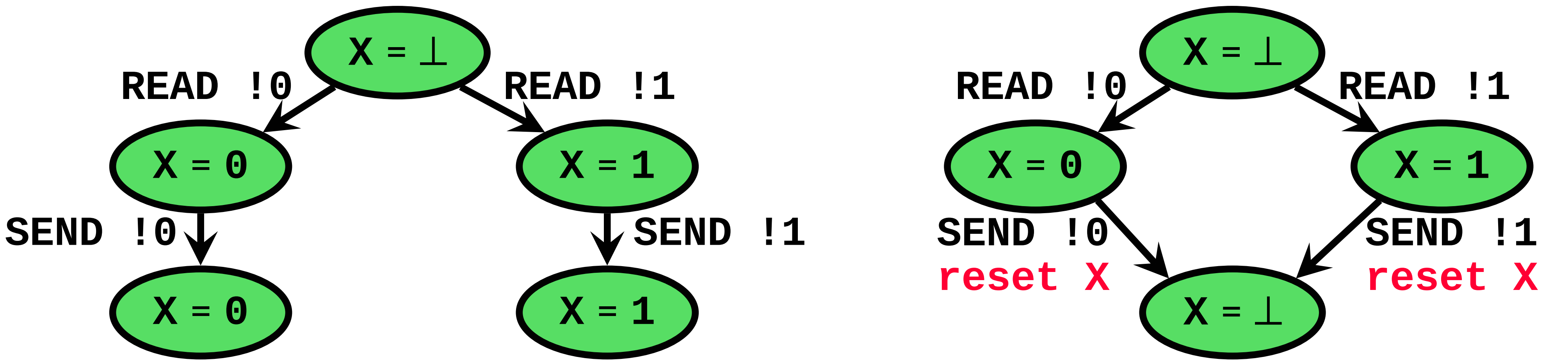
# Context

- CADP: a European verification toolbox
  - 350 licenses, 80 case studies, 17 tools using CADP
  - `http://www.inrialpes.fr/vasy/cadp`
- LOTOS: international standard (ISO 8807)
  - Based on algebraic methodology
  - Abstract data types and process algebra
- LOTOS-Compilers of CADP
  - CAESAR.ADT (data types), CAESAR (processes)
  - Generation of labeled transition systems (graphs)
  - Used in 40 demos and 60 case-studies

# Enumerative Verification

- Classical problem: **state explosion**
- Several techniques – here **resetting variables**
- [Graf-Richier-Rodríguez-Voiron 1989]:

  Manual insertion of resets in an imperative language
- Example: "**`READ ?X:bit; SEND !X; stop`**"




without reset  with reset

# Resetting Variables (1/3)

- Manual insertion of resets

  Error-prone and impossible in "assign-once" languages

- [Garavel 1992]
  - Translate LOTOS to structured Petri nets with variables

    process algebra → network model → graph

  - "Syntactic criterion":
    reset variables if places of a process loose their token
  - Significant state space reduction (CAESAR 4.2)

    process algebra → network model with resets → smaller graph

# Resetting Variables (2/3)

- [Galvez-Garavel 1993] (MSc thesis, Grenoble)
  - Attempt of a more precise analysis
  - Local and global data-flow analysis
  - Automatic insertion of resets
  - Successful state space reduction

  But: **errors** in a small number of examples
  Strong bisimulation is not preserved!
  Reason not understood $\Rightarrow$ not embedded in CAESAR

- Our goals
  - Understanding of the errors
  - Solution

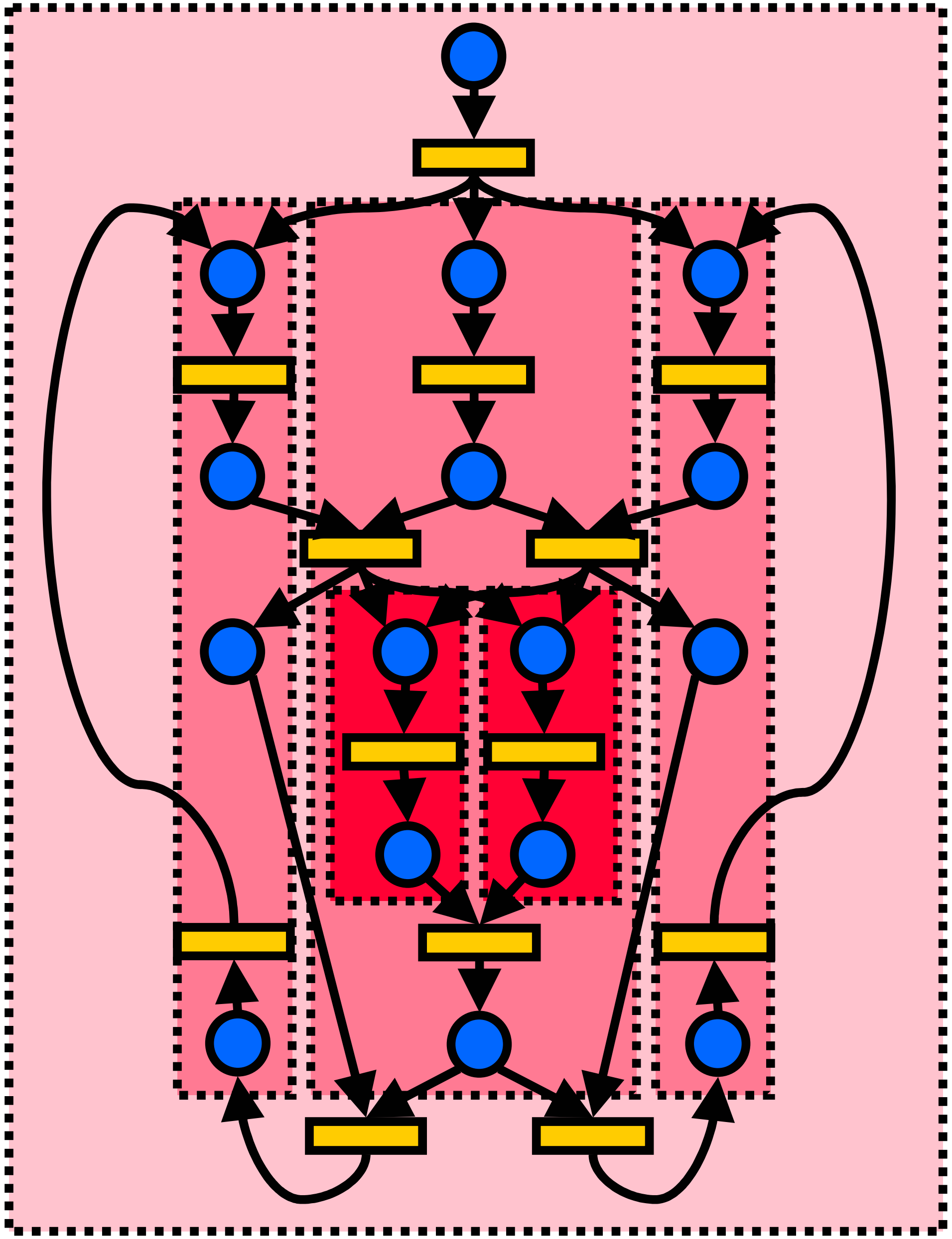
# Resetting Variables (3/3)

## Related work

- [Dong-Ramakrishnan 1999]
  - Same syntactic criterion as CAESAR
  - Removing variables instead of resetting variables
- [Holzmann 1999]
  - Imperative language
  - Simpler model: *flat* collection of processes
- [Bozga-Fernandez-Ghirvu 1999]
  - Simpler model: *flat* collection of processes
  - Provides correctness proofs

# Network Model of CAESAR

# Network Model of CAESAR (1/2)

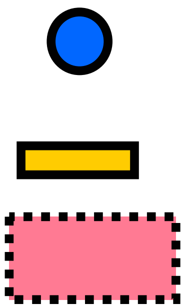


## Structured Petri Nets

- Places
- Transitions
- Units
  - Partition of the places
  - Subunit relation: $\subseteq$

## Properties of units

- Tree shaped hierarchy
- At most 1 marked place
- $U_1$ and $U_2 \subseteq U_1$ are not marked simultaneously

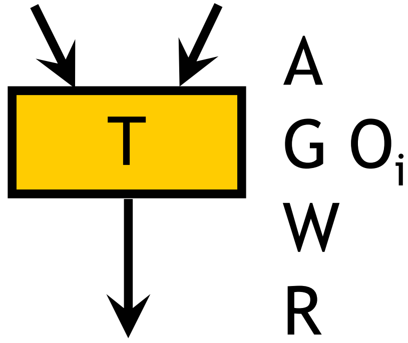


# Network Model of CAESAR (2/2)

## Typed variables

- Attached to units
- Modified by transitions:

  Action A, offer O, guard W, reaction R




## Properties of variables

- Variables are defined before used
- Shared variables are read-only

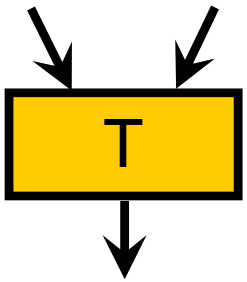
  In the LOTOS behavior: "**`G ?X:S; (P1 ||| P2)`**"
  - "**`X`**" can be read by "**`P1`**" and "**`P2`**"
  - "**`X`**" cannot be modified by "**`P1`**" or "**`P2`**"

# Local Data-Flow Analysis

# Local Data-Flow Analysis

- Intra-transition
- Predicates on transition T and variable X

  defined by structural induction on T (i.e., A, O, W, R)
  - ***use*(T, X)**: value of X accessed by T
  - ***def*(T, X)**: value of X defined *at the end* of T
  - ***use_before_def*(T, X)**: value of X accessed *at the beginning* of T, i.e., *before* a possible *redefinition*
- Example




```
COUNT := COUNT+1
READ ?VALUE
when VALUE > COUNT
```

*def*(T, **COUNT**), *def*(T, **VALUE**)
*use*(T, **COUNT**), *use*(T, **VALUE**)
*use_before_def*(T, **COUNT**)

# Global Data-Flow Analysis

# Global Data-Flow Analysis

- Inter-transition: combine local results
- Classically (sequential programs) compute fixed point on (control-flow) graph
- Principal difference: Concurrency Petri nets instead of graphs
- Idea: abstract Petri nets to graphs
  - Nodes: transitions
  - Arcs: successor relation "$T_1 \rightarrow T_2$"

# Abstracting Networks to Graphs

Several possibilities:

- Good precision: based on reachable markings
  - "$T_1 \rightarrow_M T_2$" iff exists firable sequence "..., $T_1$, $T_2$"
  - State explosion possible
- Poor precision: connection by places
  - "$T_1 \rightarrow T_2$" iff $(\exists\ Q)$ Q output of $T_1$ and Q input of $T_2$
  - Simple, but imprecise
- Improvement: analyze variables one by one
  - "$T_1 \rightarrow_X T_2$" iff $(\exists\ Q)$ as above and Q in unit of X
  - Chosen approach

# Global Data-Flow Predicates

*live*$(T_0, X)$ iff
$(\exists\ T_0 \rightarrow_X \ldots \rightarrow_X T_n)$
*use_before_def*$(T_n, X)$
and
$(\forall i \in \{1, \ldots, n-1\})$
$\neg$*def*$(T_i, X)$

Backward fixed point

*available*$(T_n, X)$ iff
$(\exists\ T_0 \rightarrow_X \ldots \rightarrow_X T_n)$
*def*$(T_0, X)$
and
$(\forall i \in \{0, \ldots, n-1\})$
*live*$(T_i, X)$

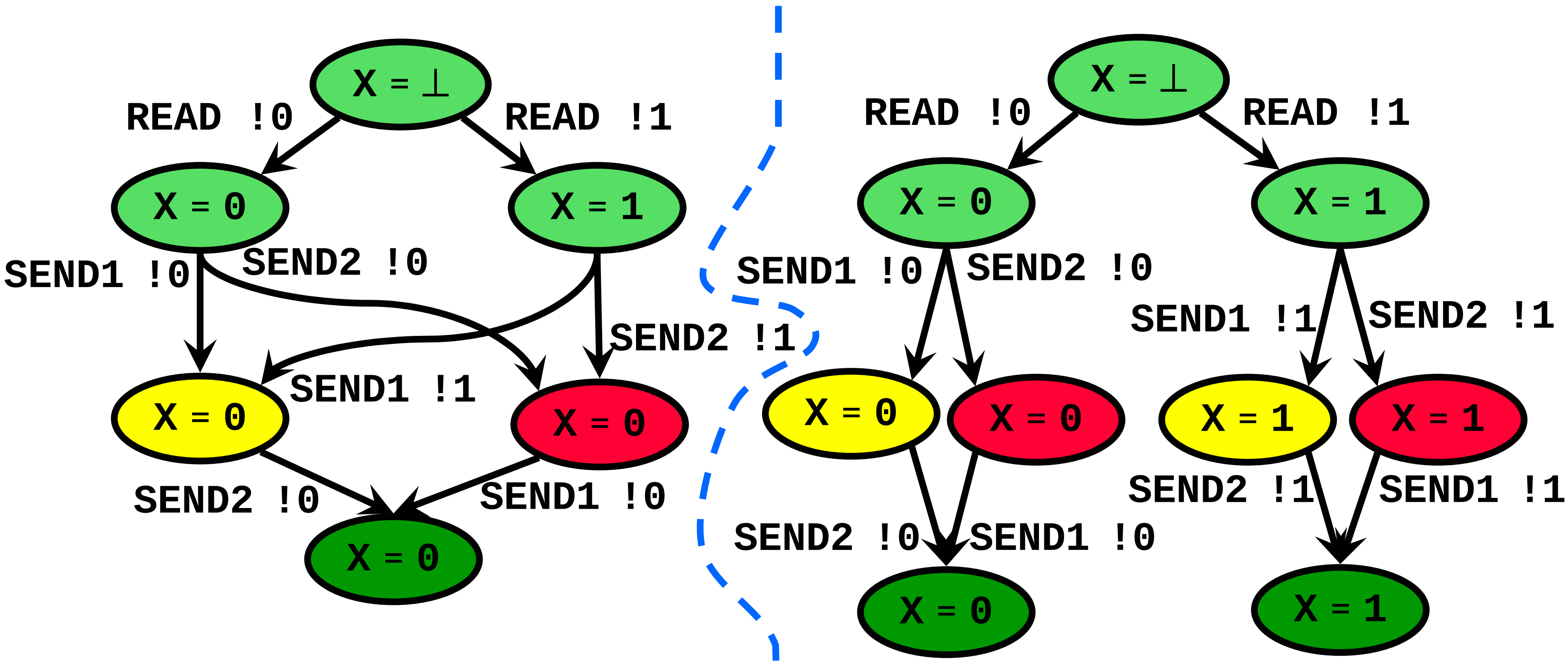
Forward fixed point

*reset*$(T, X)$ iff *available*$(T, X)$ and $\neg$*live*$(T, X)$

# Treatment of Inherited Shared Variables

# Resetting Shared Variables (1/3)

```
READ ?X: bit;
      (SEND1 !X; stop ||| SEND2 !X; stop)
```

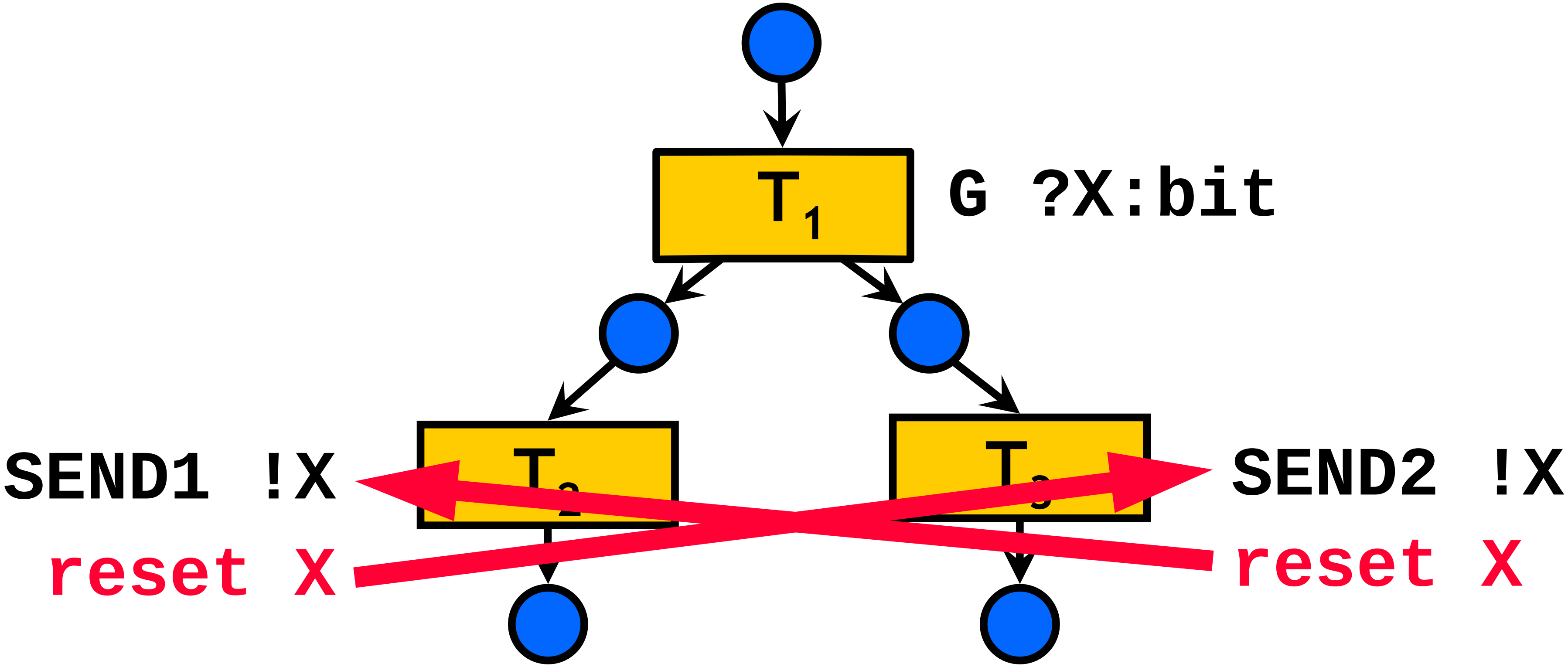



incorrect graph
(with resets; ⊥ = 0)

correct graph
(without resets)

# Resetting Shared Variables (2/3)

```
READ ?X:bit;
     (SEND1 !X; stop ||| SEND2 !X; stop)
```

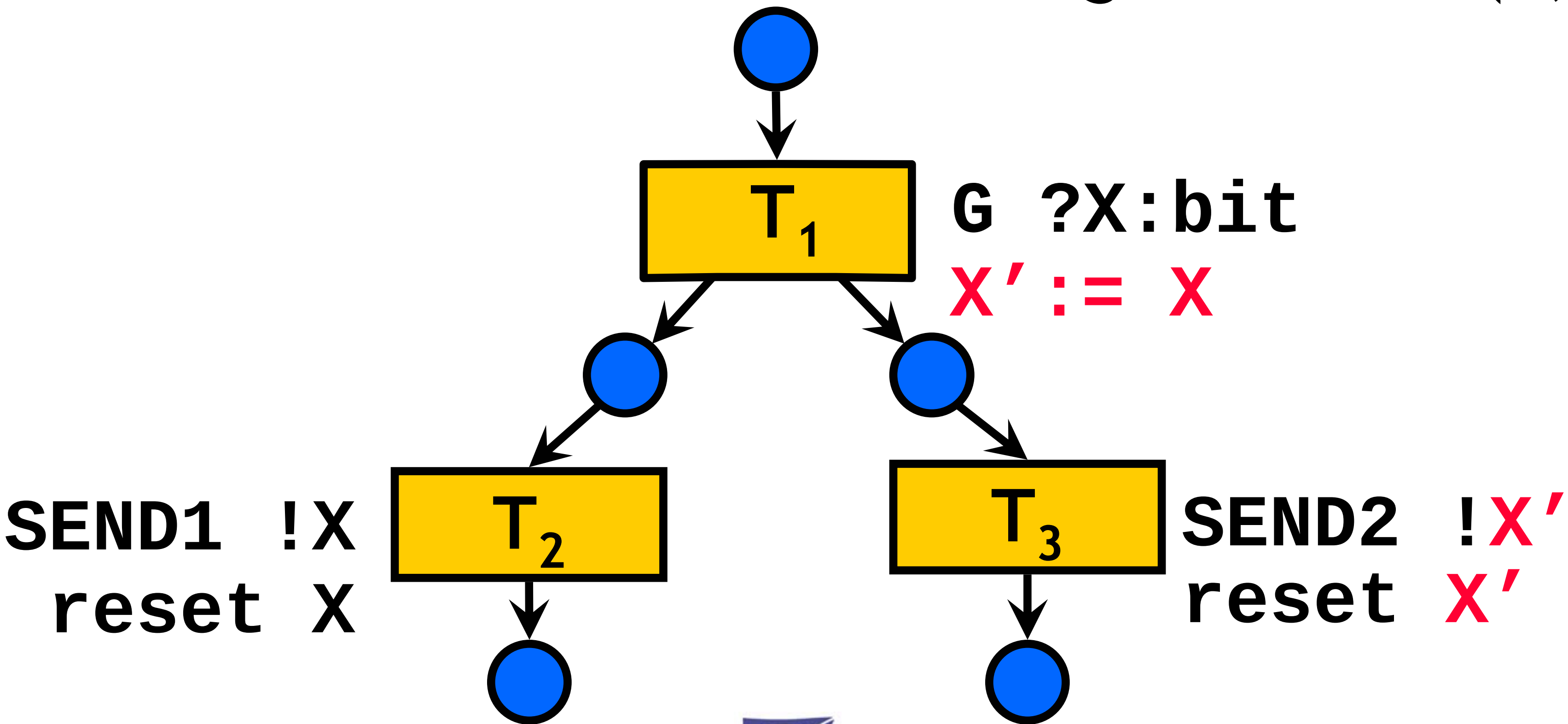



- Without resets, shared variables are read-only
- Inserting resets creates read/write(reset) conflicts

# Resetting Shared Variables (3/3)

## Solution: Duplication of "X" in unit "U"

- Create a new variable **X'** attached to U
- Replace **X** by **X'** in all transitions of U
- Insert "**X' := X**" in all T entering U s.t. *Live*(T, **X**)

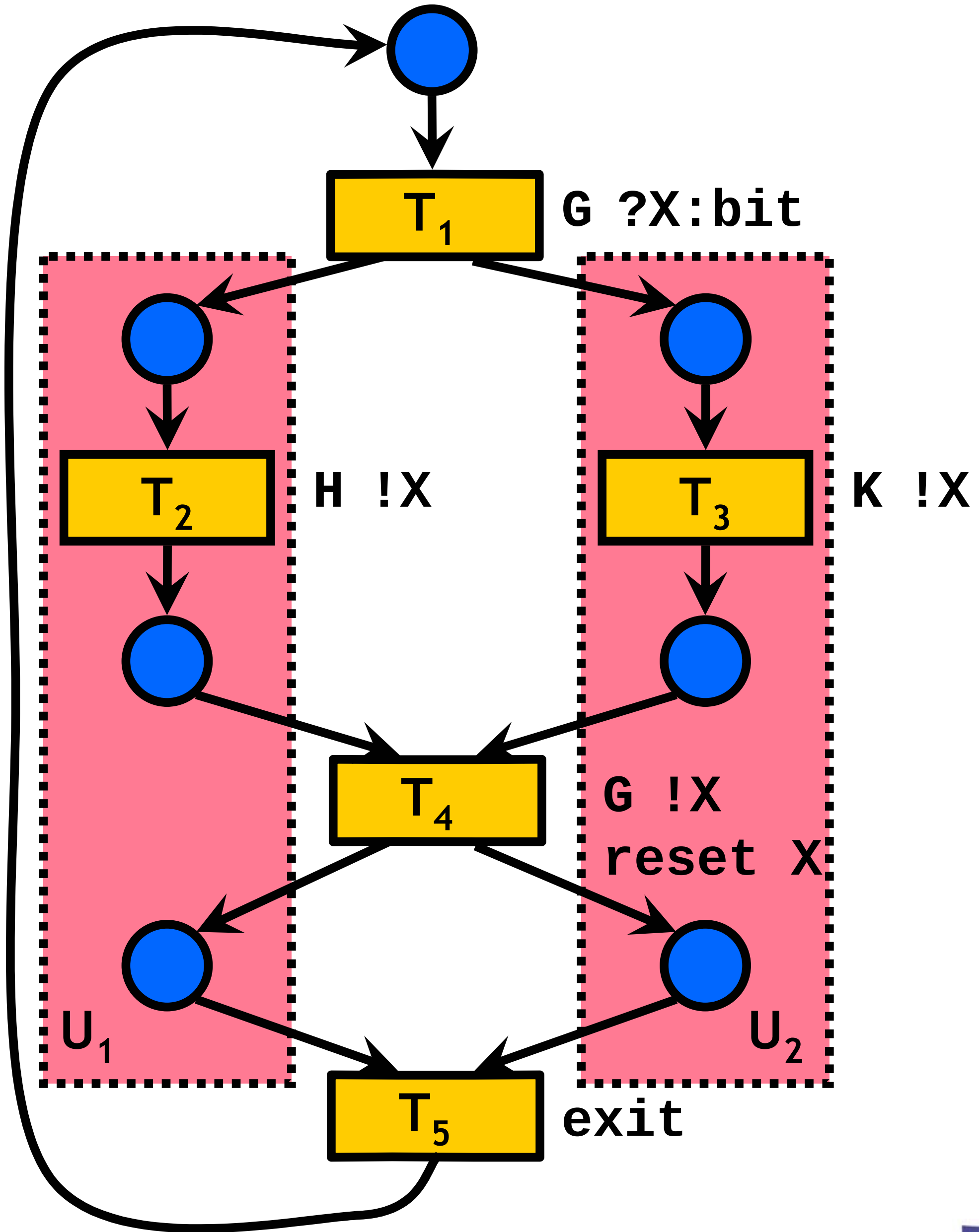


# Which Variables to Duplicate? (1/2)

- Variable duplication increases the representation of a state!
- Goal: minimal number of duplicated variables
- **Concurrent Units**: "$U_1$ ||| $U_2$"
  $U_1$, $U_2$ separate and simultaneously marked
- **Conflict** *use*($T_1$, X) versus *reset*($T_2$, X) iff
  $T_1$ transition of $U_1$, $T_2$ transition of $U_2$, and $U_1$ ||| $U_2$

  **Too rough!**

# Which Variables to Duplicate? (2/2)




- *use*($T_2$, **X**), *use*($T_3$, **X**), *use*($T_4$, **X**)
- *reset*($T_4$, **X**)
- $T_2$ transition of $U_1$
- $T_3$ transition of $U_2$
- $T_4$ transition of $U_1$ and $U_2$
- $U_1$ ||| $U_2$

**Conflict $T_2$ ($T_3$) with $T_4$?**

**NO:**

- $T_4$ synchronizes $U_1$, $U_2$
- "**reset X**" in $T_4$ correct

# Algorithm

compute *concurrent units* and *synchronizing transitions*

*VARS* := { $X_1$ … $X_n$ }

**while** *VARS* not empty **do**

  choose **X** in *VARS*

  **repeat**

    compute local and global data-flow

    compute conflicts

    **U** := choose conflicting unit

    **if U <> NULL then**

      duplicate **X** in **U** (yields **X'** )

      *VARS* := *VARS* $\cup$ { **X'** }

  **until U = NULL** (i.e., no more conflicts)

  insert "**reset X**" in all **T** such that "*reset*(**T, X**)"

# Experimental Results

# Experimental Results

- Tests: 544 LOTOS value-passing specifications
- State space reduction for 120 examples (22%)
- Average reduction factors
  States: **9** (max 220), Transitions: **12** (max 360)
- 3 examples: generation impossible before
  **reduction factor > $10^4$**
- Generating all graphs: 4 times faster
- Only 24 examples (4%) requiring duplication
- Increase of state representation outweighed by state space reduction

# Conclusion

- Resetting variables in process algebra
  - Translation to structured Petri nets with variables
  - Local and global data-flow analysis
- Rich model: Hierarchy of nested processes
- Tests on 544 examples: reductions up to $10^4$

# Open issues

- Unrestricted creation/destruction of processes
- Handling of shared read-write variables