On-the-Fly State Space Reductions for Weak Equivalences

Radu Mateescu

INRIA Rhône-Alpes / VASY

http://www.inrialpes.fr/vasy

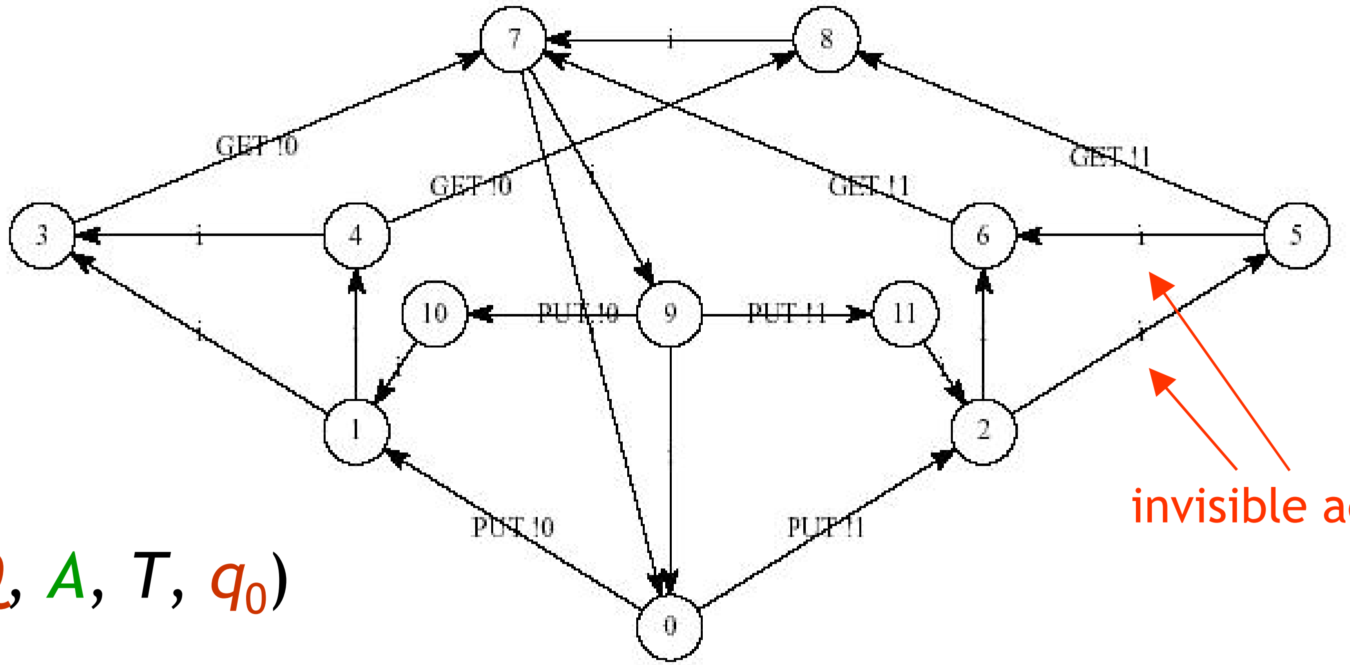
Outline

- Introduction
- On-the-fly reductions
  - Tau-compression
  - Tau-closure
  - Tau-confluence
- Applications
  - State space generation
  - Model checking
  - Equivalence checking
- Conclusion and future work

On-the-fly verification

- Characteristics:
  - Applicable to finite-state concurrent systems
  - Demand-driven construction of the state space
  - Can detect errors in (very) large systems
  - Simple way to fight against state explosion
- "Traditional" methods:
  - Equivalence checking (bisimulations)
  - Model checking (temporal logics)
- Objective:
  - Further improve performance of on-the-fly verification
  - Develop generic, reusable modules

Labelled Transition Systems



invisible actions (τ)

$M = (Q, A, T, q_0)$

CADP toolbox (http://www.inrialpes.fr/vasy/cadp)

- Explicit representation (succ/pred function)
  - BCG (Binary Coded Graphs)
- Implicit representation (successor function)
  - OPEN/CAESAR [Garavel-98]

On-the-fly LTS reductions

- Idea: insert a *reductor module* ("accelerator") in front of the on-the-fly verification tool
- Requirements for the reductor module:
  - Must work on-the-fly
    → forward traversal of the LTS following τ-transitions
  - Must be compatible with the verification problem
    → preserve *weak* equivalence relations on LTSs
  - Must enhance performance whenever possible
    → overhead compensated by the reduction achieved
- Implemented using Open/Caesar
  - Reductor : implicit LTS → reduced implicit LTS
  - Language-independent and application-independent

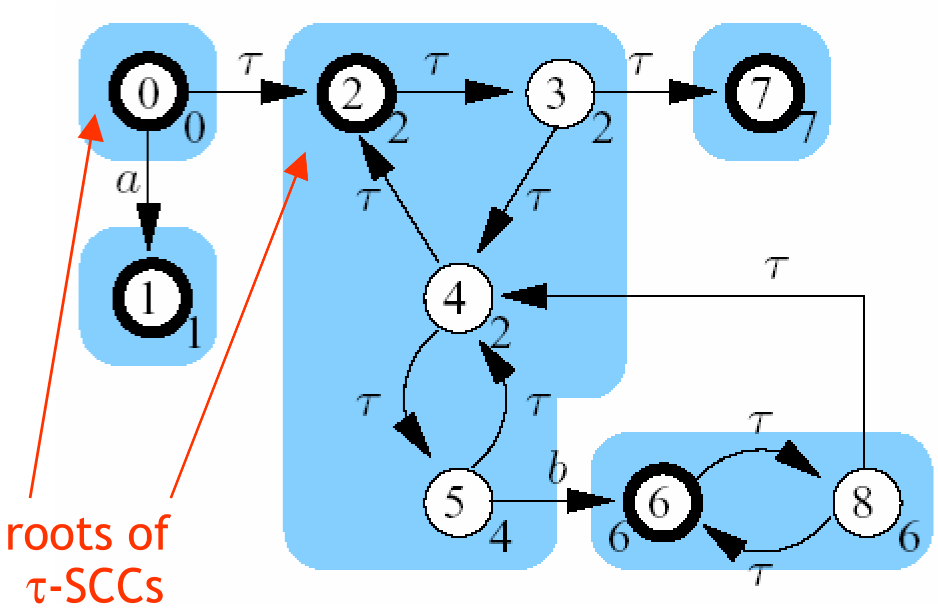
Related work
(on-the-fly reductions on LTSs)

- Property-driven reductions
  - Selective μ-calculus [Barbuti-et-al-99]
  - Equivalence derived from the formula being checked
- Transitive reflexive closure over τ-transitions
  - Algorithms based on graph traversal [Ioannidis-et-al-93]
  - Applied for test generation [Jeron-Morel-97]
    → algorithm avoiding recomputations
- Partial order reductions
  - Compatible with observational equivalence / weak mu-calculus / action LTL [Smolka-Liu-99, Magee-Kramer-99]
  - Tau-confluence
    - Global algorithm [Groote-vandePol-00]
    - Local algorithms [Blom-vandePol-02, Pace-Lang-Mateescu-03]

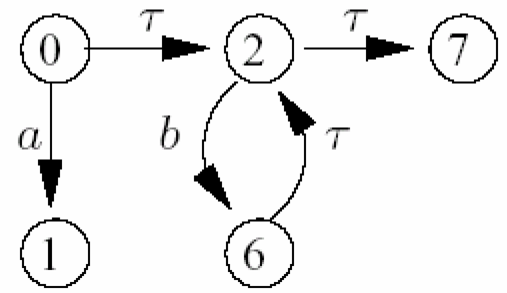
Tau-compression

- Collapsing of strongly connected components containing only τ-transitions (τ-SCCs)
- Preserves branching equivalence
- Algorithm:
  - Depth-first search (DFS) along τ-transitions
  - Detection of τ-SCCs [Tarjan-72]
  - Root of τ-SCC: representative for all states of the τ-SCC
  - Successors of representative = successors of all states in the τ-SCC
- Complexity:
  - Linear in the LTS size

Example



LTS with its τ-SCCs and their representatives

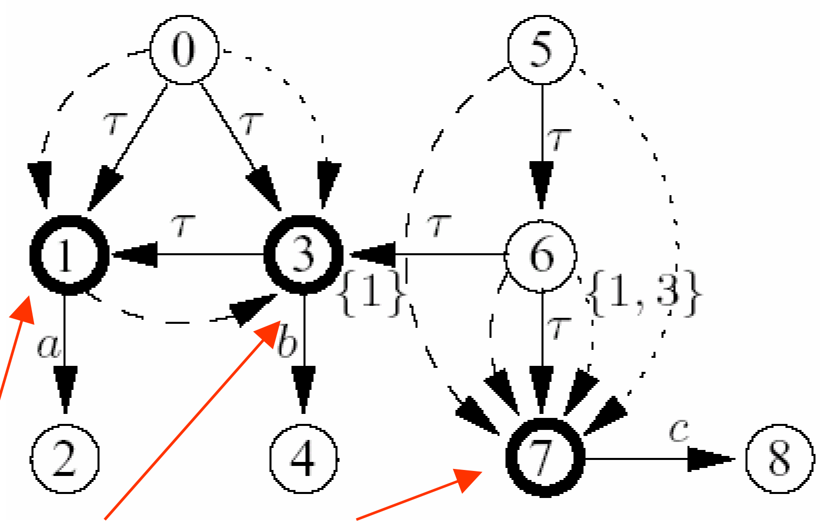


Reduced LTS after calling τ-compression on states 0 and 6

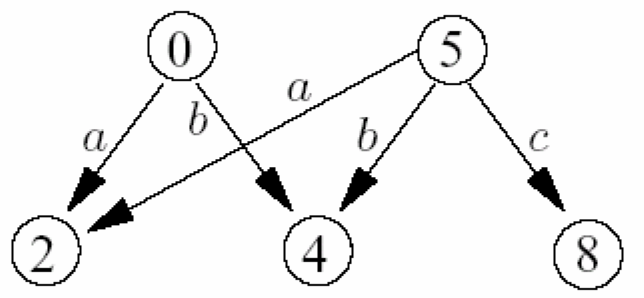
Tau-closure

- Transitive reflexive closure over τ-transitions
- Preserves $\tau^*.a$ equivalence
- Algorithm:
  - Assumes no τ-cycles (apply τ-compression first)
  - 1st DFS over τ-transitions
    - Compute reachable segments on the frontier of the DFS forest
    - Compute cross τ-transitions (relating neighbour DFS subtrees)
  - 2nd DFS over cross τ-transitions
    - Compress sequences of cross τ-transitions
- Complexity:
  - Linear in the LTS size (first call)
  - Increase towards quadratic (subsequent calls)

Example



states in the frontier
of the DFS forest



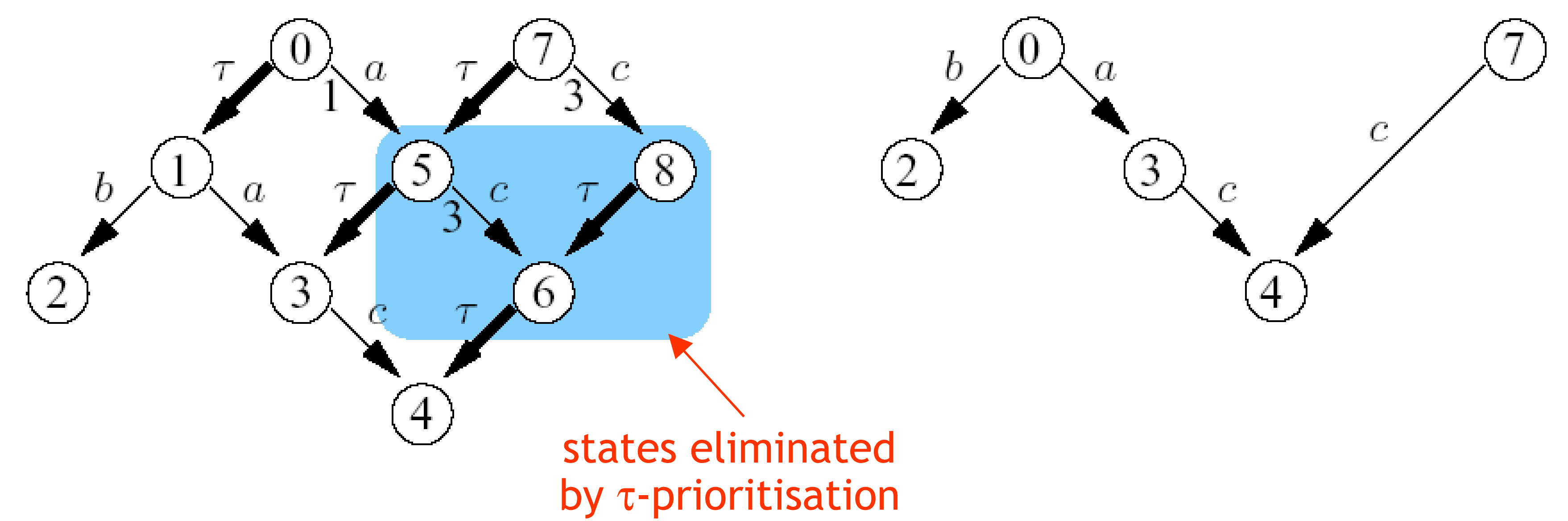
LTS with fields *next* (dashed arrows), *last* (dotted arrows), and *cross* (marked when not empty)

Reduced LTS after calling τ-closure on states 0 and 5

Tau-confluence

- Identify confluent τ-transitions [Groote-vdPol-00]
  - Delete neighbours of confluent τ-transitions (τ-prioritisation)
  - Confluent τ-transitions can be collapsed
- Preserves branching equivalence
- Algorithm:
  - Assumes no τ-cycles (apply τ-compression first)
  - Detects confluent τ-transitions by a local resolution of a Boolean Equation System [Pace-Lang-Mateescu-03]
  - Collapse sequences of confluent τ-transitions
- Complexity:
  - Linear in LTS size and quadratic in LTS branching factor

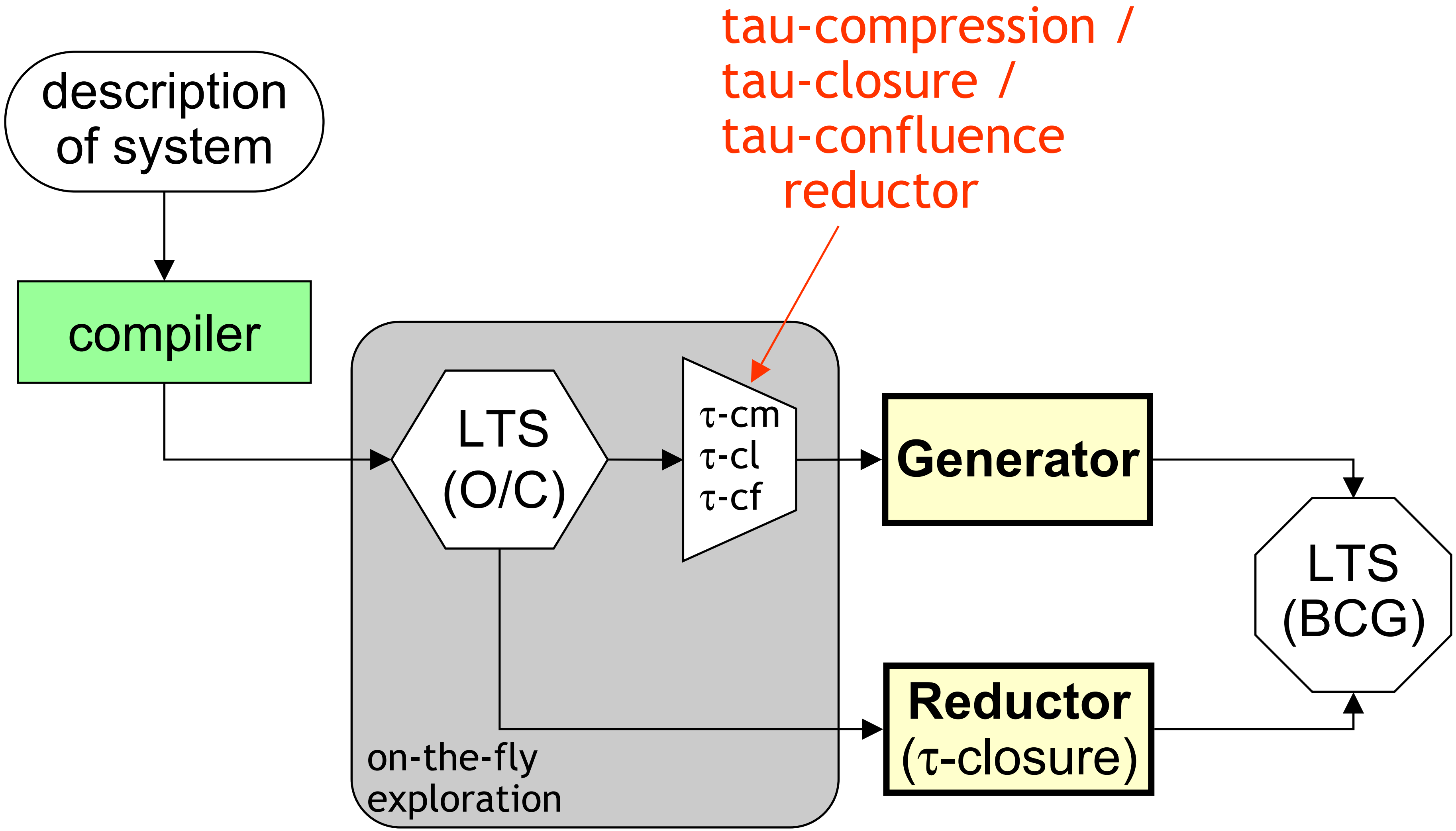
Example



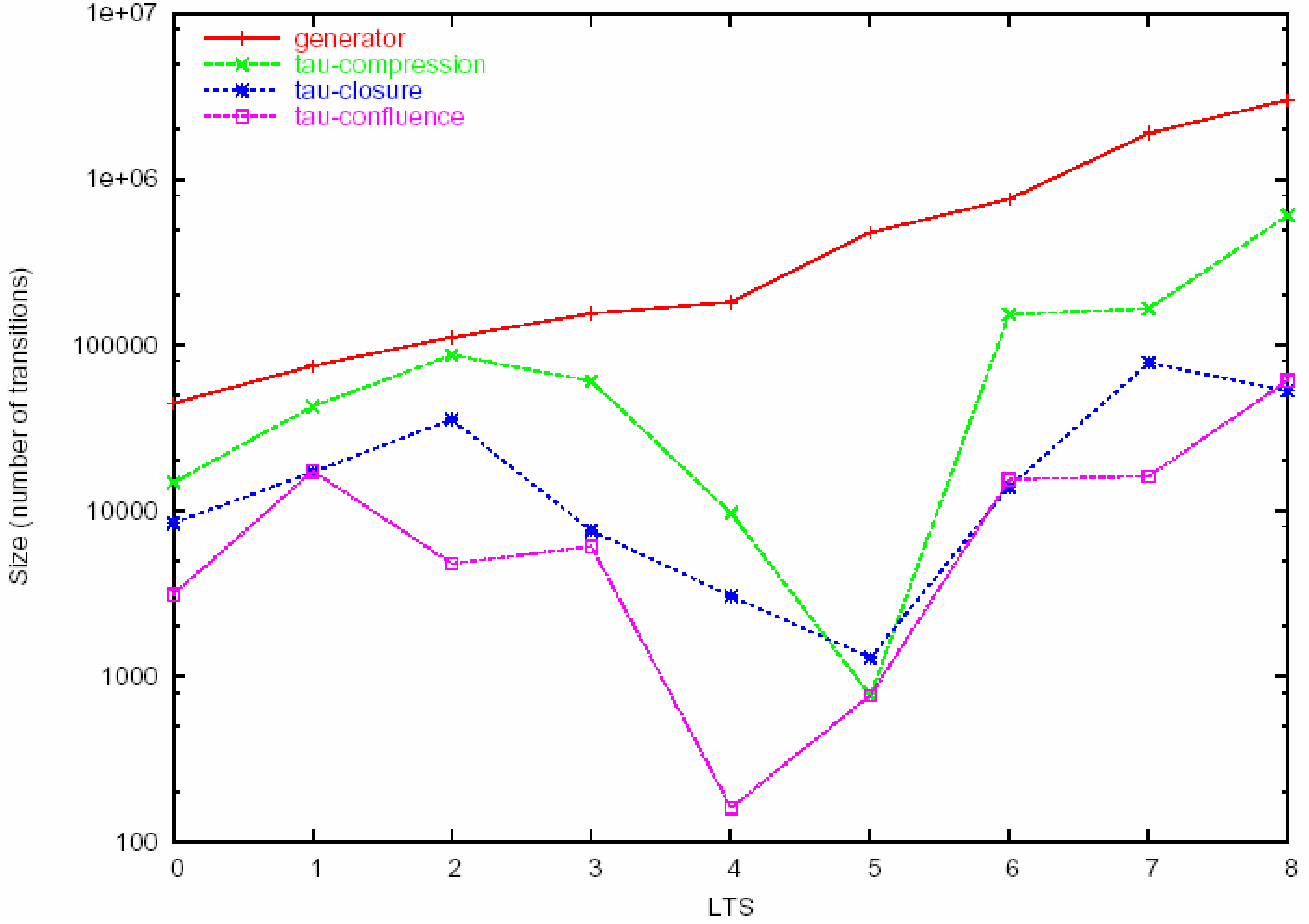
LTS with τ-confluent transitions (thick arrows) and state representatives computed by collapsing τ-sequences

Reduced LTS after calling τ-confluence on states 0 and 7

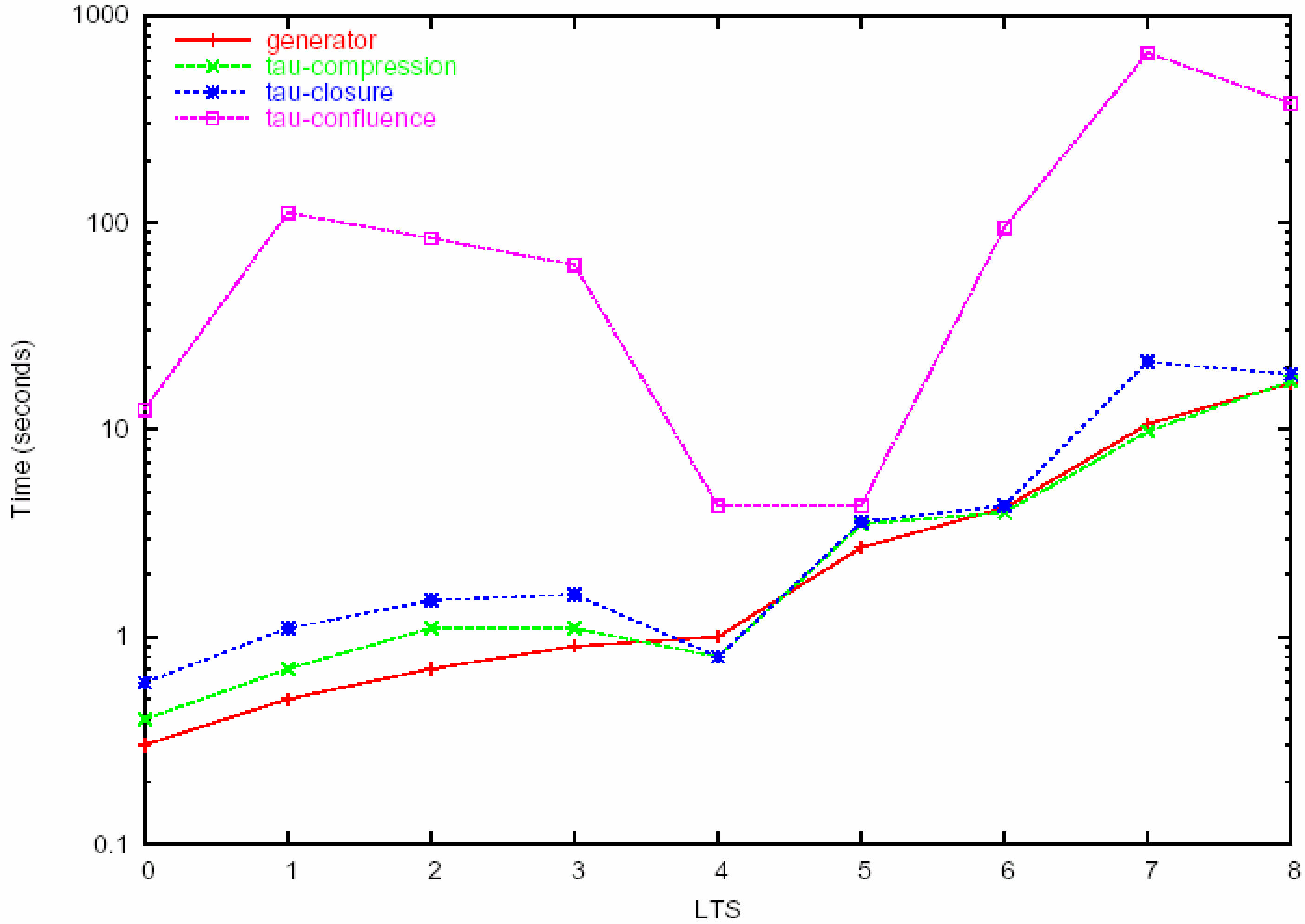
State space generation



Generator/reductors vs. Generator
(LTS size)



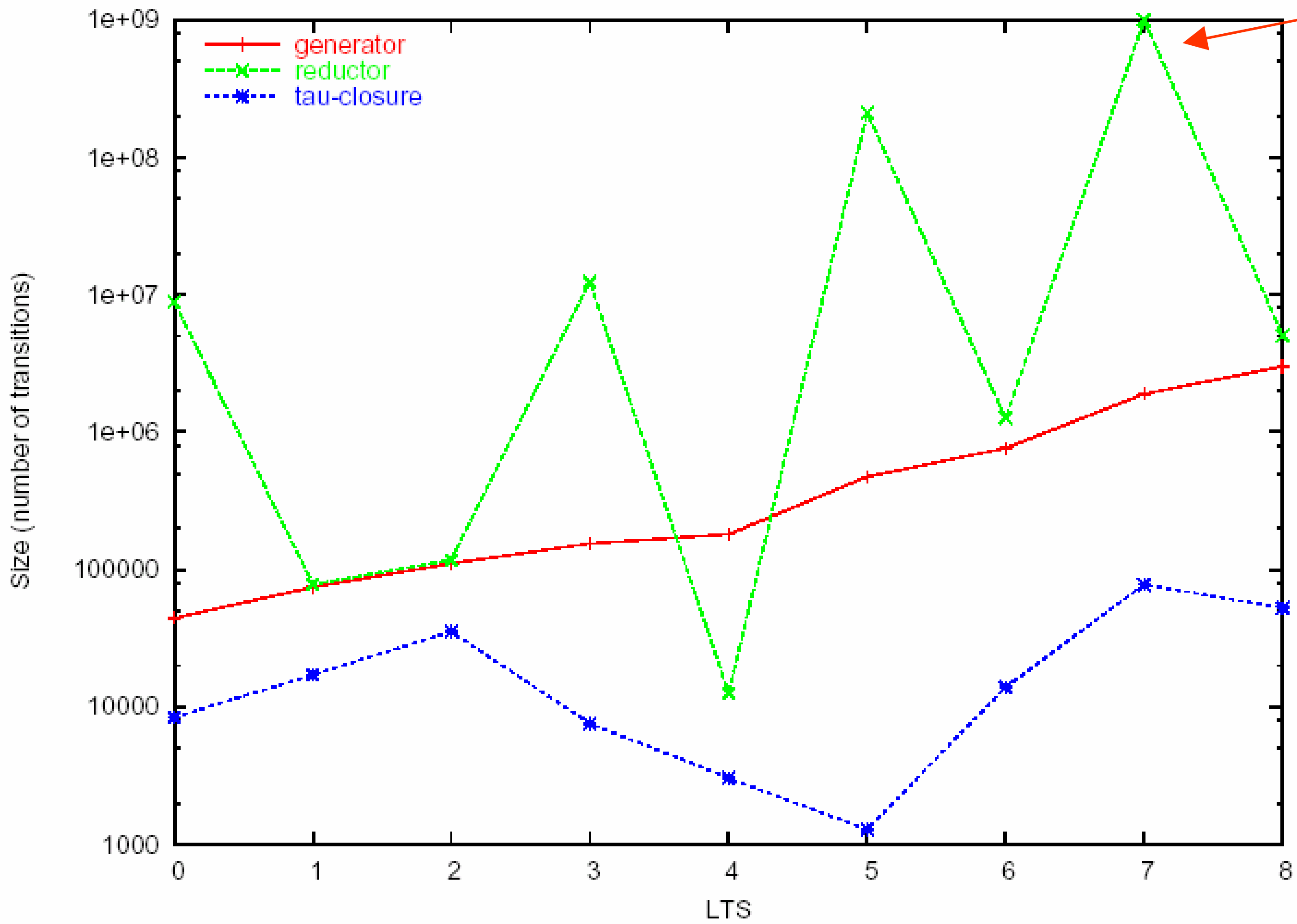
Generator/reductors vs. Generator
(time)



Generator/τ-closure vs. Reductor

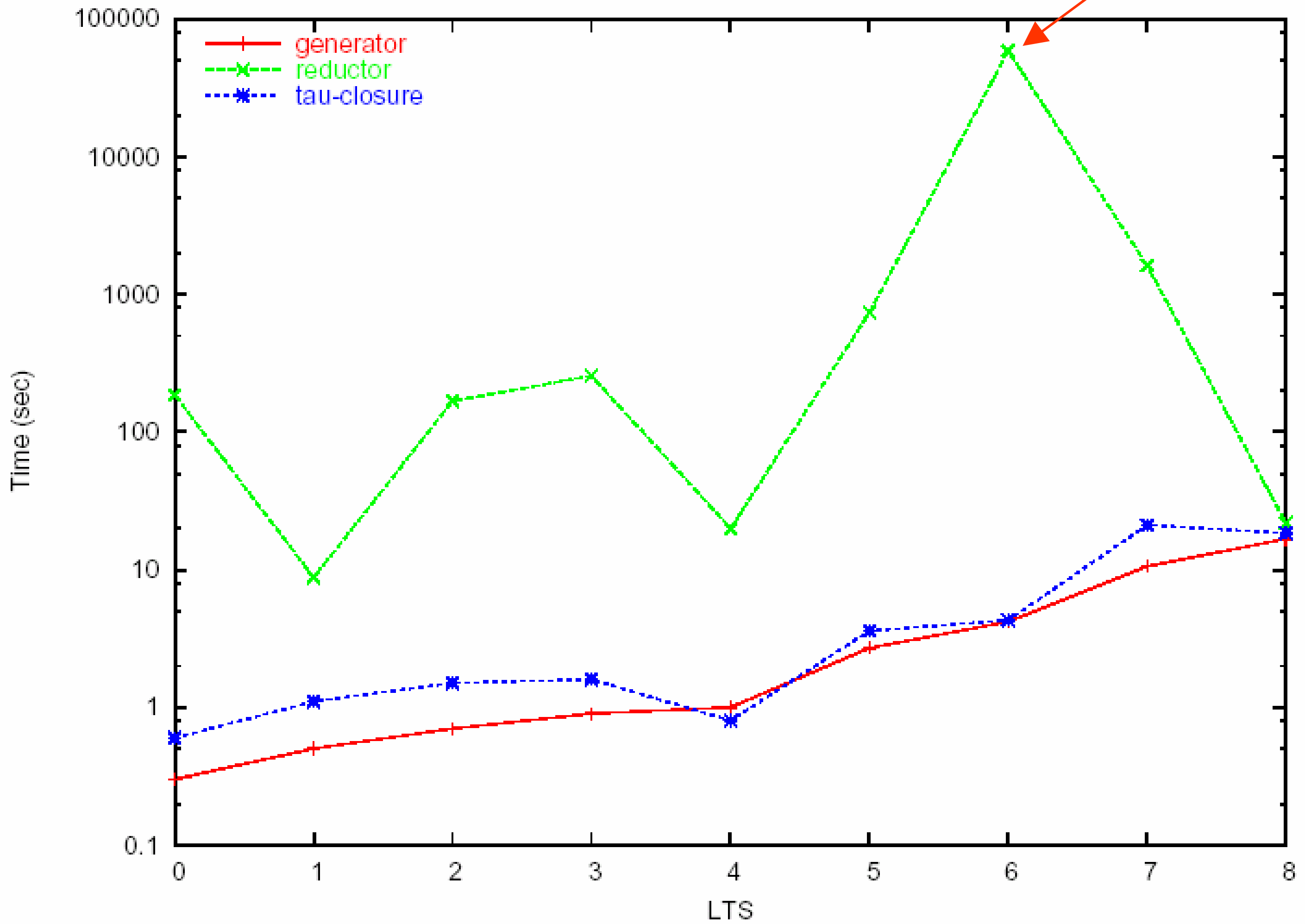
(LTS size)

disk space shortage
(too many transitions)

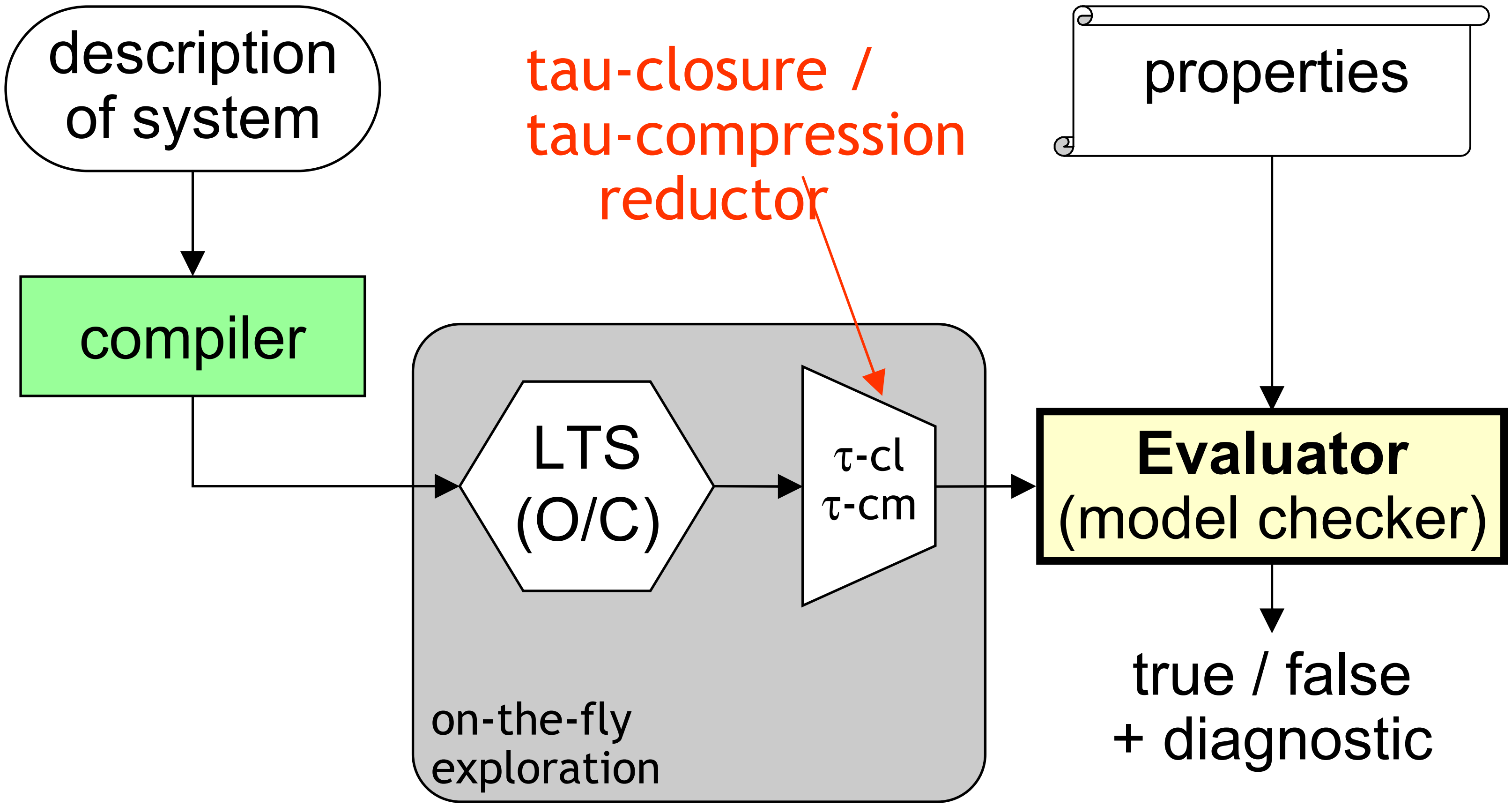


Generator/τ-closure vs. Reductor
(time)

disk space shortage
(too many transitions)



Model checking



Temporal logic properties

- Regular alternation-free μ-calculus [Mateescu-Sighireanu-02]
- Two properties considered:

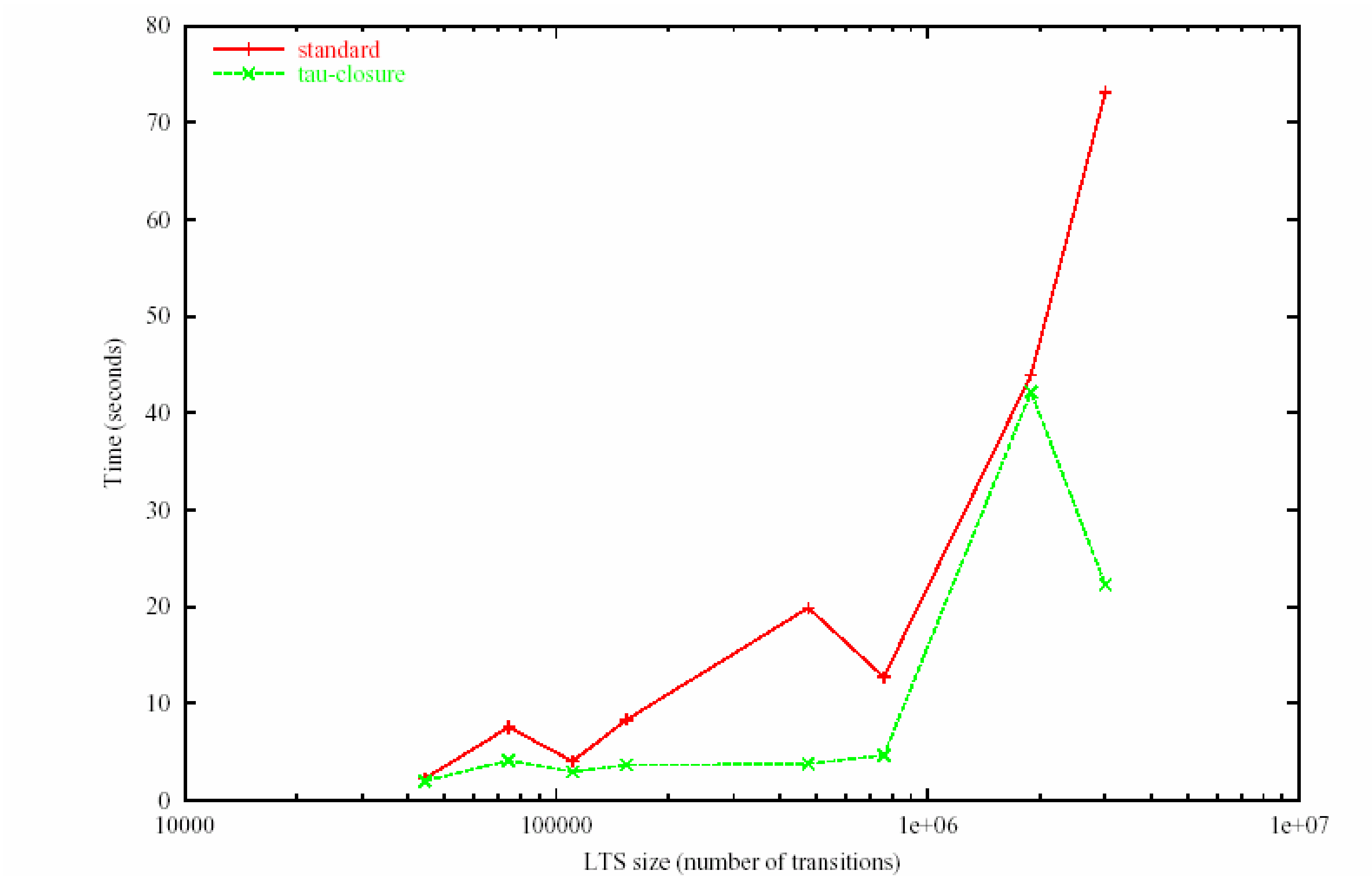
  P1: [true* . a . (not b)* . c] false

  safety / τ*.a / τ-closure

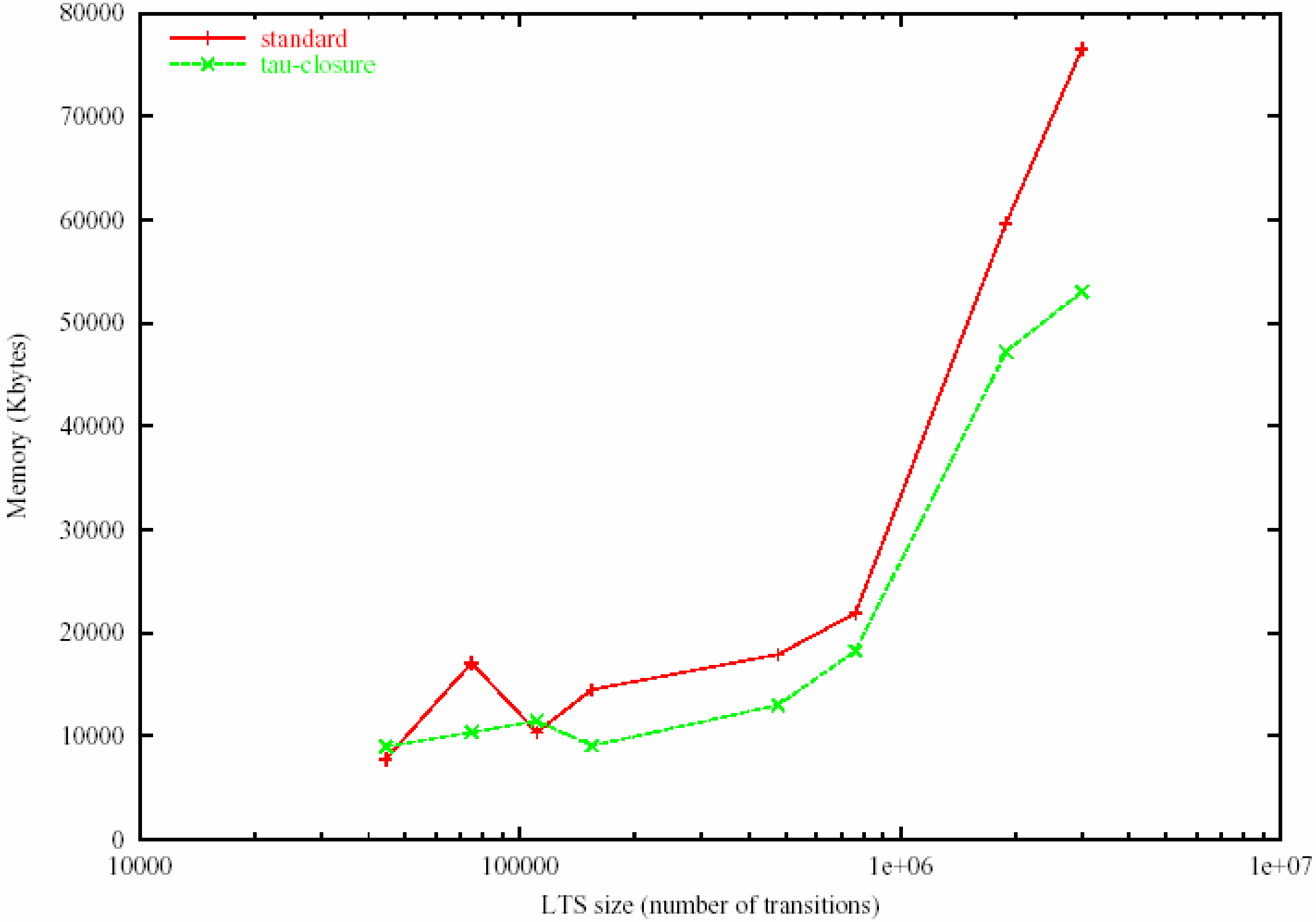
  P2: [true* . a] < true* . b > true

  liveness / branching / τ-compression
- Actions a, b, c are chosen such that P1, P2 are true (worst-case)
- Actions other than a, b, c are hidden during check (increase reduction)

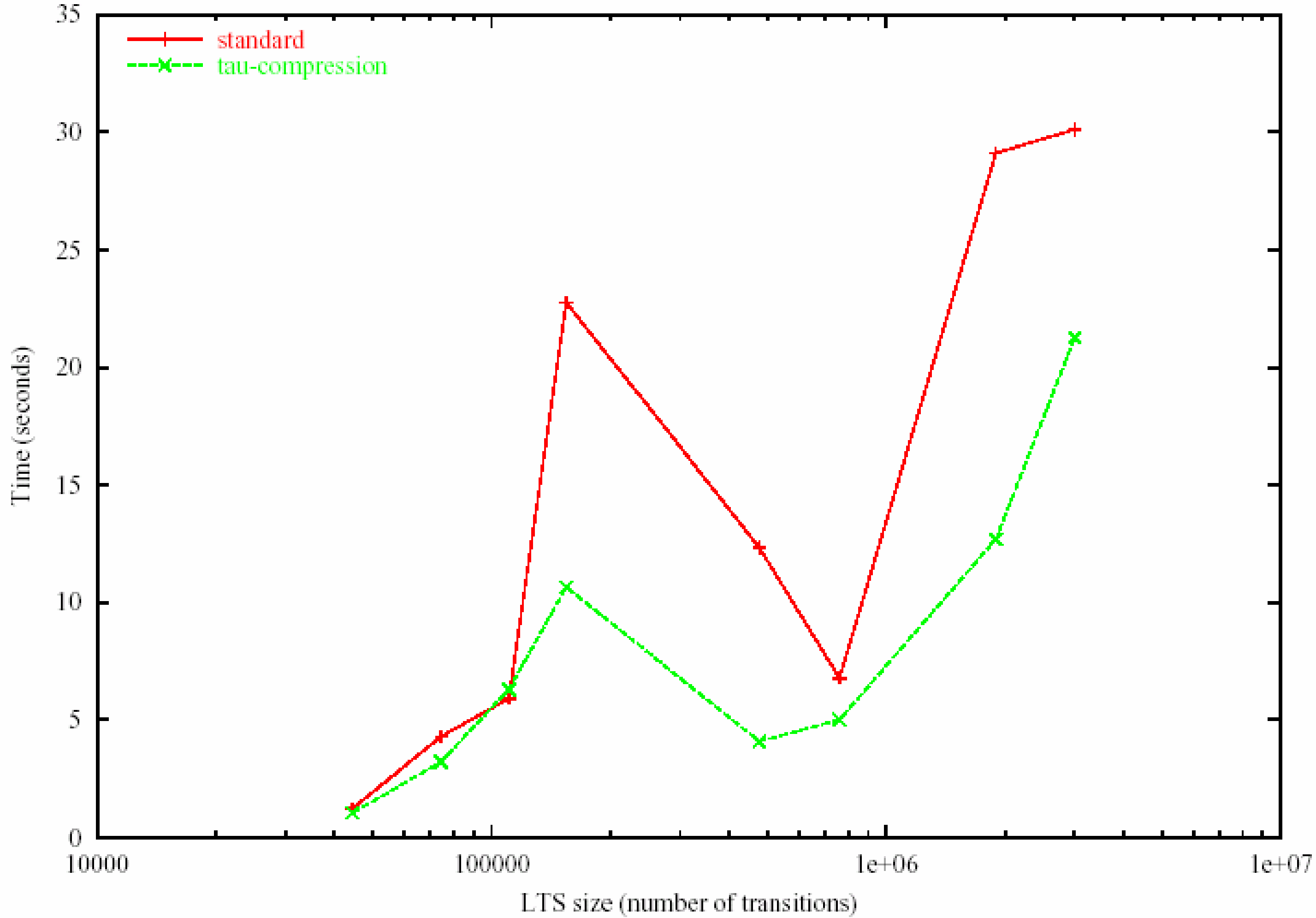
Evaluator/τ-closure vs. Evaluator
(time - property P1)



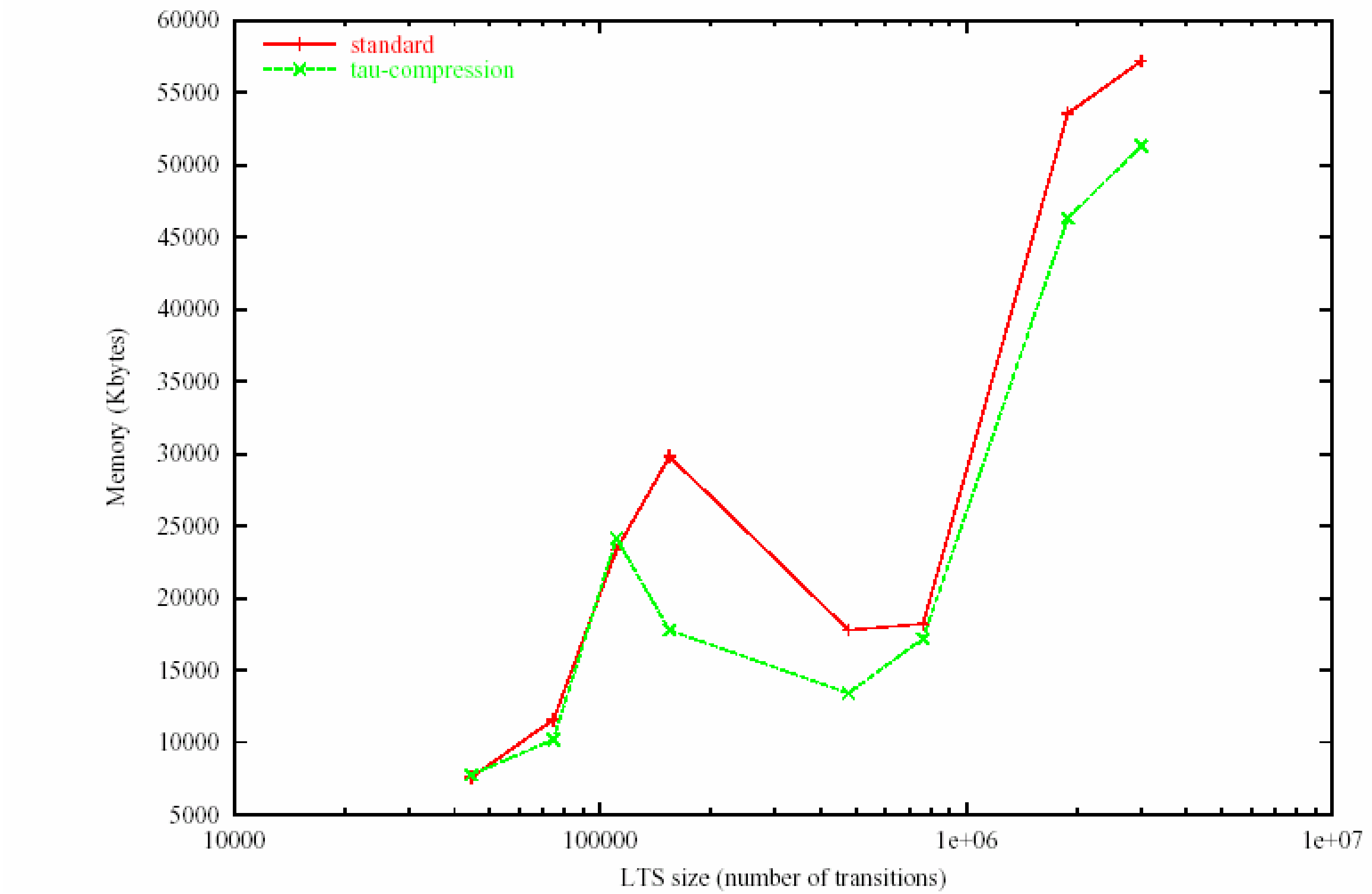
Evaluator/τ-closure vs. Evaluator
(memory - property P1)



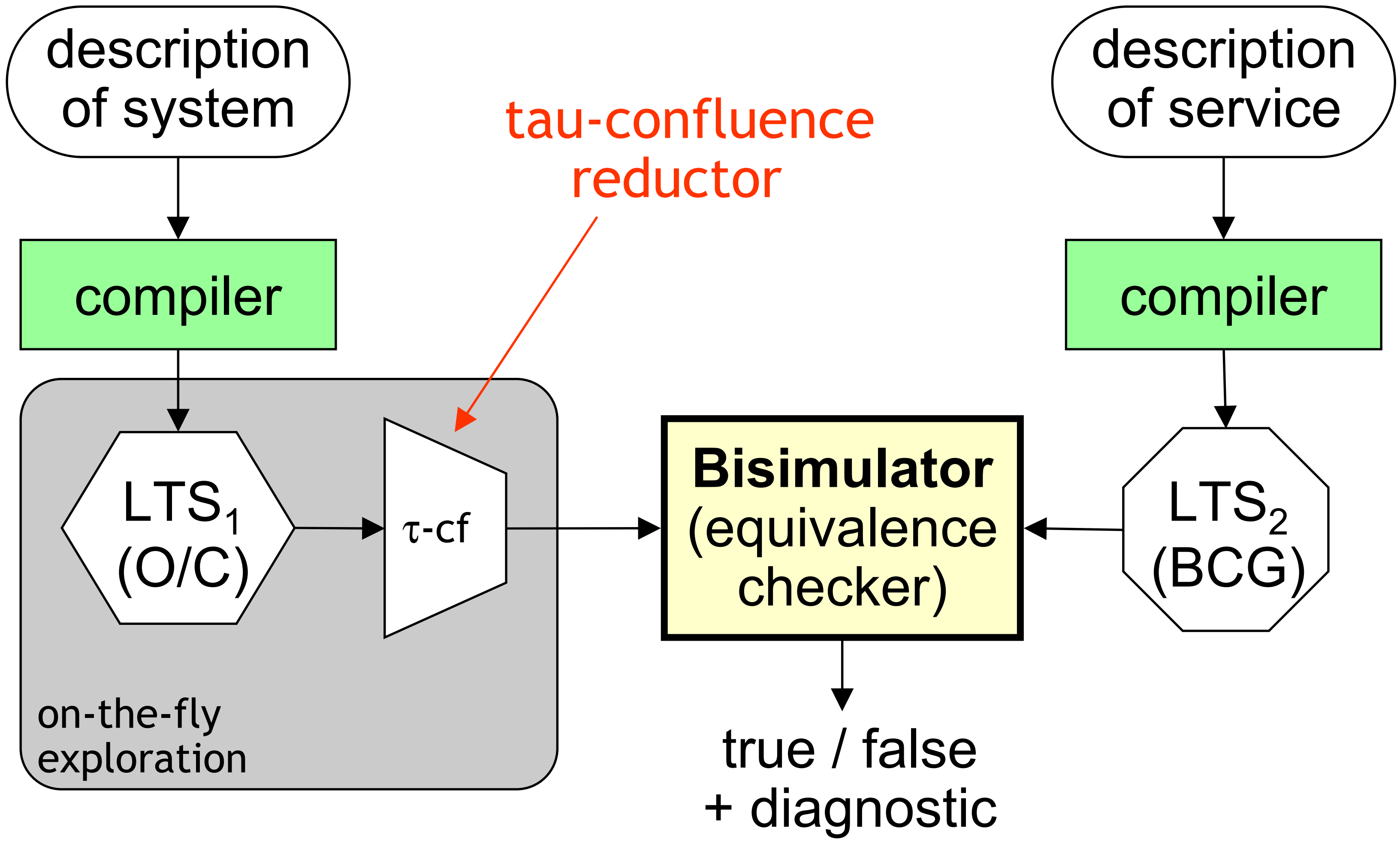
Evaluator/τ-compression vs. Evaluator
(time - property P2)



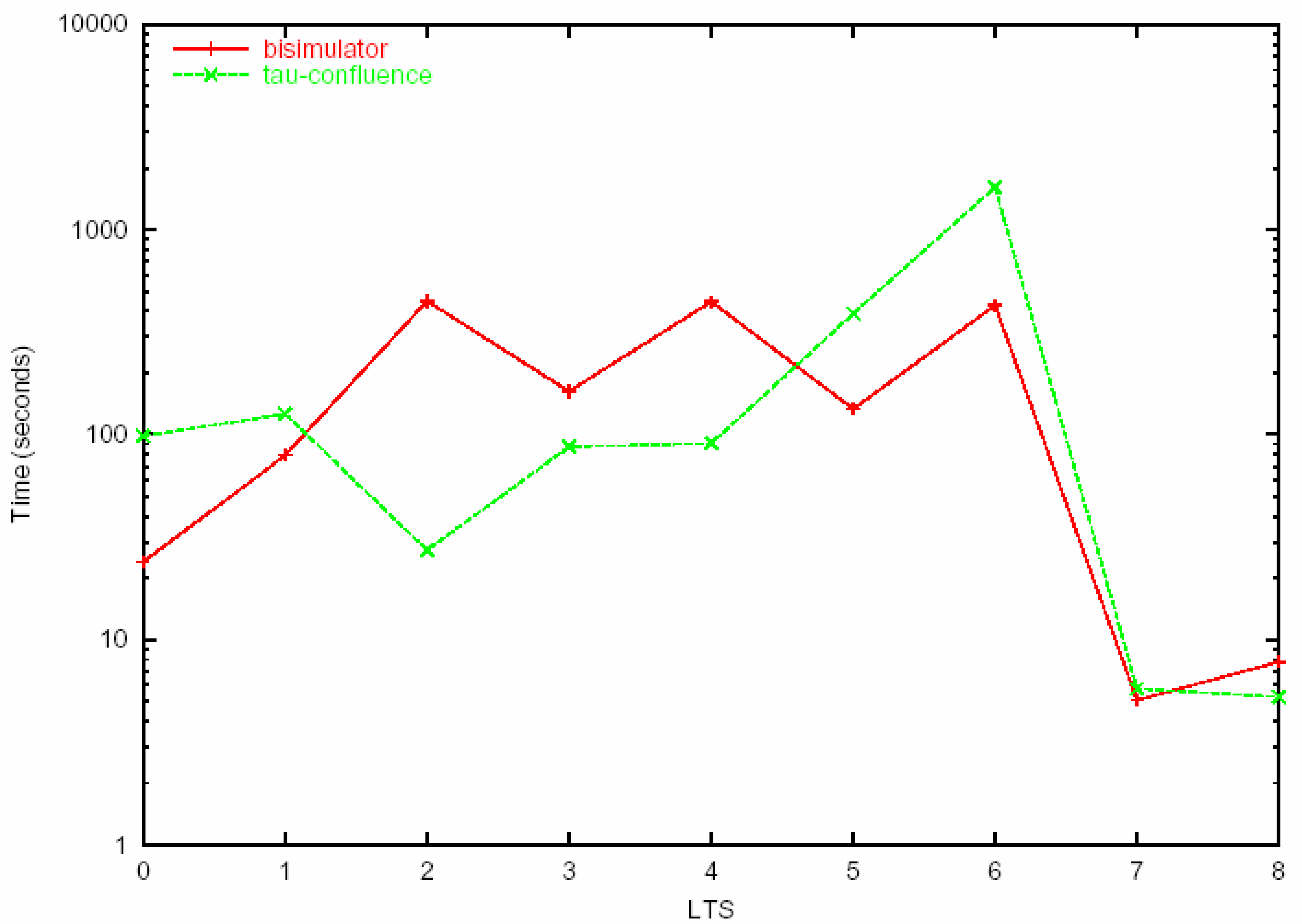
Evaluator/τ-compression vs. Evaluator
(memory - property P2)



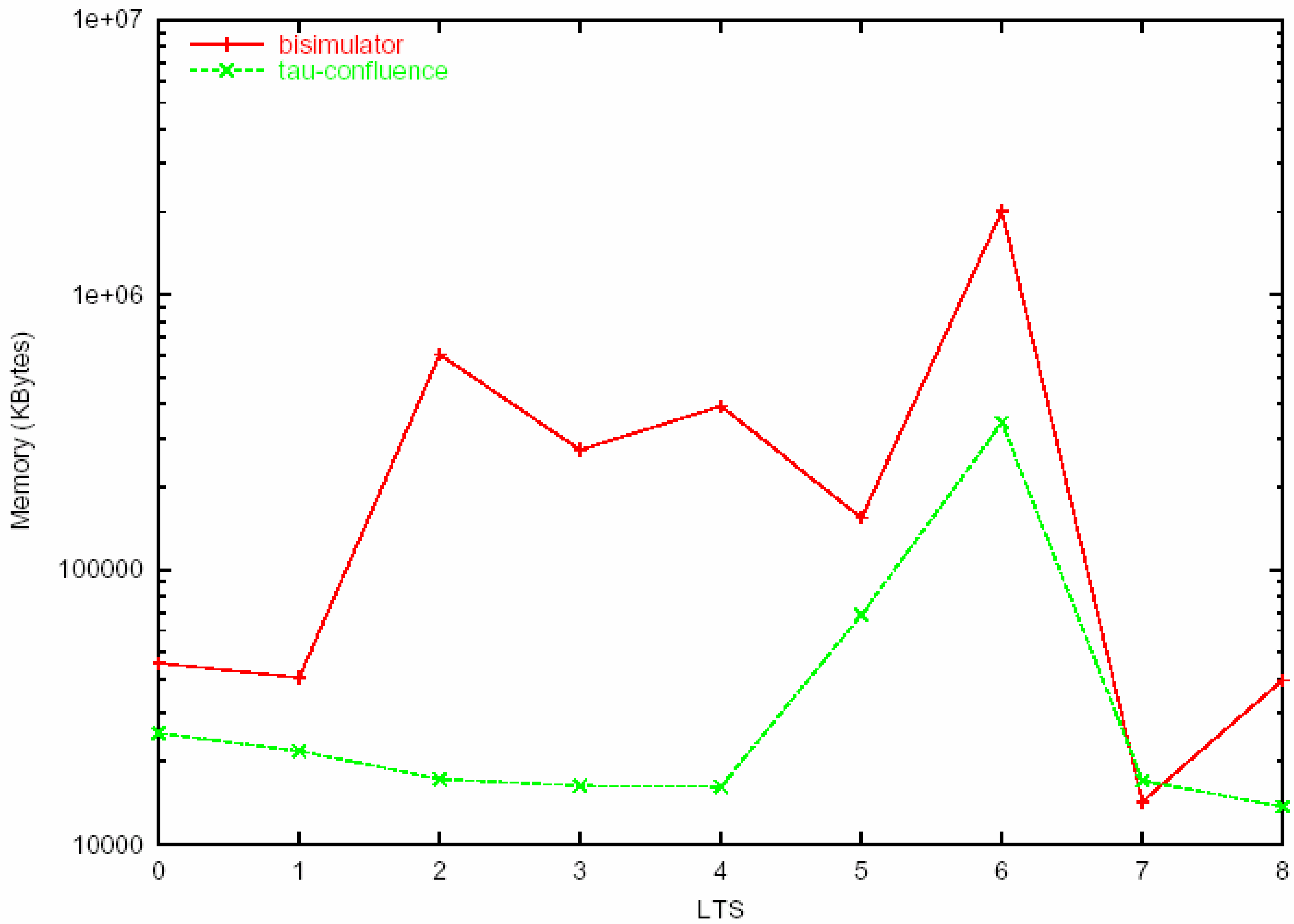
Equivalence checking



Bisimulator/τ-confluence vs. Bisimulator
(time - observational equivalence)



Bisimulator/τ-confluence vs. Bisimulator
(memory – observational equivalence)



Conclusion and future work

Already done:

- Three reductor modules (8,300 lines of C code)
- Language- and application-independent (Open/Caesar)
- Currently under integration within the O/C library

Ongoing:

- Continue experiments (VLTS benchmark suite)
- Apply reductors to other tools (Exhibitor, OCIS, …)
- Study other reductions (weak τ-confluence, τ-inertness, …)