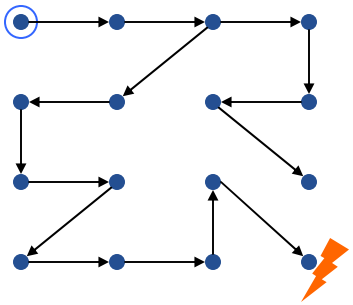
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

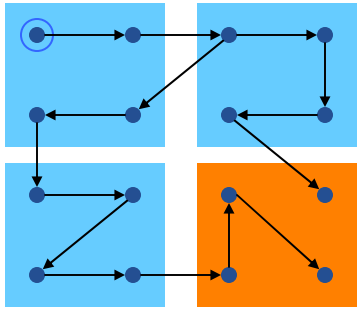
A Complete Bounded Model Checking Algorithm for Pushdown Systems

Gérard Basler
Daniel Kroening
Georg Weissenbacher

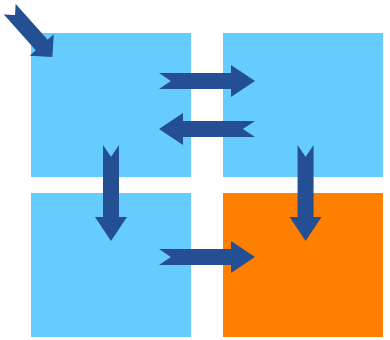
Abstract-Verify-Refine Paradigm (CEGAR)



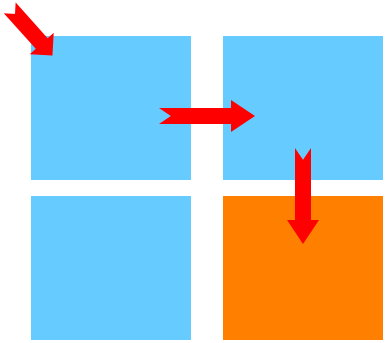
concrete transition system



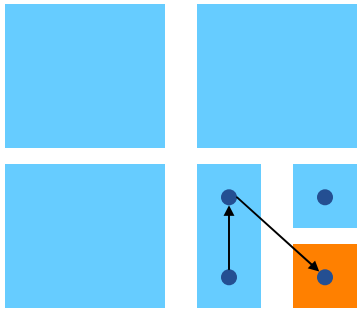
abstract states



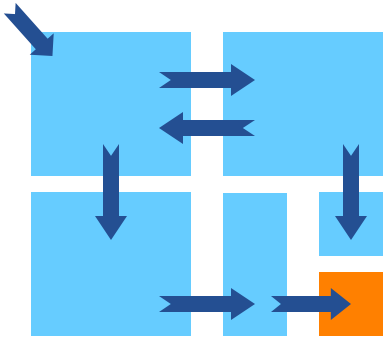
abstract transitions



spurious counterexample



refinement



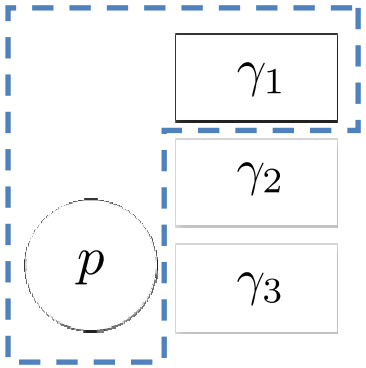
check new abstraction

Predicate Abstraction

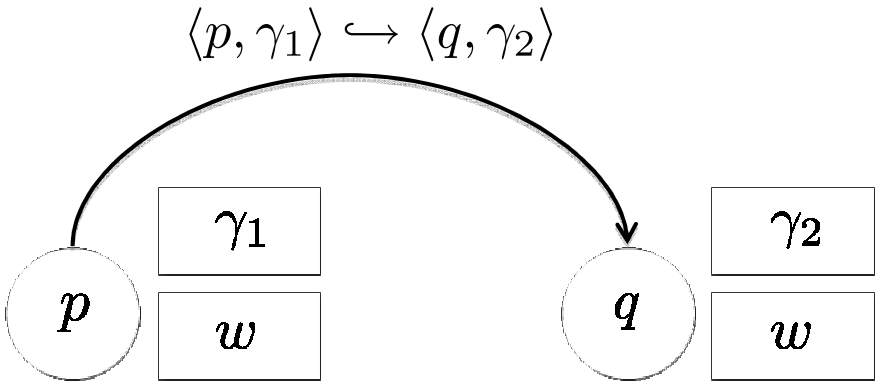
- Preserves control flow structure
- Tracks facts in the program using *predicates*
- Generates *pushdown systems*

What can a Pushdown System do?

- Finite number of variables, all of them Boolean
- Global state and stack
- (Recursive) function calls
- Only *head* visible (global variables + top of stack)



Transitions



- Assignment / Assumption: Modify head
- Call: Modify head and push new element on stack
- Return: Modify global variables and pop topmost stack element

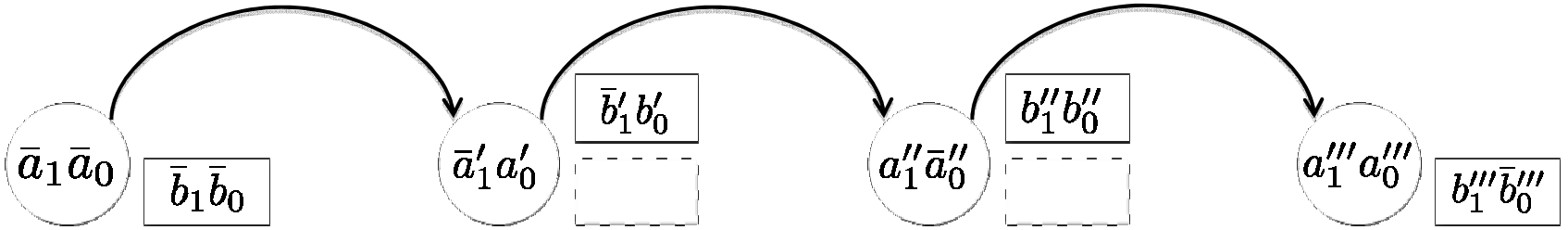
Model Checking Pushdown Systems

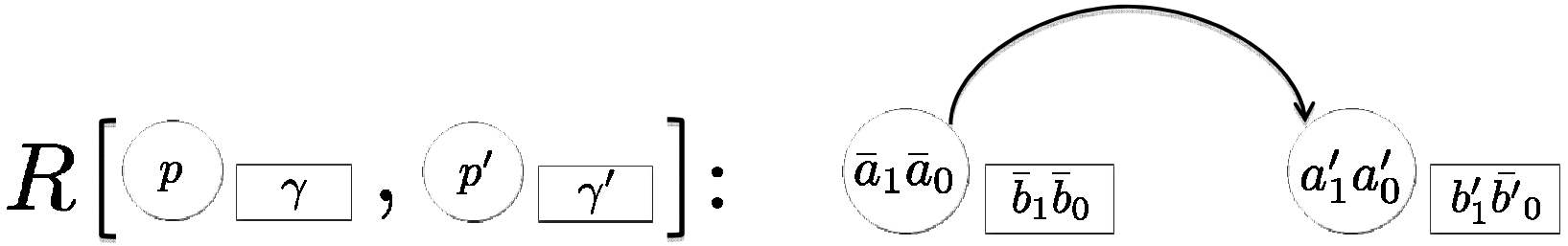
- *Reachability* of locations in *sequential* pushdown systems (PDS) is decidable
  - Symbolic model checkers based on BDDs: Bebop, Moped
  - BDD-based techniques don‘t scale for large number of variables
- Observations:
  - Location reachable in few steps

  ➔ Bounded Model Checking
  - If error location unreachable in original program: Location reachable in PDS in every iteration but the last

A transition sequence

- Symbolic transition sequence
- Relates first and last state of a path





Symbolic summarization

- Key idea to check reachability in pushdown systems (Bebop, Moped)
- Only finite number of possible input / output pairs
- Fixed-point check for SAT/QBF-based summarization:

$$R_{NEW} \subseteq R_{OLD}?$$

$$\forall\langle p_0, \gamma_0\rangle, \langle p'_0, \gamma'_0\rangle.\ \exists\langle p_1, \gamma_1\rangle, \langle p'_1, \gamma'_1\rangle.$$

$$R_{NEW}\left[\begin{array}{c} p_0 \\ \gamma_0 \end{array}, \begin{array}{c} p'_0 \\ \gamma'_0 \end{array}\right] = R_{OLD}\left[\begin{array}{c} p_1 \\ \gamma_1 \end{array}, \begin{array}{c} p'_1 \\ \gamma'_1 \end{array}\right]$$

Universal Summaries

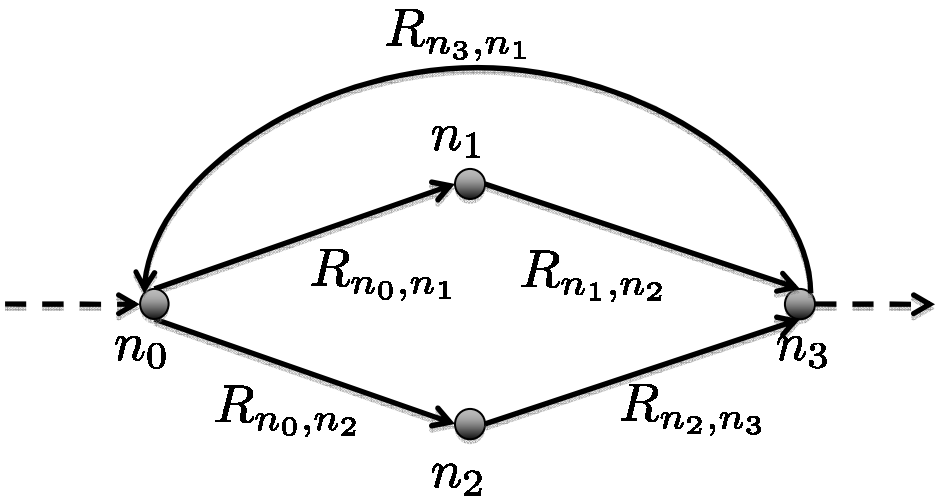
- *Universal summary* provides a summary for *any arbitrary* entry state
- "calling context" unconstrained

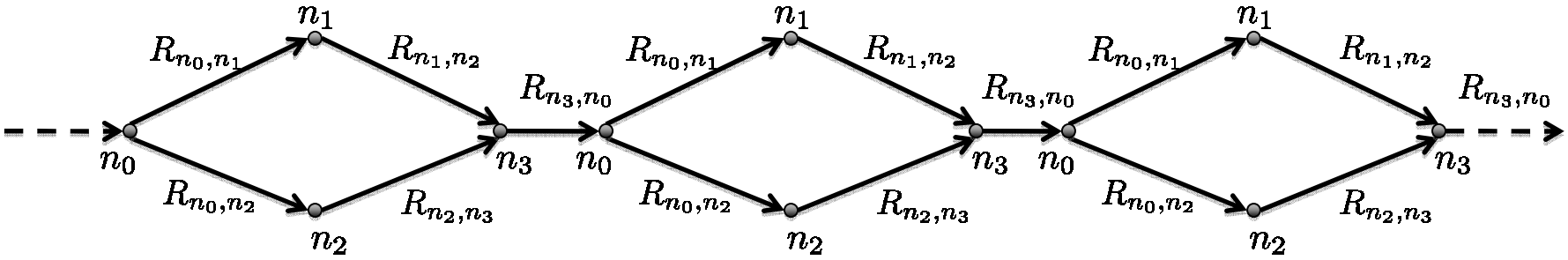
$$\Sigma_{\mathcal{U}}(\langle p, \gamma \rangle, \langle p', \gamma' \rangle)$$

$$\Longleftrightarrow$$

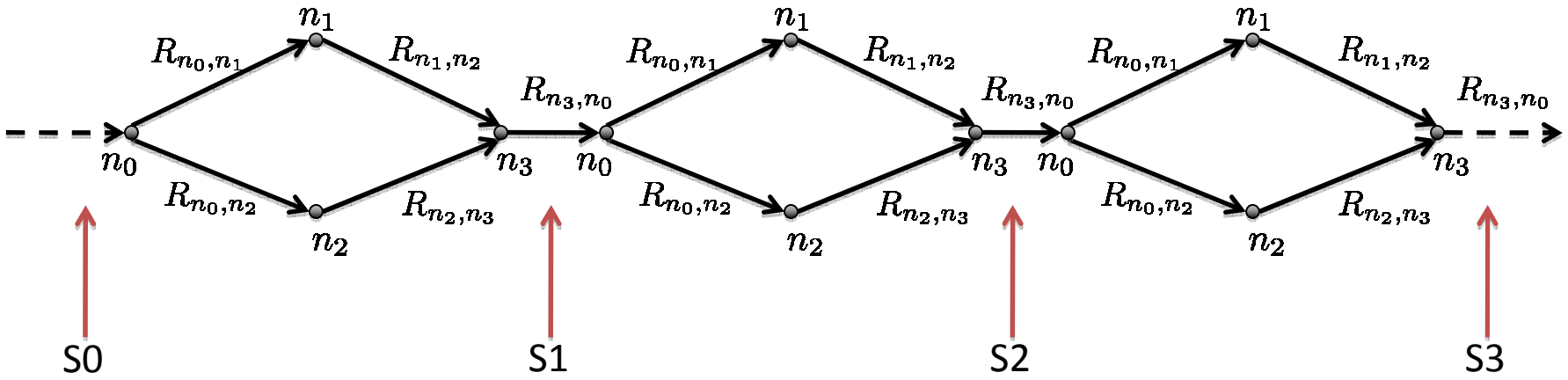
$$\exists \langle p_1, w_1 \rangle, \ldots, \langle p_n, w_n \rangle.$$
$$\langle p, \gamma \rangle \rightarrow \langle p_1, w_1 \rangle \rightarrow \ldots \rightarrow \langle p_n, w_n \rangle \rightarrow \langle p', \gamma' \rangle \wedge$$
$$\forall i \in \{1..n\}.|w_i| \geq 2$$

Constructing Universal Summaries using BMC





BMC: when to stop unrolling?

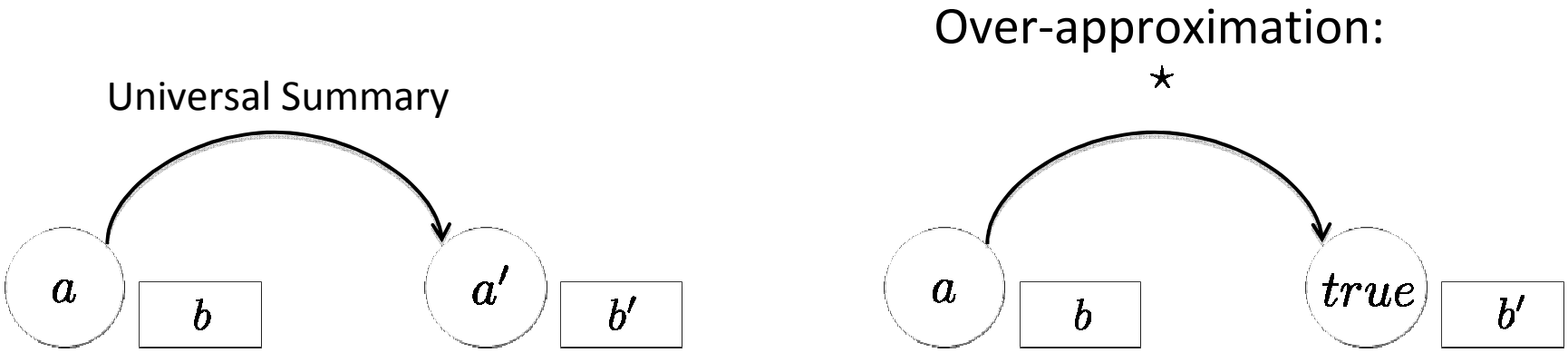


- Unroll up to *longest path* that doesn't visit any state twice
- All states in the path are *pairwise* different
- $\exists S0, S1, S2, S3 \;.\; S0 \neq S1 \wedge S0 \neq S2 \wedge S0 \neq S3 \wedge S1 \neq S2 \ldots$

Eager approach

- *Eager* application of universal summaries leads to large formulas
- Worst case: exponential number of unrollings
- Predicate abstraction:
  - If property holds: Location reachable in every iteration but the last. SLAM: up to 20 iterations until location unreachable

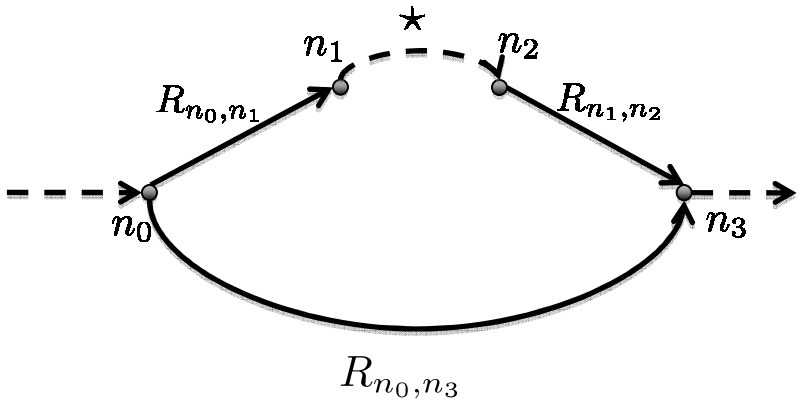
Over-Approximations for summaries

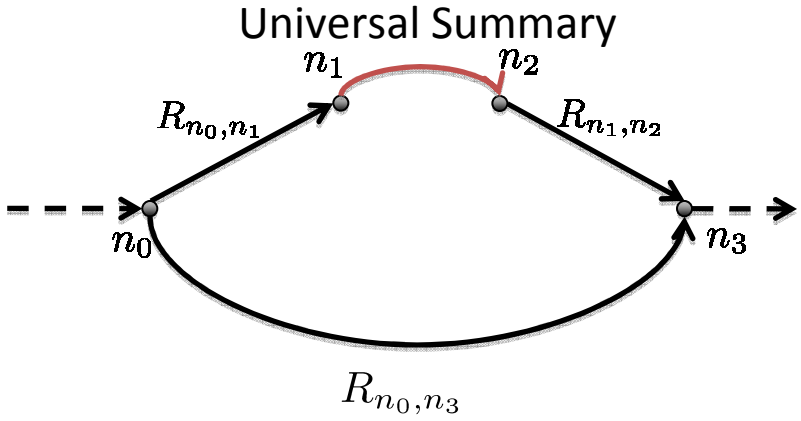


- If location unreachable in over-approximation, then location is unreachable in original PDS

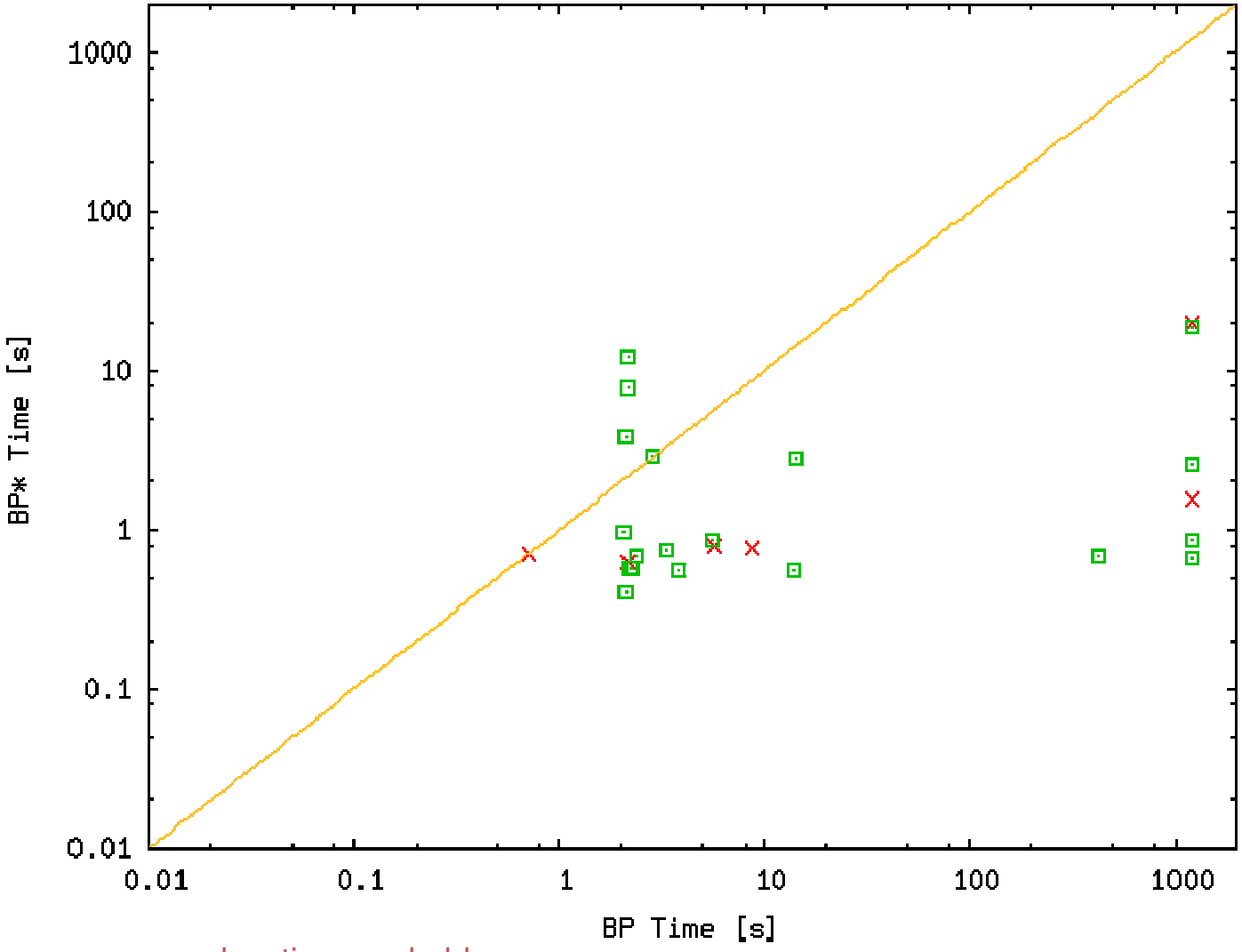
Abstraction and Refinement with Summaries

- *Spurious* paths
  - Refine transition system until feasible path found or head unreachable
  - Fall back to QBF algorithm, if computation of universal summary not possible



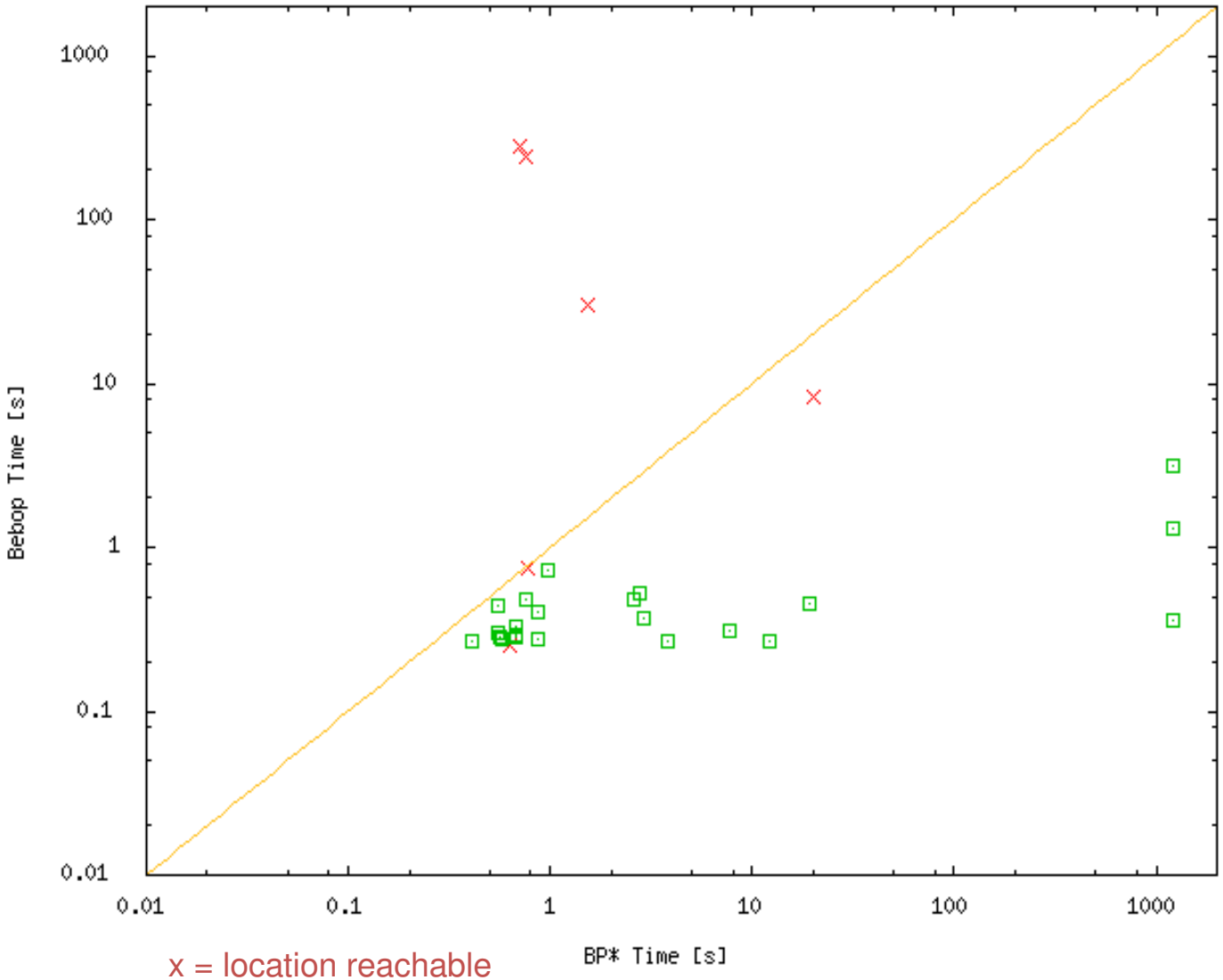


Benchmarks



x = location reachable

Benchmarks



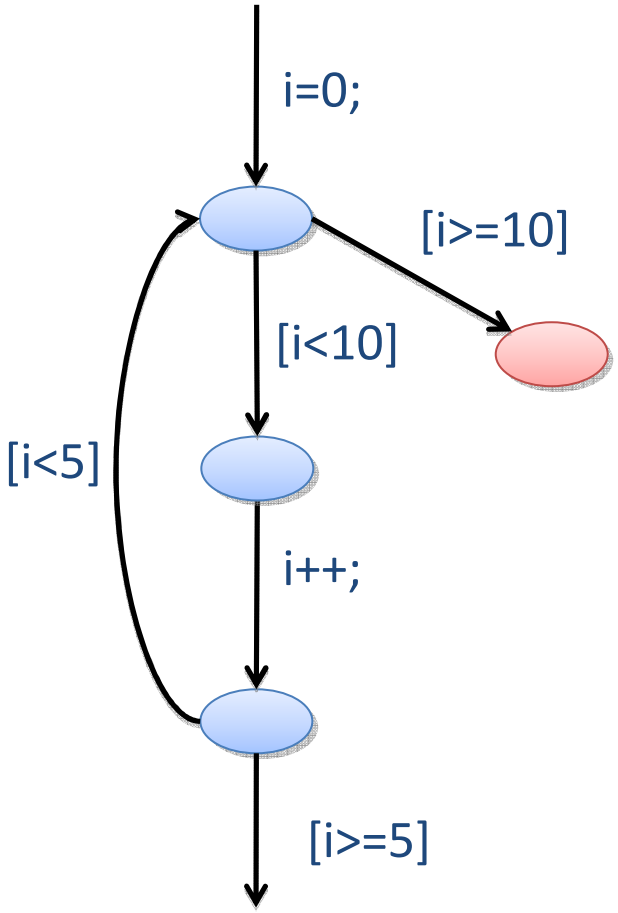
x = location reachable

Conclusion

- BMC based method superior to BDD-based model checking if location reachable
- BDD-based and bounded model checker can be run in parallel
- Improve heuristics for constructing universal summaries and refinement

Backup slides

PredicateAbstraction

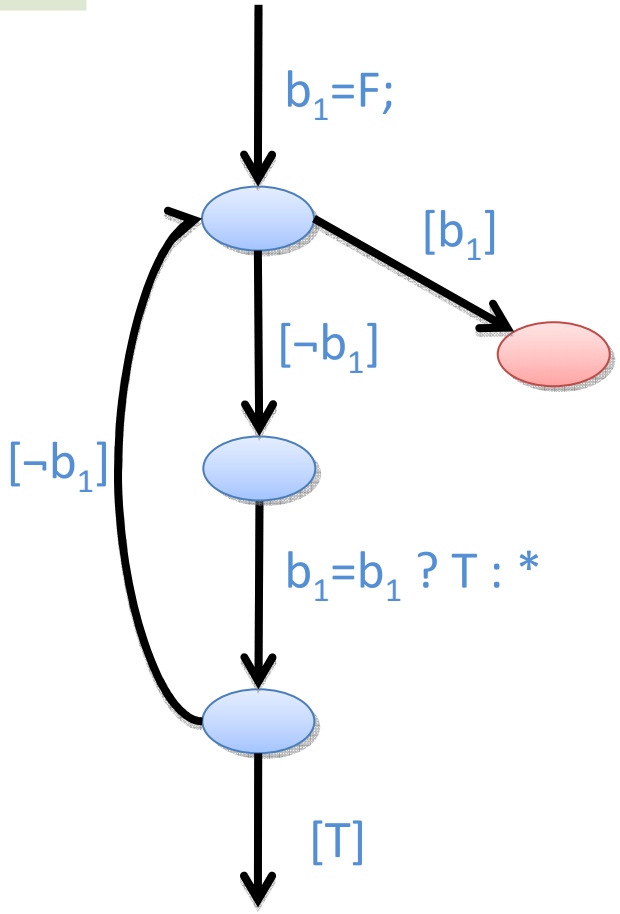



```
int i=0;
do {

assert(i < 10);

   i++;

}while(i < 5);
```

PredicateAbstraction

b_1

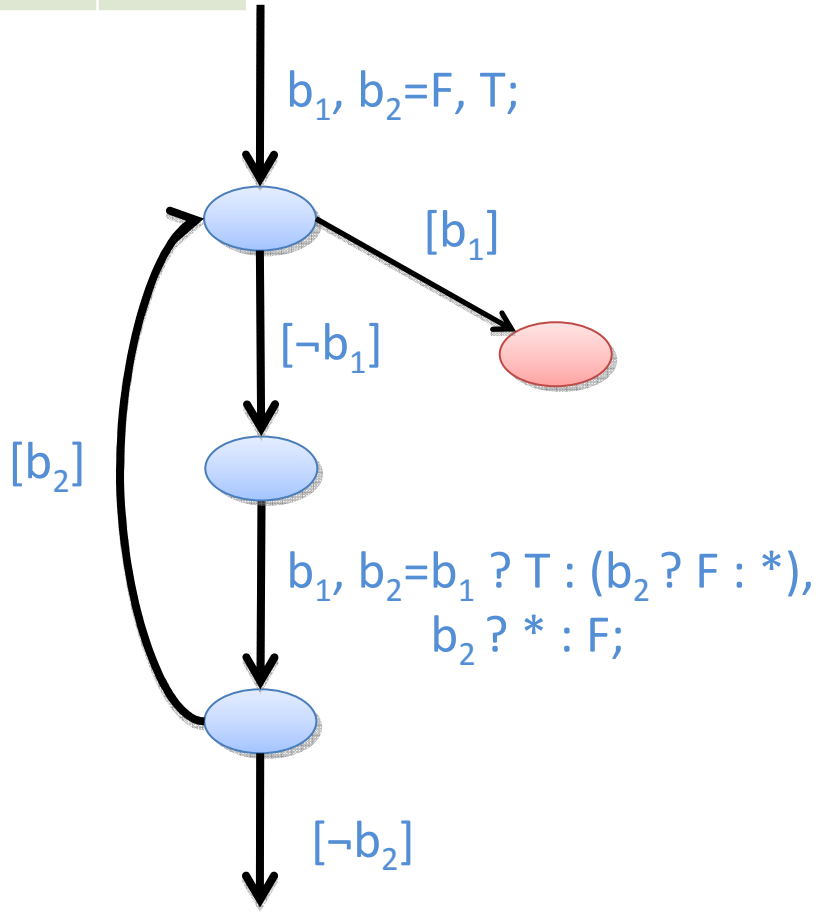
i > 10




```
int i=0;
do {

assert(i < 10);

   i++;

}while(i < 5);
```

PredicateAbstraction

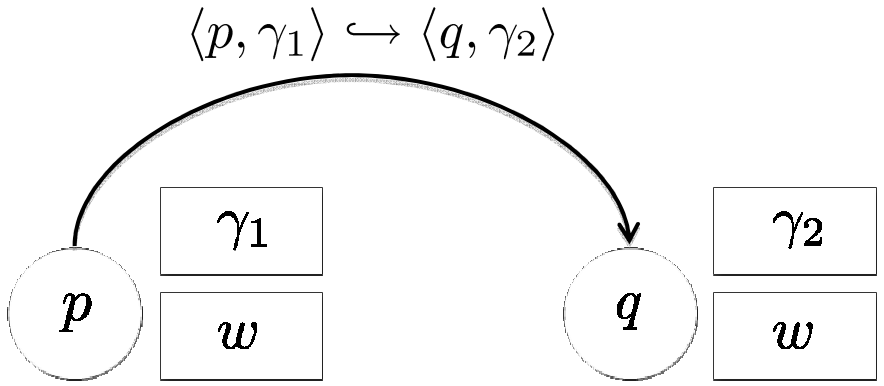
b_1	b_2
i > 10	i < 5




```
int i=0;
do {

assert(i < 10);

   i++;

}while(i < 5);
```

Transitions: *Neutrations*

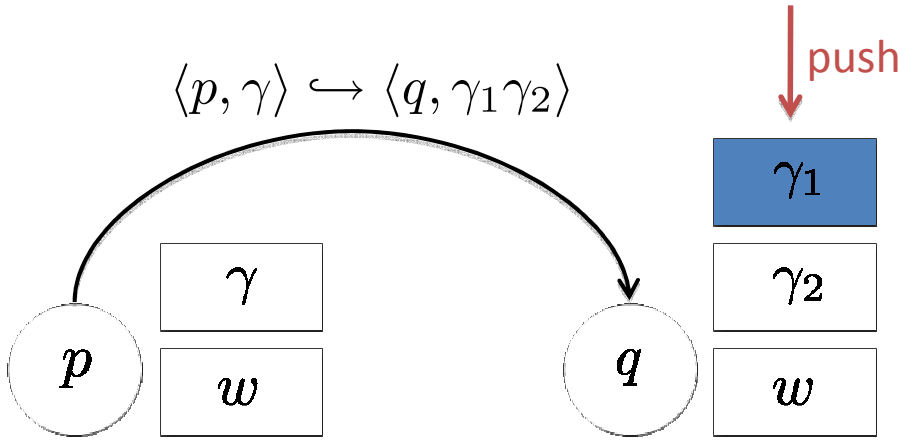
:=
assume



- Modify the control state p
- Modify the topmost stack element γ_1
- Do not modify the elements below γ_1

Transitions: *Expansions*

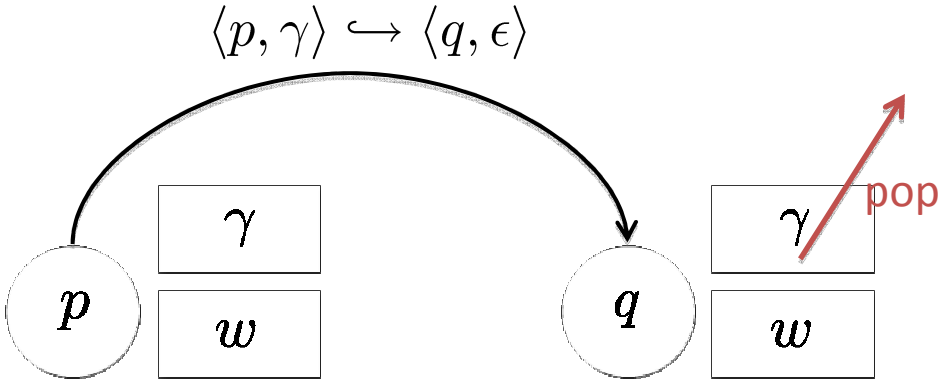
call



- Modify the control state p
- Modify the topmost stack element γ
- Push a new element on the stack

Transitions: *Contractions*

return



- Modify the control state p
- Pop the topmost stack element γ

Model Checking Boolean Programs

- *Reachability* of locations in Boolean Programs is decidable
  - BDD based symbolic model checkers Bebop, Moped
- So why bother to work on a „solved“ problem?
  - SatAbs: >70% of runtime spent verifying Boolean Programs
  - BDD-based techniques don‘t scale for large number of variables
- But is there something faster than BDDs?
  - SAT-solvers can solver instances with a huge number of variables
  - QBF-solvers are improving steadily