

# Directed Search for the Verification of Communication Protocols

*6. Juni 2003*

Alberto Lluch Lafuente

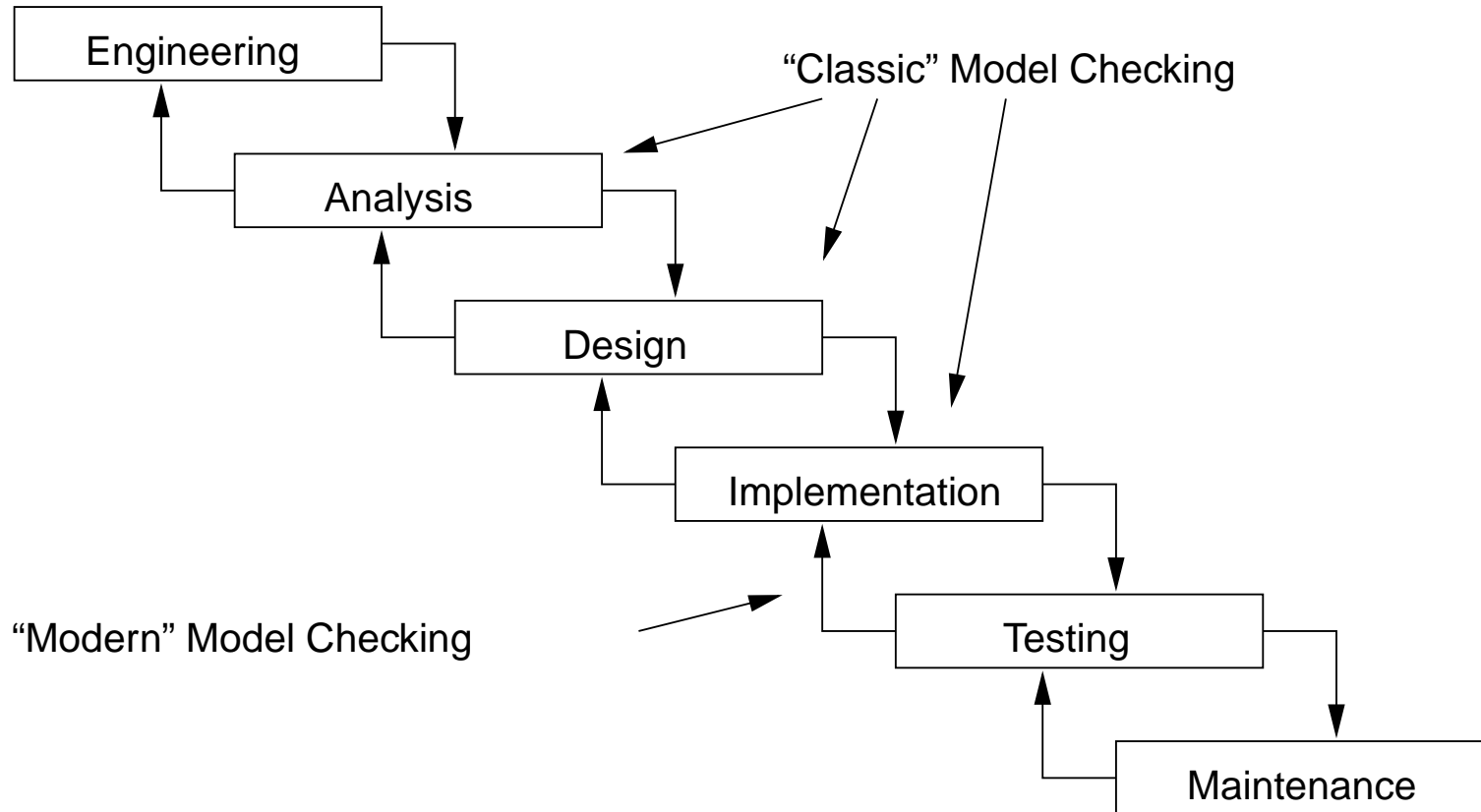
lafuente@informatik.uni-freiburg.de

Institut für Informatik

Albert-Ludwigs-Universität Freiburg

Germany

# System Development

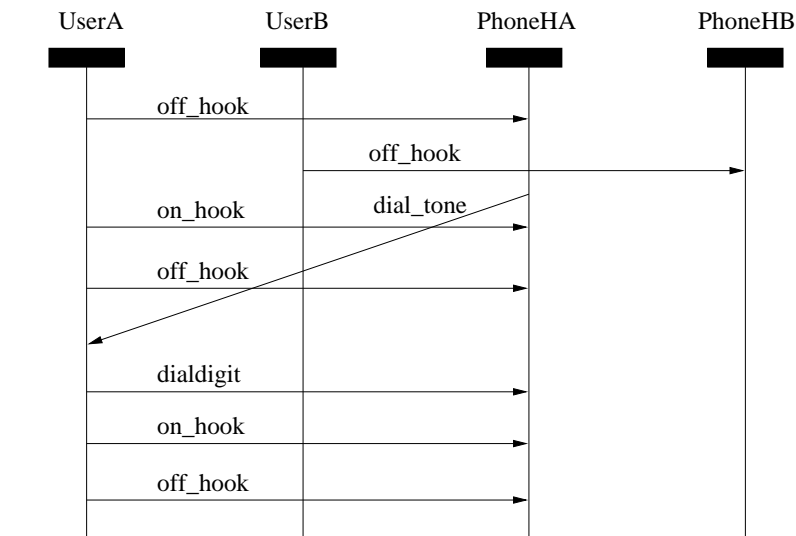


Verification methods: Testing, simulation, deductive reasoning and Model Checking.

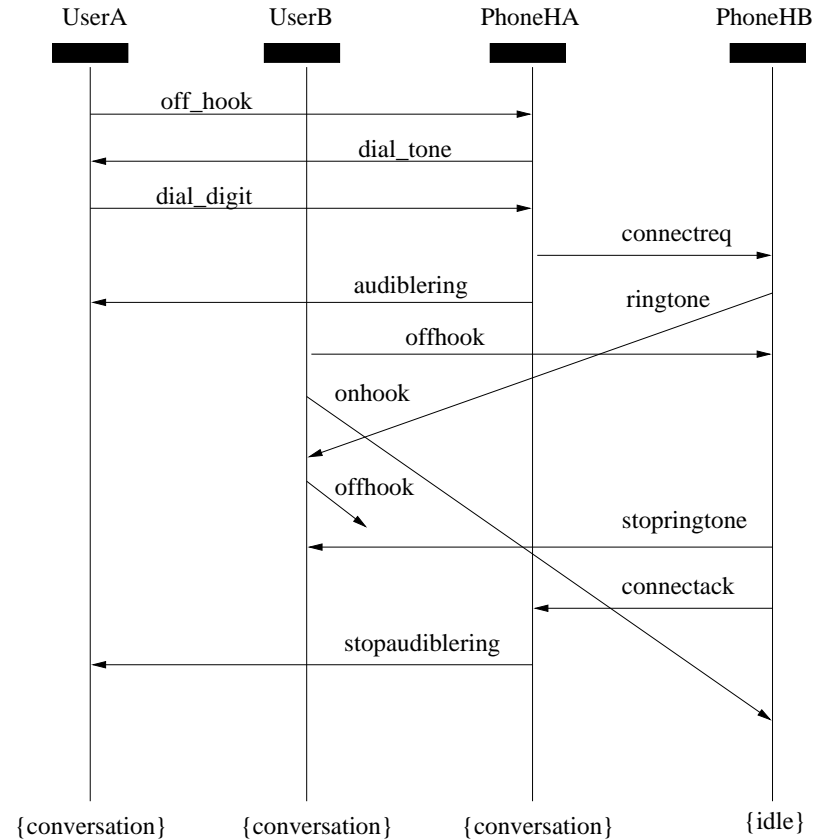
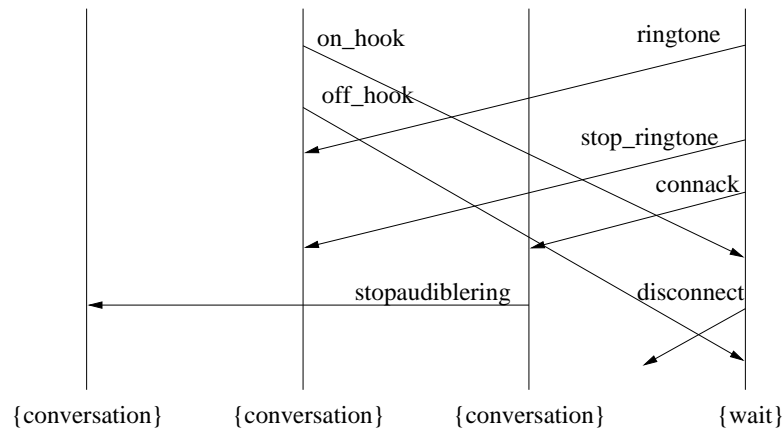
# System Verification

- Model Checking: Given a system model ( $M$ ) and a specification ( $\phi$ ), decide automatically whether or not  $M$  satisfies  $\phi$  ( $M \models \phi$ ).
- A negative answer is a counterexample (error trace or error trail) explaining how the error occurs.
- Model checking exhaustively analyzes the state space of the system ...
- ... which can be large enough to impede the task: state explosion problem.
- In practice, model checking is more effective as debugging tool.

# Counterexample Length



... 440 further messages ...



# Goals

- Directed, guided, heuristic search algorithms to:
  - Accelerate the search to errors.
  - Obtain minimal counterexamples.
- Algorithms?
- Heuristics?
- Liveness?
- Combination with reduction techniques
  - Partial order reduction
  - Bitstate hashing compression

# Modeling Asynchronous Systems

- A model is an abstraction of the system design or implementation, capturing the important details.
- Asynchronous system  $M$  = asynchronous composition of various communicating processes  $P_1 \dots P_n$ .
- Communicating process: finite automata with variables.
- A finite transition system is a tuple  $M = \langle S, S_0, T \rangle$ 
  - A state of  $S$  is a valuation of the variables and the states of each  $P_i$ .
  - A transition of  $T$  is a transition in one  $P_i$ .

# Correctness Specification

- Linear-Time Temporal Logic (LTL): extension of propositional logic with temporal operators **F** (in the **F**uture), **G** (**G**lobally), **X** (in the ne**X**t state).
- Safety properties express that, under certain conditions, a *bad* event never happens.
- Liveness properties express that, under certain conditions, a *good* event will ultimately occur.
- For our purpose:
  - safety errors = finite system executions, represented as path to an error state.
  - liveness errors = infinite system executions, represented as cycles.

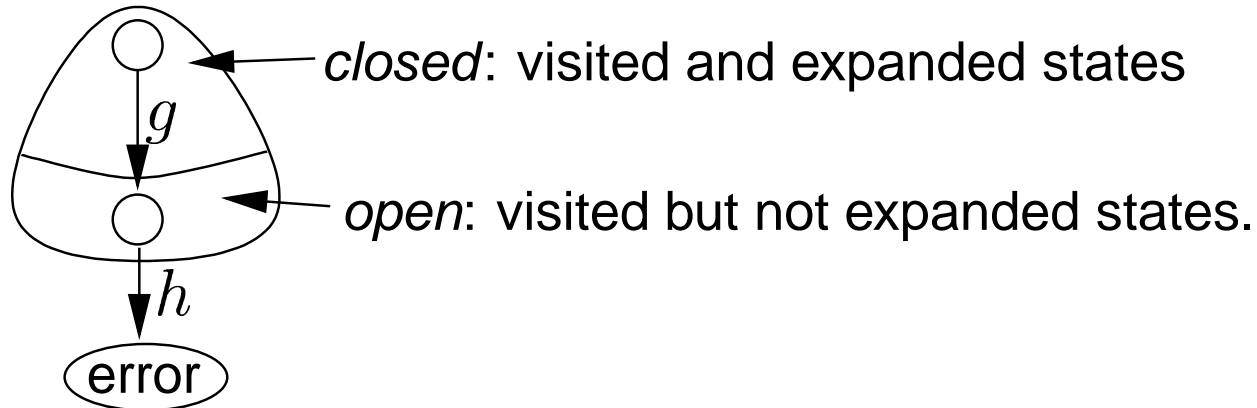
# Automata-Based Verification

- Model as automata  $M$ , Error Specification ( $\neg\phi$ ) as automata  $N$ .
- $M \models \phi \Leftrightarrow L(M \cap N) = \emptyset$
- Liveness verification: interpret  $M, N$  and  $M \cap N$  as Büchi automata and check if  $M \cap N$  accepts no run.
- Safety verification: interpret  $M, N$  and  $M \cap N$  as automata over finite runs ending in an accepting state.
- On-the-fly explicit-state verification.

# Directed Safety Verification

- Finding safety errors is reduced to reachability analysis.
- Explicit-state model checkers use DFS due to its memory-efficiency.
- Main drawback: blind strategy, non-minimal counterexamples.
- Alternative: use AI heuristic search algorithms, like A\* and best-first.
- Heuristics are used to guide the exploration of the state space to ...
  - Explore that part more likely to contain errors.
  - Find (near-to) minimal counterexamples.

# A\* and Best-First



- Iteratively expand states from *open*.
- Best-First: Extract states according to heuristic  $h$ .
- A\*: Extract states according  $h + g$ .
  - Optimal counterexamples if  $h$  is lower bound.

# Heuristics

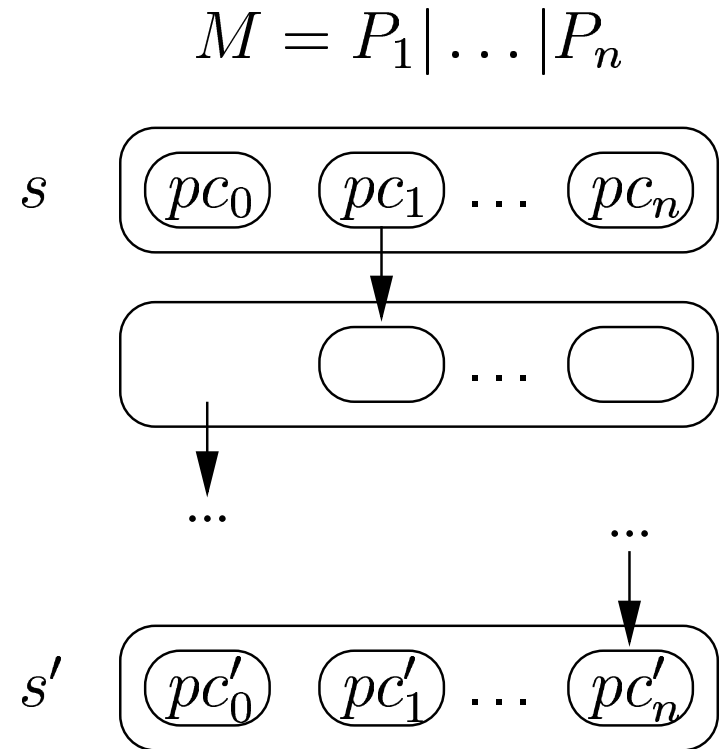
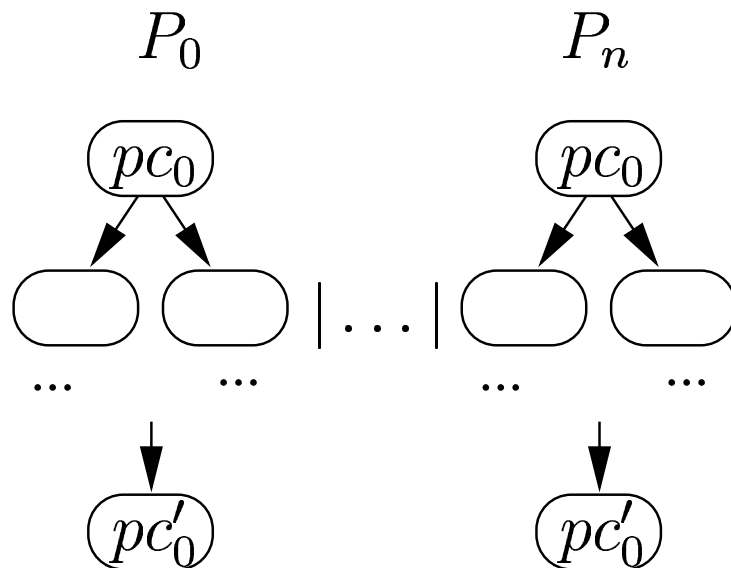
- Evaluation heuristics:
  - *Desirability* of expanding a state.
  - Accelerate the search (BF).
- Estimates heuristics:
  - Approximate the distance to error states.
  - Find minimal paths ( $A^*$ ).
- Two cases:
  - No error state has been found yet.
  - A counterexample is available.

# Formula-based Heuristic

- Safety errors characterized by propositional formula  $f$ .
- $h_f(s)$  estimates the number of transition necessary to reach from  $s$  a state  $s'$  where  $f$  holds:
  - $h_{\text{true}}(s) = 0$ ,  $h_{\text{false}}(s) = \infty$ ,  $h_{\neg g}(s) = \bar{h}_g(s)$
  - $h_{x \otimes y}(s) = 0$  (if  $x \otimes y$ ),  $= |x - y|$  (otherwise)
  - $h_v(s) = 0$  (if  $v$  is *true*),  $= 1$  (otherwise)
  - $h_{\text{full}_{(q)}}(s) = \text{capacity}(q) - \text{length}(q)$
  - $h_{\text{empty}_{(q)}}(s) = \text{length}(q)$
  - $h_{g \vee g'}(s) = \min\{h_g(s), h_{g'}(s)\}$
  - $h_{g \wedge g'}(s) = h_g(s) + h_{g'}(s)$
- Main drawback:  $f$  is very simple in practice.

# FSM Distance (1)

- From system state  $s = (pc_0, \dots, pc_n, v_0, \dots)$  to system state  $s' = (pc'_0, \dots, pc'_n, v'_0, \dots)$  each  $P_i$  must progress from  $pc_i$  to  $pc'_i$ .



# FSM Distance (2)

- The minimal number of system transitions from  $s$  to  $s'$  is less or equal to the sum of the minimal distances from  $pc_i$  to  $pc'_i$  in each  $P_i$ .
- Hence, FSM Distance is a lower bound to the distance to  $s'$ .
- $A^*$  is able to deliver the minimal path to  $s'$ .
- FSM distance is computed in  $O(n)$ .
  - Pre-computing the distances in  $P_i$  in  $O(|P_i|^3)$ .
  - In practice:  $|P_i| \ll |M|$ , since  $|M|$  is  $O(|P_1| \cdot \dots \cdot |P_n|)$ .
- Alternative: Hamming Distance.

# Results on Safety Error Detection

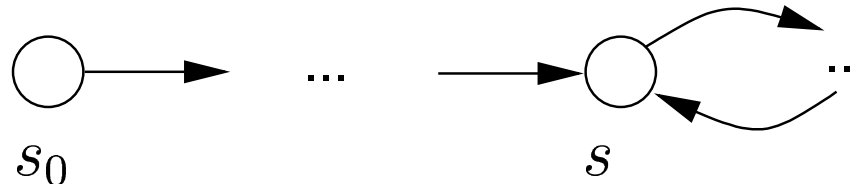
- Finding minimal counterexamples when no error state is given is not easy.
- 1st search for error state  $s'$ :
  - Best-first vs DFS ?
  - Heuristic goodness depends on system/specification.
- 2nd find minimal counterexample:
  - A\* more efficient than Dijkstra's, Iterative DFS.
  - Minimal paths to  $s'$ , near-to-minimal counterexamples.
  - Heuristics: FSM, Hamming Distance.

# Liveness Verification

- Liveness Verification: given a state transition graph, find a cycle containing at least an accepting state.
- Solution: nested depth-first search algorithm.
- Observation: structure of  $M \cap N$  depends on  $N$ .
- A partition function can be defined such that:
  - Cycles in  $M \cap N$  are localized.
  - Acceptance classification of partitions.
- Improved nested depth-first search:
  - Accepting cycles can be detected earlier.
  - Search can be bounded.
  - Parallelization becomes easier.

# Heuristic Search for Liveness

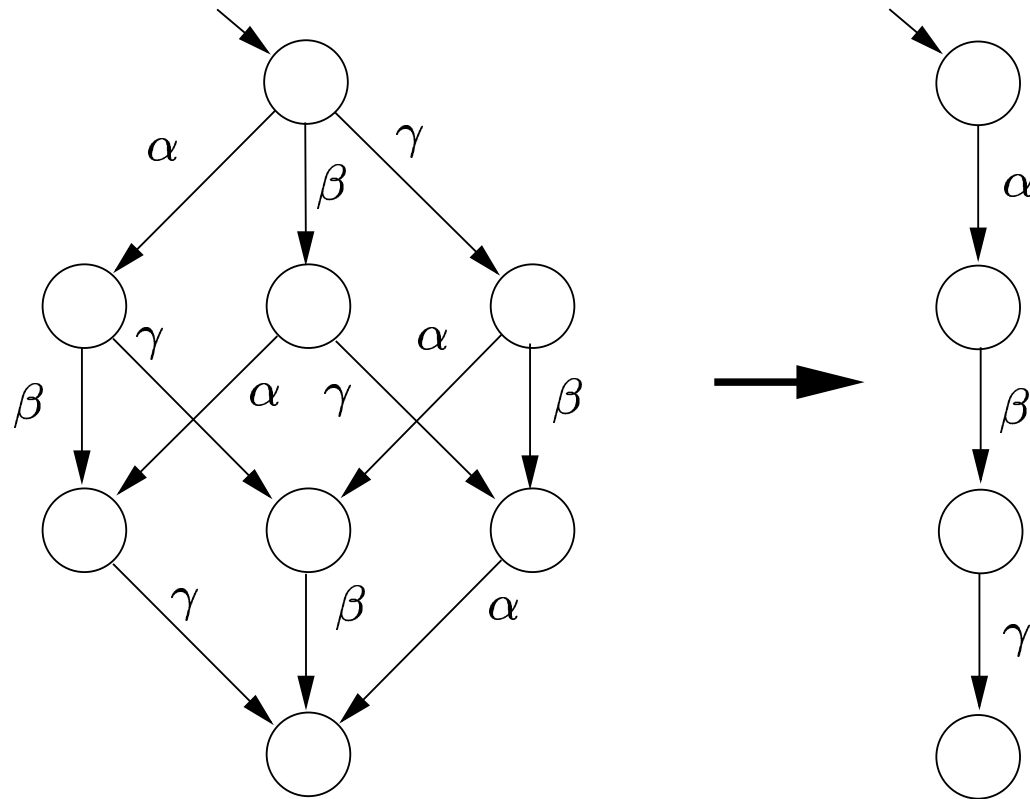
- Checking cycles is not easy with heuristic search, ...
- but one can apply it to improve liveness counterexamples.
- Liveness counterexamples = initial path to cycle seed  $s$  + cycle through  $s$ .



- Simple approach: find the shortest path to  $s$  from the initial state and the shortest cycle through  $s$ , e.g. by using  $A^*$  with FSM Distance.
- Then put both paths together.

# Partial Order Reduction

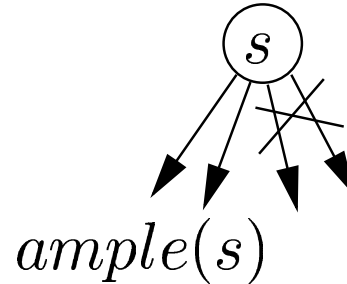
- Commutative transitions are typical in asynchronous systems and lead to *equivalent* executions.



- POR avoids the exploration of equivalent paths.

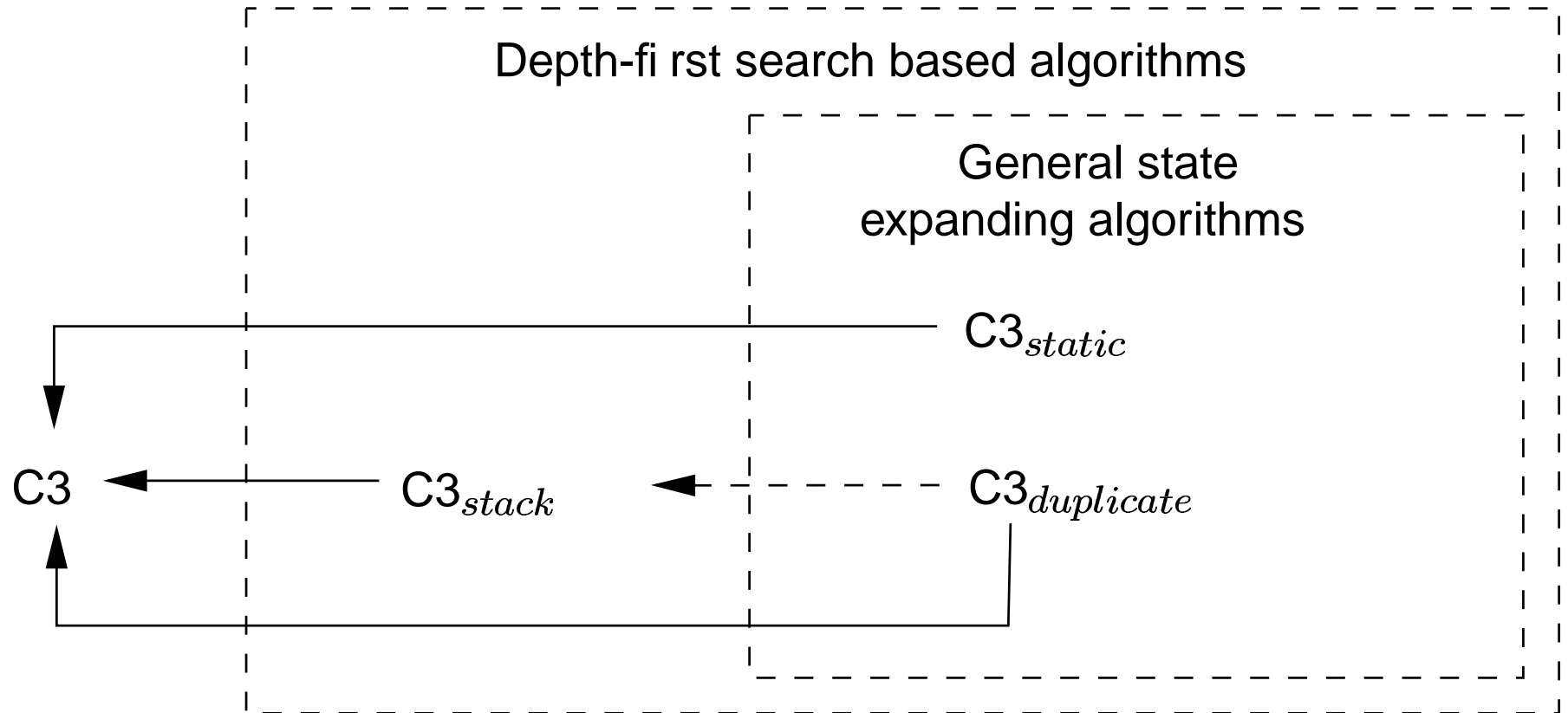
# Ample Set Method

- Use  $ample(s) \subseteq enabled(s)$  instead  $enabled(s)$ .



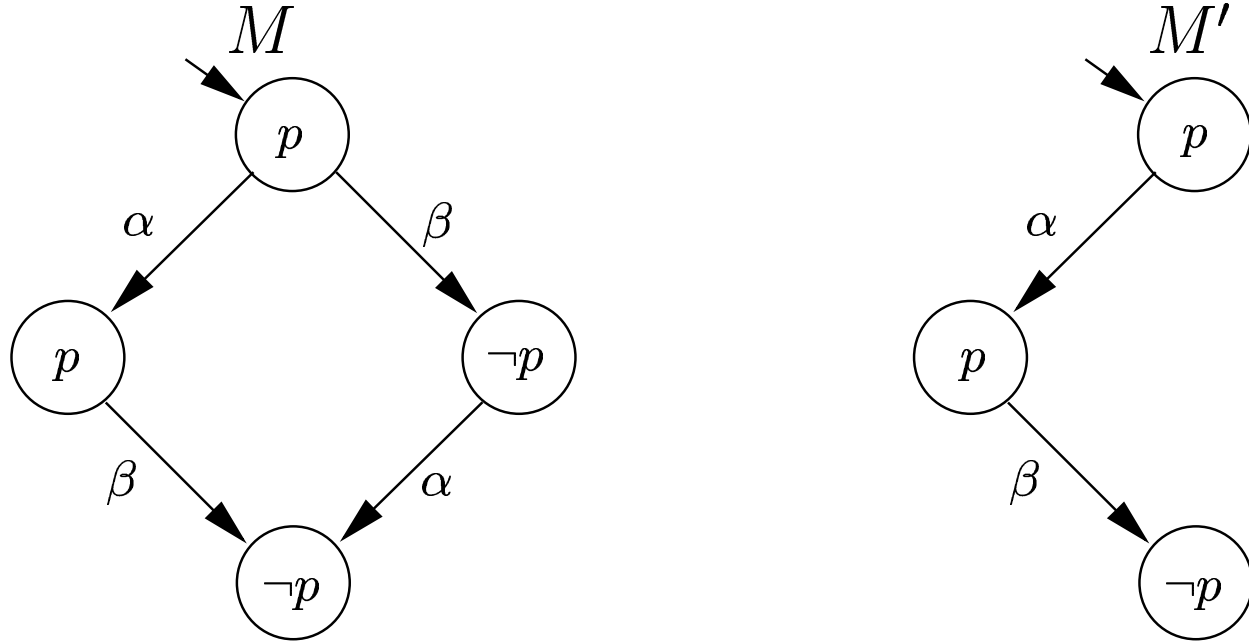
- Four necessary and sufficient conditions for  $ample$  ensure a correct reduction.
- Conditions **C0-2** are independent of the search algorithm.
- Condition **C3** entails cycle detection, hence dependent on the search algorithm.

# Hierarchy of C3 Conditions



# POR and Counterexample Length

- Shortest counterexample in the reduced state space may be longer.
- Example:  $\phi = \mathbf{G}p$ ,  $\alpha, \beta$  independent,  $M \sim_{st} M'$ .



- This problem can be mitigated by reordering the counterexample.

# Bitstate Hashing

- Partial compression method applied in protocol validation (SPIN).
- A state is compressed down to 1 or 2 bits.
- DFS stores search information in the search stack.
- Heuristic search ( $A^*$ ) requires to store with each state: path information and cost to reach the state.
- Bitstate hashing more efficient with stack-based algorithms (DFS, IDA\*) than with general search algorithms ( $A^*$ , BF).

# Conclusion

- Best-first search strategies for finding (safety) errors
- A\* for shortening already established (safety and liveness) counterexamples.
- Checking liveness can be improved by exploiting structural properties of the specification.
- Partial order reduction is compatible with A\*,BF, though less effective than with DFS, IDA\*.
- Bitstate hashing less efficient with heuristic search.
- Heuristic search as alternative to enhance the bug finding capabilities of model checkers: HSF-SPIN, JPF.
- Future Work: Symmetry reduction, Directed search for specific domains, Distributed Model Checking.