Model Checking Push-down Systems

Verification of Infinite State Systems

Samik Basu

September 23, 2002

Organization

- Programs with (recursive) procedure calls
  - ⋆ Control flow graphs
  - ⋆ Push-down Models of Programs
- Property
  - ⋆ Linear Temporal Logic
    - Büchi Automata
  - ⋆ Modal mu-calculus
- Model Checking by *abstraction*

Model Checking

Given a Model M of the program & a property φ specified in temporal logic, $M \stackrel{?}{\models} \varphi$

- Automata-theoretic Approach
  - ⋆ M and φ are represented as *finite* state automata
  - ⋆ Product automata construction
  - ⋆ Check for language emptyness

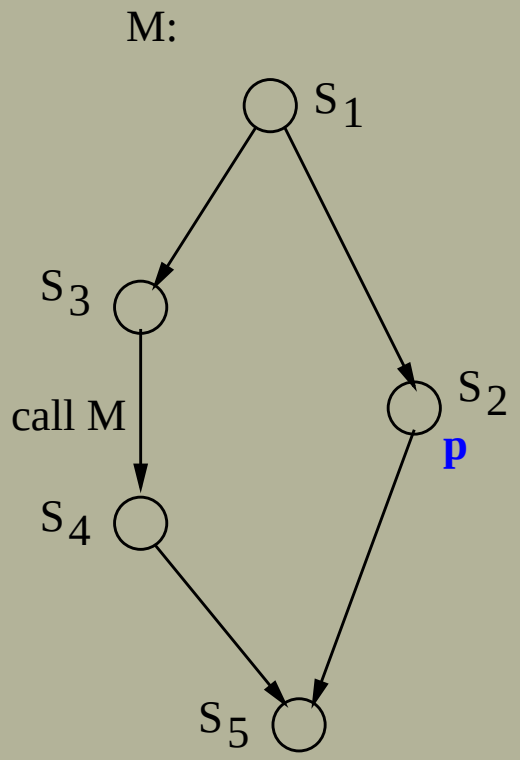
Program Model

Control-flow graph

Push-down Model

Program Model

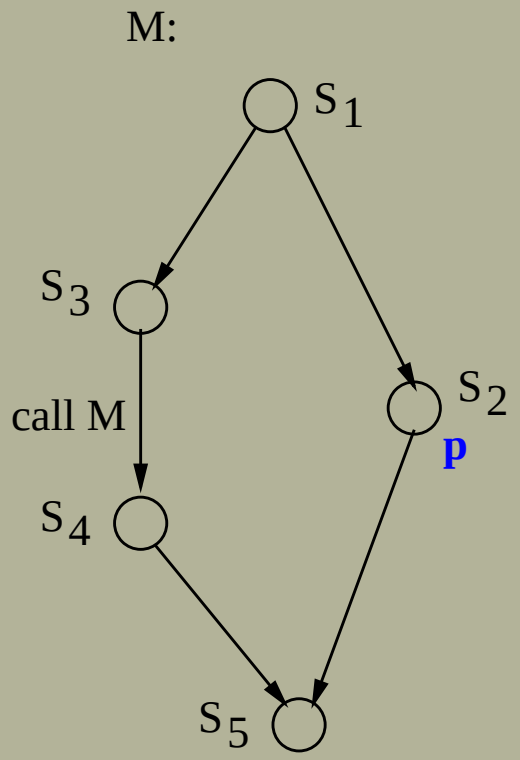
Control-flow graph



Push-down Model

Program Model

Control-flow graph

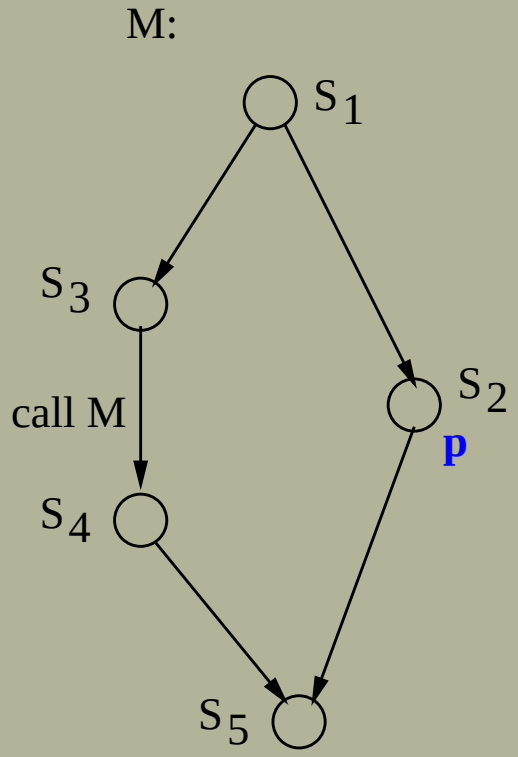


Push-down Model

$S = \{s_1, s_2, s_3, s_4, s_5\}$ – stack alphabets

Program Model

Control-flow graph



Push-down Model

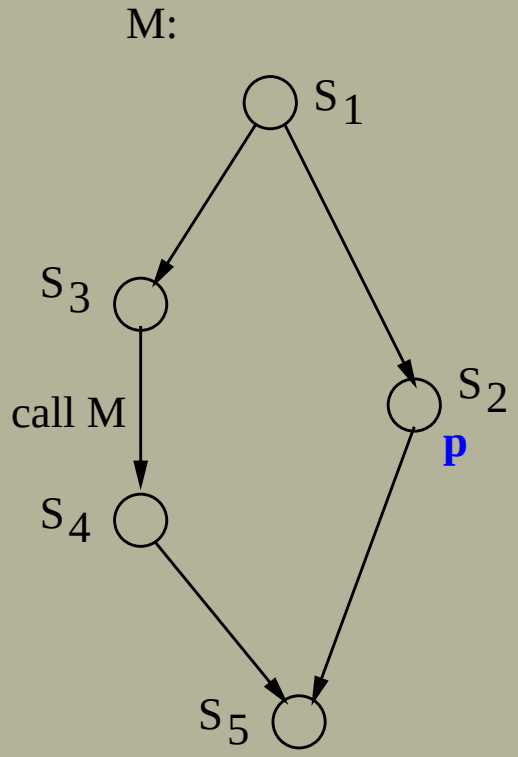
$S = \{s_1, s_2, s_3, s_4, s_5\}$ – stack alphabets

Transition Rules:

$$s_1 \hookrightarrow [s_2]$$
$$s_1 \hookrightarrow [s_3]$$
$$s_2 \hookrightarrow [s_5]$$
$$s_4 \hookrightarrow [s_5]$$
$$s_5 \hookrightarrow []$$

Program Model

Control-flow graph



Push-down Model

$S = \{s_1, s_2, s_3, s_4, s_5\}$ – stack alphabets

Transition Rules:

$$s_1 \hookrightarrow [s_2]$$
$$s_1 \hookrightarrow [s_3]$$
$$s_2 \hookrightarrow [s_5]$$
$$s_4 \hookrightarrow [s_5]$$
$$s_5 \hookrightarrow []$$
$$s_3 \hookrightarrow [s_1, s_4]$$

LTL formula

Examples

Fp existence of reachable state where p holds true

Gp property p holds in all reachable states

GFp in any infinite path, states where p holds appear infinitely many times

FGp in any infinite path, a state is reached from where p holds true globally

Büchi Automata

$\mathcal{B} = (Q, \rightarrow, \Sigma, Q_0, F)$ where

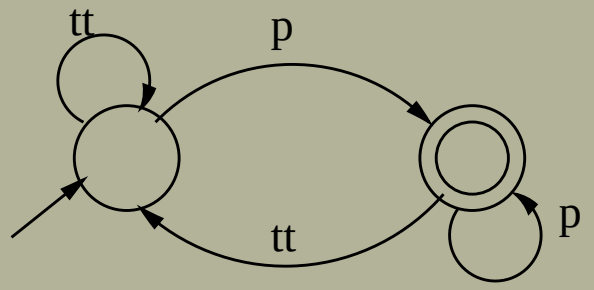
Q – States,
Q_0 – Init States,
Σ – Propositions,
$\rightarrow$ – $Q \times \Sigma \times Q$,
F – Final States
Accepting Sequence visits F
infinitely many times

Büchi Automata

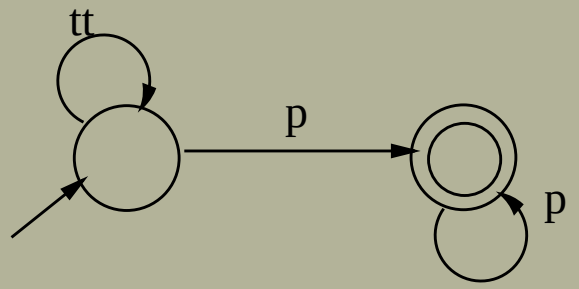
$\mathcal{B} = (Q, \rightarrow, \Sigma, Q_0, F)$ where

Q – States,
Q_0 – Init States,
Σ – Propositions,
$\rightarrow$ – $Q \times \Sigma \times Q$,
F – Final States
Accepting Sequence visits F infinitely many times

GF(p)



FG(p)



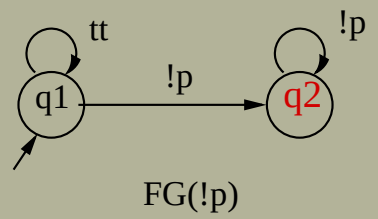
Product Construction

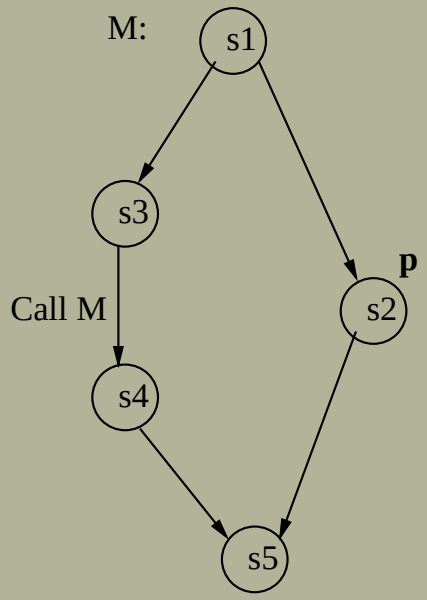
PDA rules	Büchi rules	Product rules
	p is true at s_1	
$s_1 \hookrightarrow []$	$q_1 \xrightarrow{p} q_2$	$(q_1, s_1) \hookrightarrow (q_2, [])$
$s_1 \hookrightarrow [s_2]$	$q_1 \xrightarrow{p} q_2$	$(q_1, s_1) \hookrightarrow (q_2, [s2])$
$s_1 \hookrightarrow [t, s_2]$	$q_1 \xrightarrow{p} q_2$	$(q_1, s_1) \hookrightarrow (q_2, [t, s2])$

Product is another Push-down System

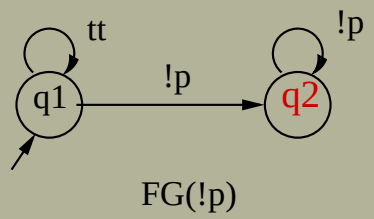
Abstract Product Construction – II

Abstract Product Construction – II

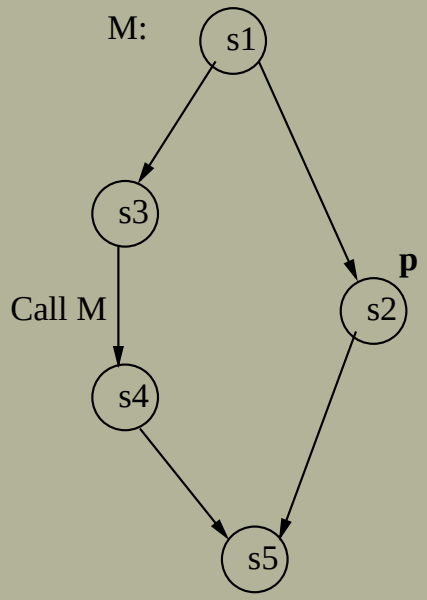


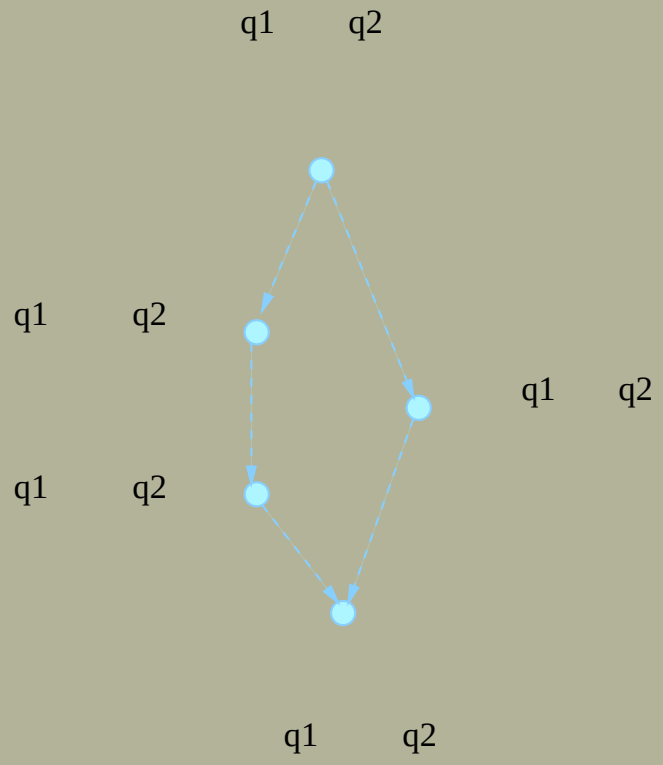


Abstract Product Construction – II



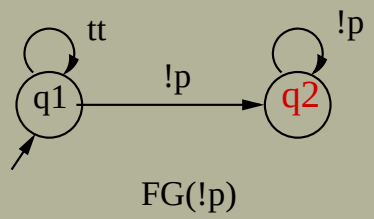
FG(!p)



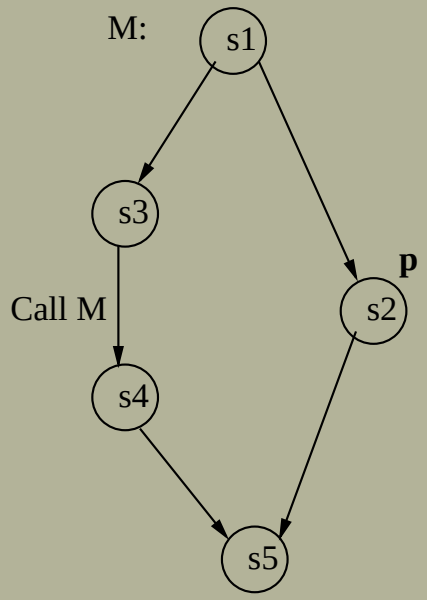


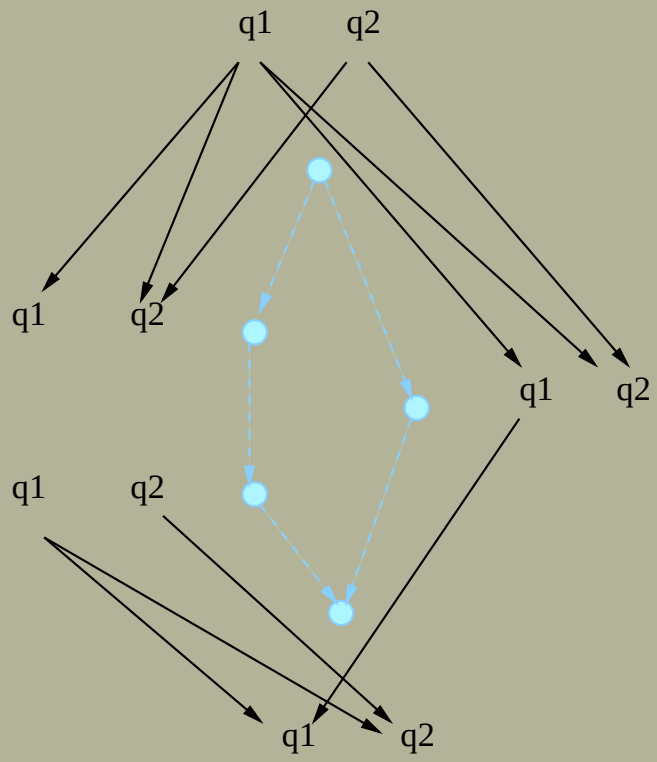
Abstract Product

Abstract Product Construction – II



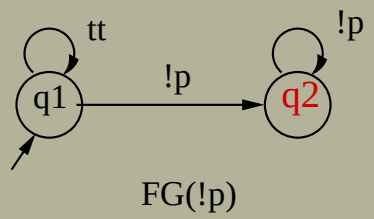
FG(!p)



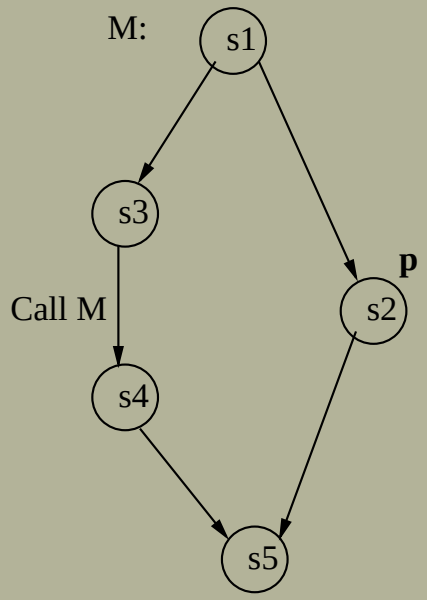


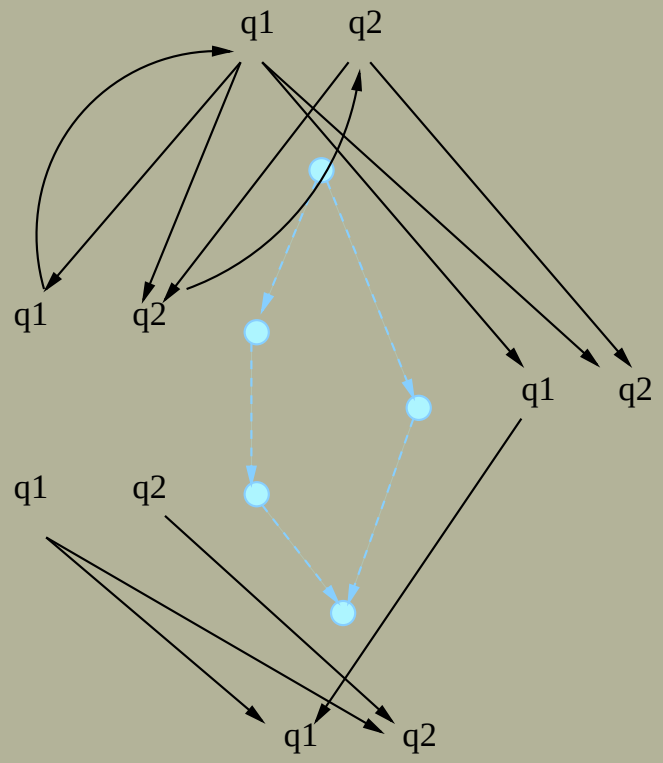
Abstract Product

Abstract Product Construction – II



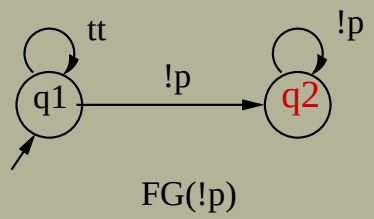
FG(!p)



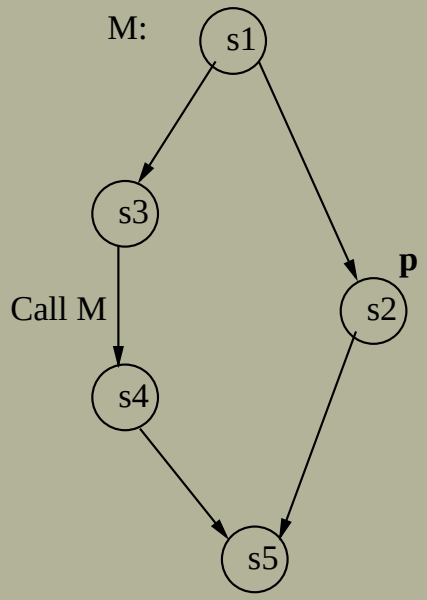


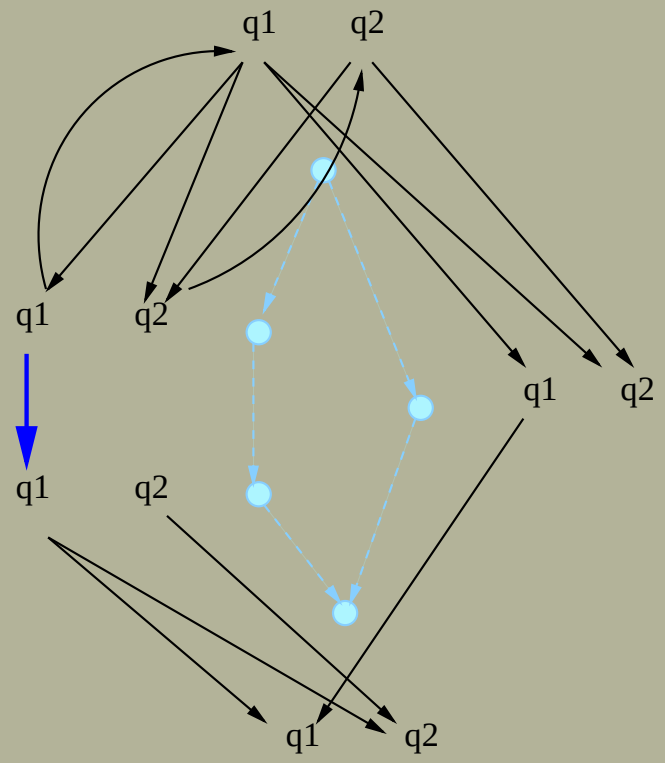
Abstract Product

Abstract Product Construction – II



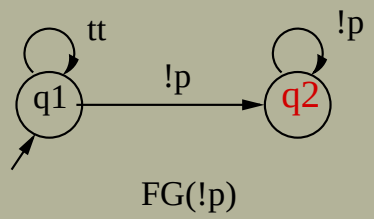
FG(!p)



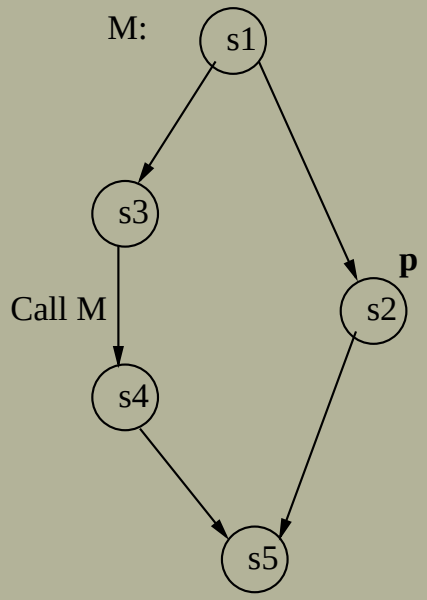


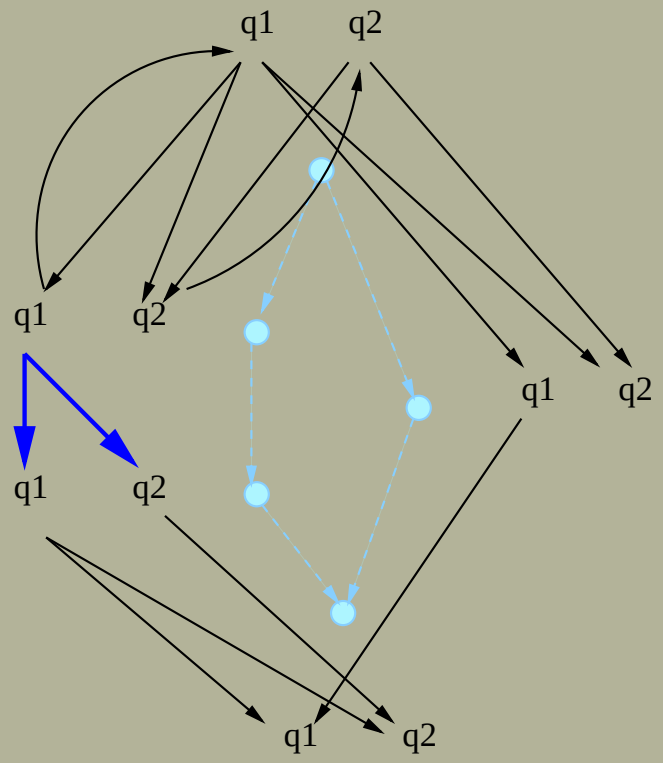
Abstract Product

Abstract Product Construction – II



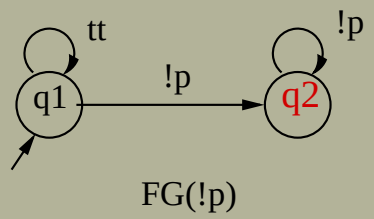
FG(!p)



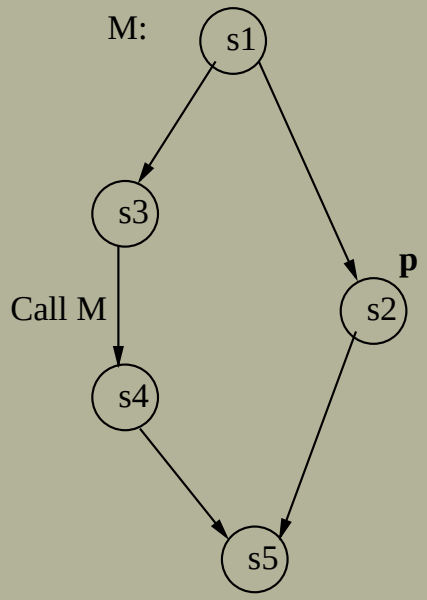


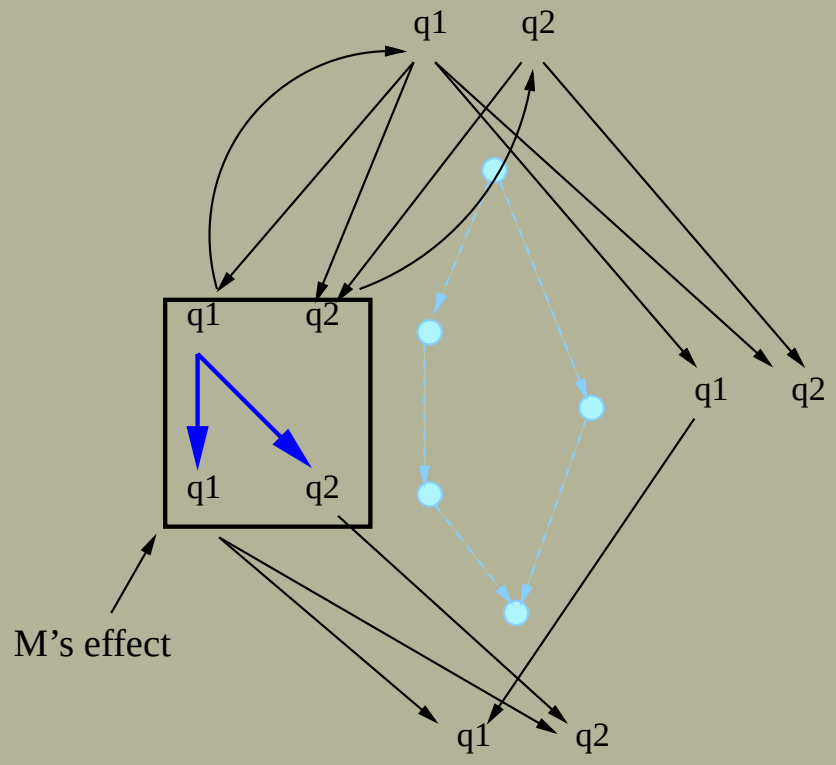
Abstract Product

Abstract Product Construction – II



FG(!p)





Abstract Product

Summarizing Effects of Procedure Calls

- `erase` – How the state of Büchi automata changes when the top-of-stack symbol is erased
- Keep track of whether a final state of Büchi automata is visited

Summarizing Effects of Procedure Calls

- `erase` – How the state of Büchi automata changes when the top-of-stack symbol is erased
- Keep track of whether a final state of Büchi automata is visited

Base Case $\quad \texttt{erase}(q_1, s_1, B, q_2)\ \texttt{if} \quad (q_1, s_1) \hookrightarrow (q_2, [])$ & $B = q_1 \in F$

Summarizing Effects of Procedure Calls

- `erase` – How the state of Büchi automata changes when the top-of-stack symbol is erased
- Keep track of whether a final state of Büchi automata is visited

Base Case	$\texttt{erase}(q_1, s_1, B, q_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [])$ & $B = q_1 \in F$
Direct Transfer	$\texttt{erase}(q_1, s_1, B, q_3)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2]) \wedge$ $\texttt{erase}(q_2, s_2, B_2, q_3)$ & $B = (q_1 \in F) \vee B_2$

Summarizing Effects of Procedure Calls

- `erase` – How the state of Büchi automata changes when the top-of-stack symbol is erased
- Keep track of whether a final state of Büchi automata is visited

Base Case	$\texttt{erase}(q_1, s_1, B, q_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [])$ & $B = q_1 \in F$
Direct Transfer	$\texttt{erase}(q_1, s_1, B, q_3)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2]) \wedge$ $\texttt{erase}(q_2, s_2, B_2, q_3)$ & $B = (q_1 \in F) \vee B_2$
Call Transitions	$\texttt{erase}(q_1, s_1, B, q_4)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2, s_3]) \wedge$ $\texttt{erase}(q_2, s_2, B_2, q_3) \wedge$ $\texttt{erase}(q_3, s_3, B_3, q_4)$ & $B = (q_1 \in F) \vee B_2 \vee B_3$

Abstract Product from PDS rules & `erase`

$$\hookrightarrow + \texttt{erase}$$

- $\circ\!\!\longrightarrow$ – keeps track of whether a final Büchi state is visited (goodness label) & whether the stack depth has increased (resource label)

Abstract Product from PDS rules & `erase`

$$\hookrightarrow + \texttt{erase}$$

- $\circ\!\!\longrightarrow$ – keeps track of whether a final Büchi state is visited (goodness label) & whether the stack depth has increased (resource label)

Direct Transfer $\quad (q_1, s_1) \overset{B,0}{\circ\!\!\longrightarrow} (q_2, s_2)$ `if` $\quad (q_1, s_1) \hookrightarrow (q_2, [s_2])$ & $B = q_1 \in F$

Abstract Product from PDS rules & `erase`

$$\hookrightarrow + \texttt{erase}$$

- $\circ\!\!\longrightarrow$ – keeps track of whether a final Büchi state is visited (goodness label) & whether the stack depth has increased (resource label)

Direct Transfer	$(q_1, s_1) \overset{B,0}{\circ\!\!\longrightarrow} (q_2, s_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2])$ & $B = q_1 \in F$
Call with no return	$(q_1, s_1) \overset{B,1}{\circ\!\!\longrightarrow} (q_2, s_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2, s_3])$ & $B = q_1 \in F$

Abstract Product from PDS rules & `erase`

$$\hookrightarrow + \texttt{erase}$$

- $\circ\!\!\longrightarrow$ – keeps track of whether a final Büchi state is visited (goodness label) & whether the stack depth has increased (resource label)

Direct Transfer	$(q_1, s_1) \overset{B,0}{\circ\!\!\longrightarrow} (q_2, s_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2])$ & $B = q_1 \in F$
Call with no return	$(q_1, s_1) \overset{B,1}{\circ\!\!\longrightarrow} (q_2, s_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2, s_3])$ & $B = q_1 \in F$
Call with matched return	$(q_1, s_1) \overset{B,0}{\circ\!\!\longrightarrow} (q_3, s_3)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2, s_3]) \wedge$ $\texttt{erase}(q_2, s_2, B_2, q_3)$ & $B = (q_1 \in F) \vee B_2 \vee B_3$

Abstract Product from PDS rules & `erase`

$$\hookrightarrow + \texttt{erase}$$

- $\circ\!\!\longrightarrow$ – keeps track of whether a final Büchi state is visited (goodness label) & whether the stack depth has increased (resource label)

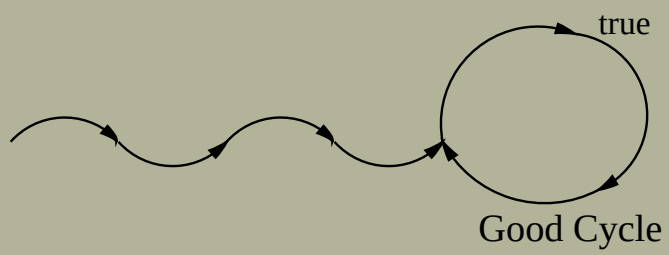
Direct Transfer	$(q_1, s_1) \overset{B,0}{\circ\!\!\longrightarrow} (q_2, s_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2])$ & $B = q_1 \in F$
Call with no return	$(q_1, s_1) \overset{B,1}{\circ\!\!\longrightarrow} (q_2, s_2)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2, s_3])$ & $B = q_1 \in F$
Call with matched return	$(q_1, s_1) \overset{B,0}{\circ\!\!\longrightarrow} (q_3, s_3)$ `if`	$(q_1, s_1) \hookrightarrow (q_2, [s_2, s_3]) \wedge$ $\texttt{erase}(q_2, s_2, B_2, q_3)$ & $B = (q_1 \in F) \vee B_2 \vee B_3$

Finite Set of $\circ\!\!\longrightarrow$ rules: R-graph

Accepting Sequence in R-graph

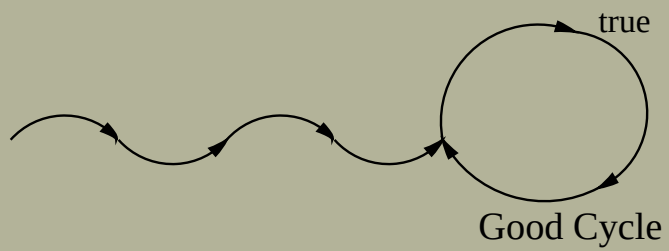
- $\overset{\texttt{true},\_}{\circ\!\!\longrightarrow}$ appears in a cycle – *Good Cycle*

- Accepting Sequence: Sequence of transitions from the start state leading to a good cycle



Accepting Sequence in R-graph

- $\overset{\texttt{true},\_}{\circ\!\!\longrightarrow}$ appears in a cycle – *Good Cycle*
- Accepting Sequence: Sequence of transitions from the start state leading to a good cycle



- Resource Constraint: Good cycle consists of only $\overset{-,0}{\circ\!\!\longrightarrow}$ edges
  - ⋆ Property is satisfied for any finite depth stack

SCC Detection

- Disjunctive Property: At least one $\circ\!\!\longrightarrow$ is labeled by `true`
- Conjunctive Property: All $\circ\!\!\longrightarrow$ is labeled by 0

SCC Detection

- Disjunctive Property: At least one $\circ\!\!\longrightarrow$ is labeled by `true`
- Conjunctive Property: All $\circ\!\!\longrightarrow$ is labeled by 0

- Tarjan's SCC detection algorithm
  - ⋆ Disjunctive Property: Couvreur – FM'99

SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

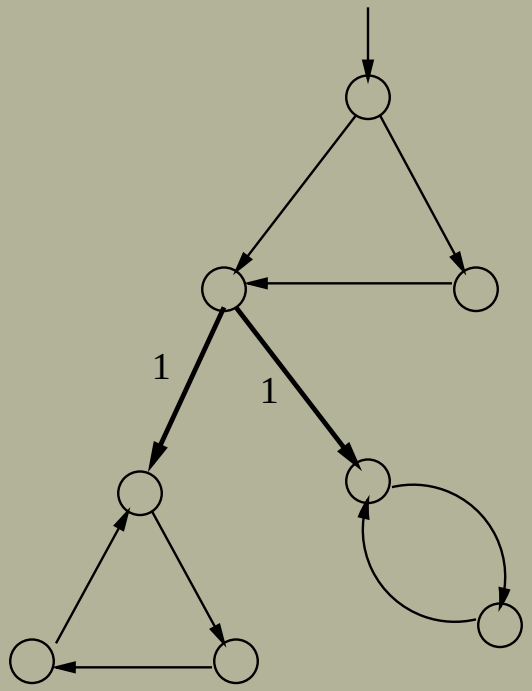
SCC detection performed per partition

SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition

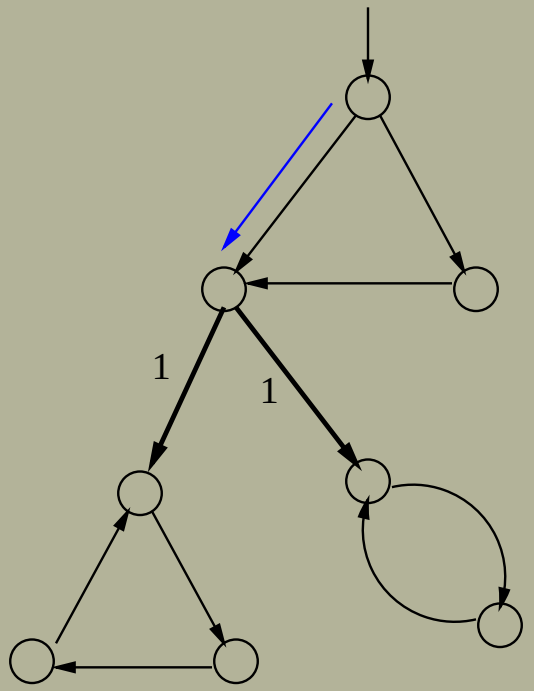


SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition

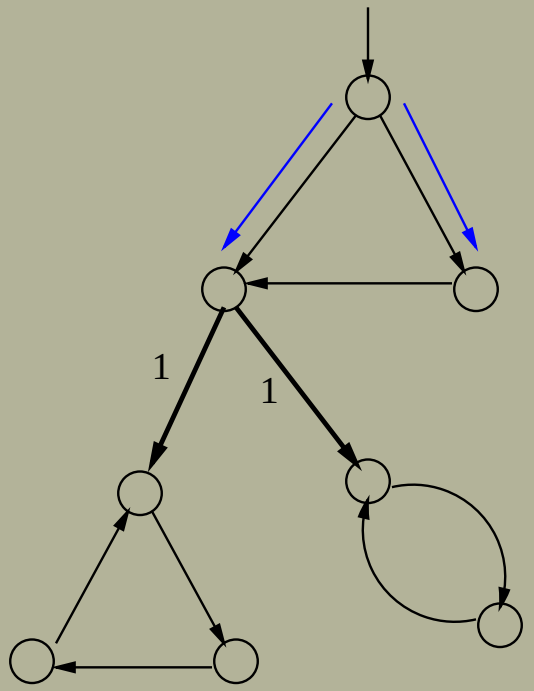


SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition

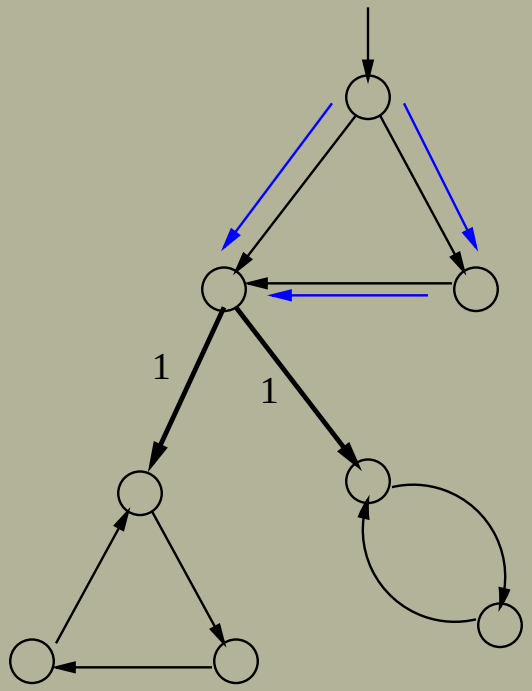


SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition

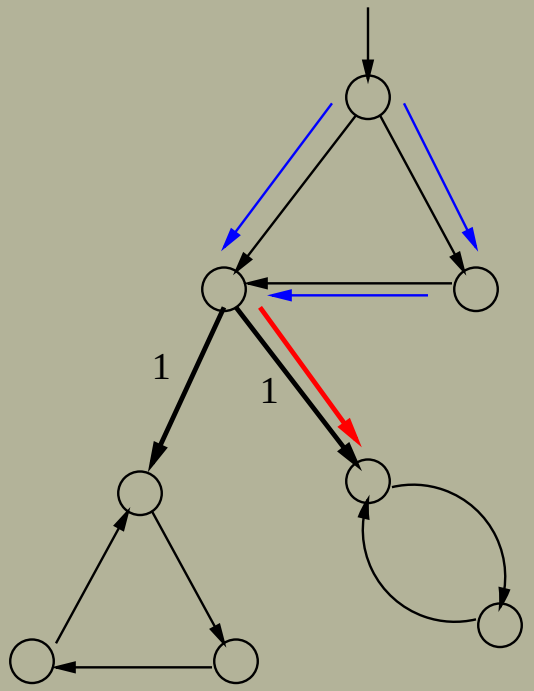


SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition

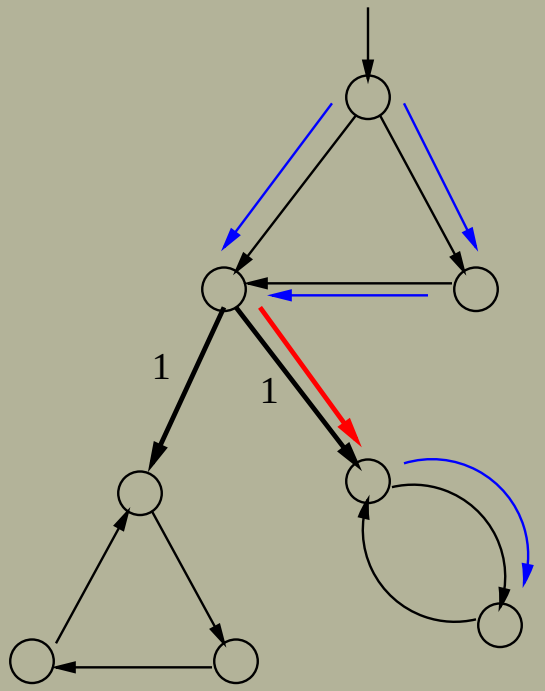


SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition

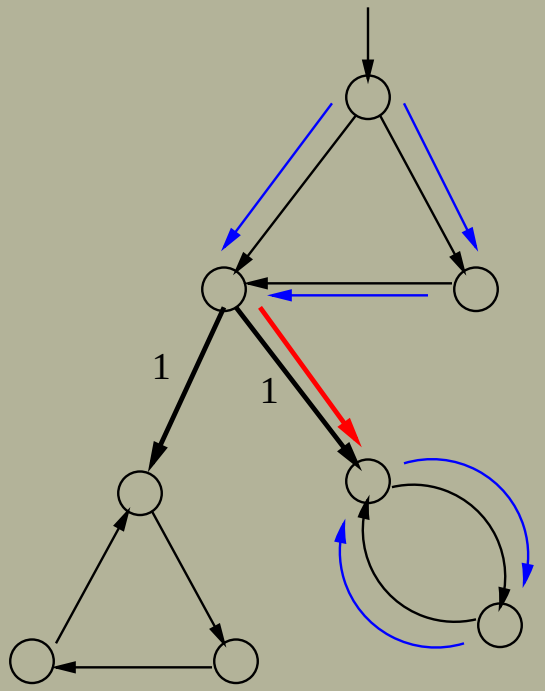


SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition

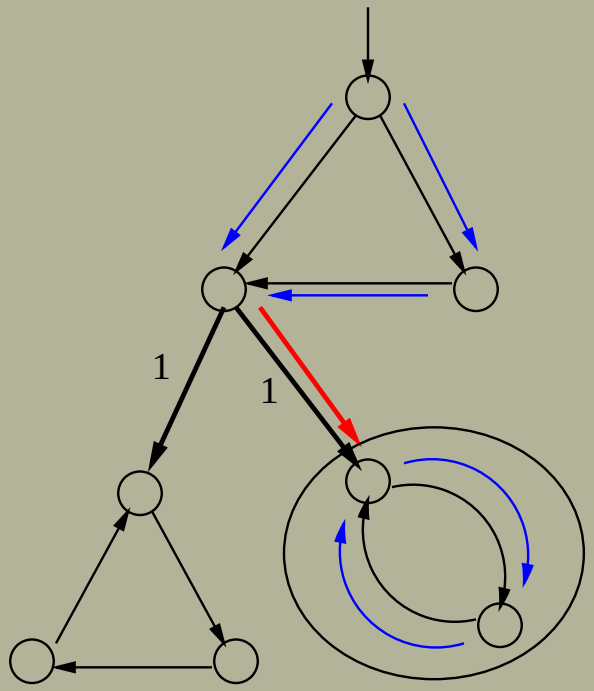


SCC Detection with Conjunctive Constraints

R-graph partitioned in DFS-graphs with only 0-edges

Each partition bridged by 1-edges

SCC detection performed per partition



Summary

- Recursion handled by summarizing the effect of procedure calls
- Distinguishes finite stack runs from stack diverging runs
  - ⋆ Verification is independent of recursion control parameter: abstracted to generate finite model in terms of data domain
  - ⋆ Result obtained is valid for all possible finite values of recursion control parameter
- Implemented in XSB: local, on-the-fly model checker

References

- Rachability Analysis of push-down automata
  Bouajjani, Esparza, Maler – CONCUR'97

- A direct symbolic approach to model checking push-down systems
  Finkel, Willems, Wolper – INFINITY'97

- Efficient Algorithms for model checking push-down systems
  Esparza, Hansel, Rossmanith, Schwoon – CAV'00

- A BDD-based model checker for recursive programs
  Esparza, Schwoon – CAV'01

- Precise interprocedural dataflow analysis via graph reachability
  Reps, Horwitz, Sagiv – POPL'95

Mu-calculus Model Checking for Programs

- Program models: Control flow graphs with action labels on transitions
- Property expressed in modal mu-calculus
- Given the set of formula true at a state s of control flow model, find the set of formula true at the states that has transitions to s

Mu-calculus

Syntax

$$F \to P \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \langle a \rangle F \mid [a]F \mid X$$
$$X \to \mu X.F \mid \nu X.F$$

Semantics: $[\![F]\!]_e$ gives the set of states where F is true in the current environment

$$e : X \mapsto 2^S$$

$$\begin{array}{rcl}
[\![P]\!]_e & = & \{s \mid P \texttt{ is true in } s\} \\
[\![F_1 \vee F_1]\!]_e & = & [\![F_1]\!]_e \cup [\![F_2]\!]_e \\
[\![F_1 \wedge F_1]\!]_e & = & [\![F_1]\!]_e \cap [\![F_2]\!]_e \\
[\![\langle a \rangle F]\!]_e & = & \{s \mid \exists s \xrightarrow{a} t \texttt{ s.t. } t \in [\![F]\!]_e\} \\
[\![[a]F]\!]_e & = & \{s \mid \forall s \xrightarrow{a} t \texttt{ s.t. } t \in [\![F]\!]_e\} \\
X & = & e(X) \\
\mu X.F & = & \tau^i(\texttt{false})
\end{array}$$

where $\tau^i(P) = \tau^{i-1}(\tau(P))$

$$\tau(\texttt{false}) = [\![F]\!]_{e[X \mapsto \texttt{false}]}$$

Examples

- $X_1 = \nu X_1.(\langle - \rangle \mathtt{tt} \wedge [-]X_1)$: freedom from deadlock
- $X_2 = \mu X_2.(\langle a \rangle X_3 \vee \langle - \rangle X_2)$
  $X_3 = \mu X_3.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X_3)$: there exists a path where action a is followed eventually by action b
- Properties can be expressed using arbitrary interleaving of least and greatest fixed point formula
- Focus: Alternation-free mu-calculus
  - ⋆ Each formula can be interpreted separately starting from the inner most fixed point formula

Model Checking

- Compute the set of formula that are true in state s of procedure P using the set of formula that are true in the end-state of P
- $X\ (s)\ Y$: if $s \xrightarrow{a} t$ then $X\ (a)\ X_1\ (t)\ Y$ (a is an atomic action or a call to a procedure)
- Initialization
  - ⋆ End-states each procedure has identity mapping: $X\ (\texttt{endstate})\ X$
  - ⋆ Least fixed point formula variables are false at all states
  - ⋆ Greatest fixed point formula variables are true at all states
- $M \models X$ if $X\ (s)$ `deadlock` where
  - ⋆ s is the start state of M
  - ⋆ `deadlock` is the set of formula true at the endstate of M

Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

Example

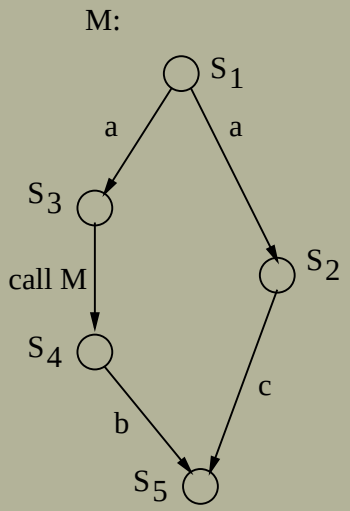
$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$

Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

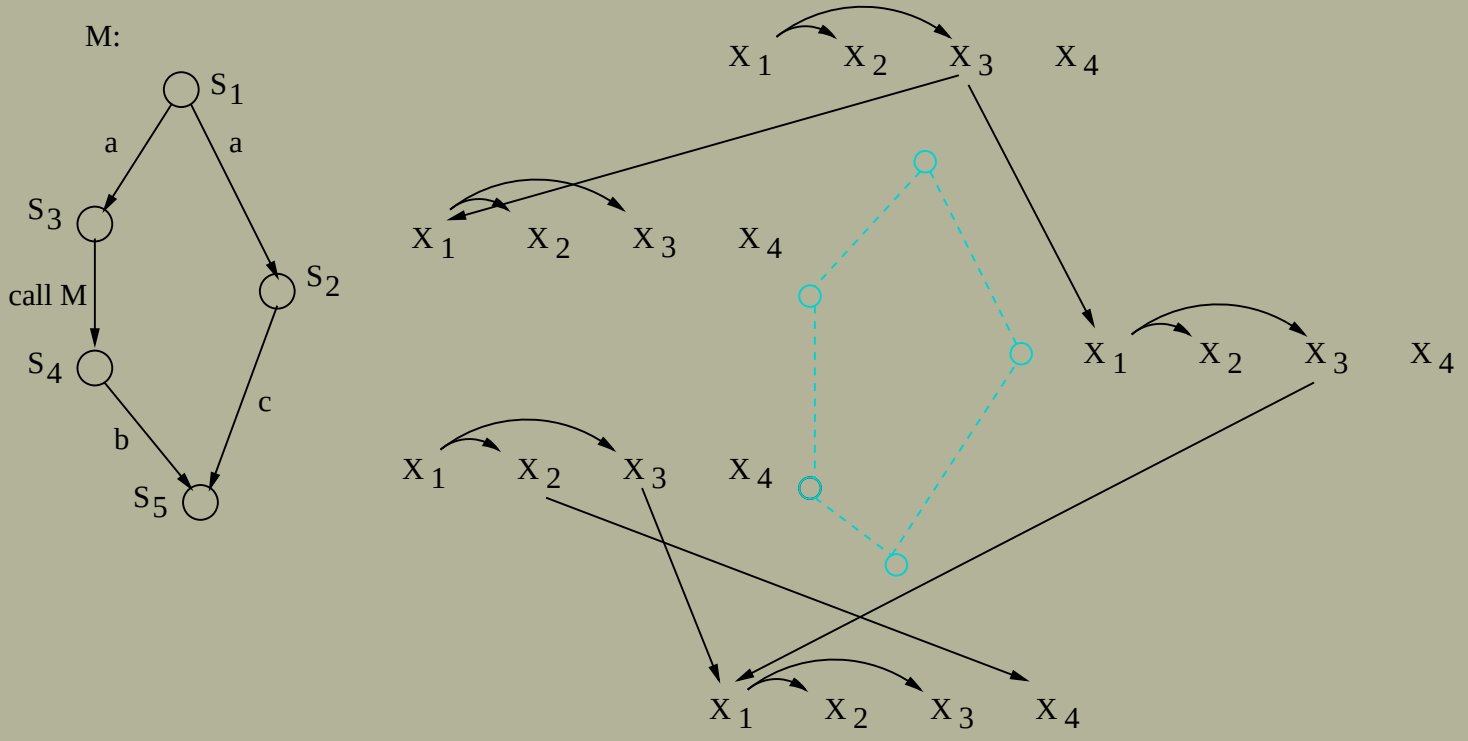
$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

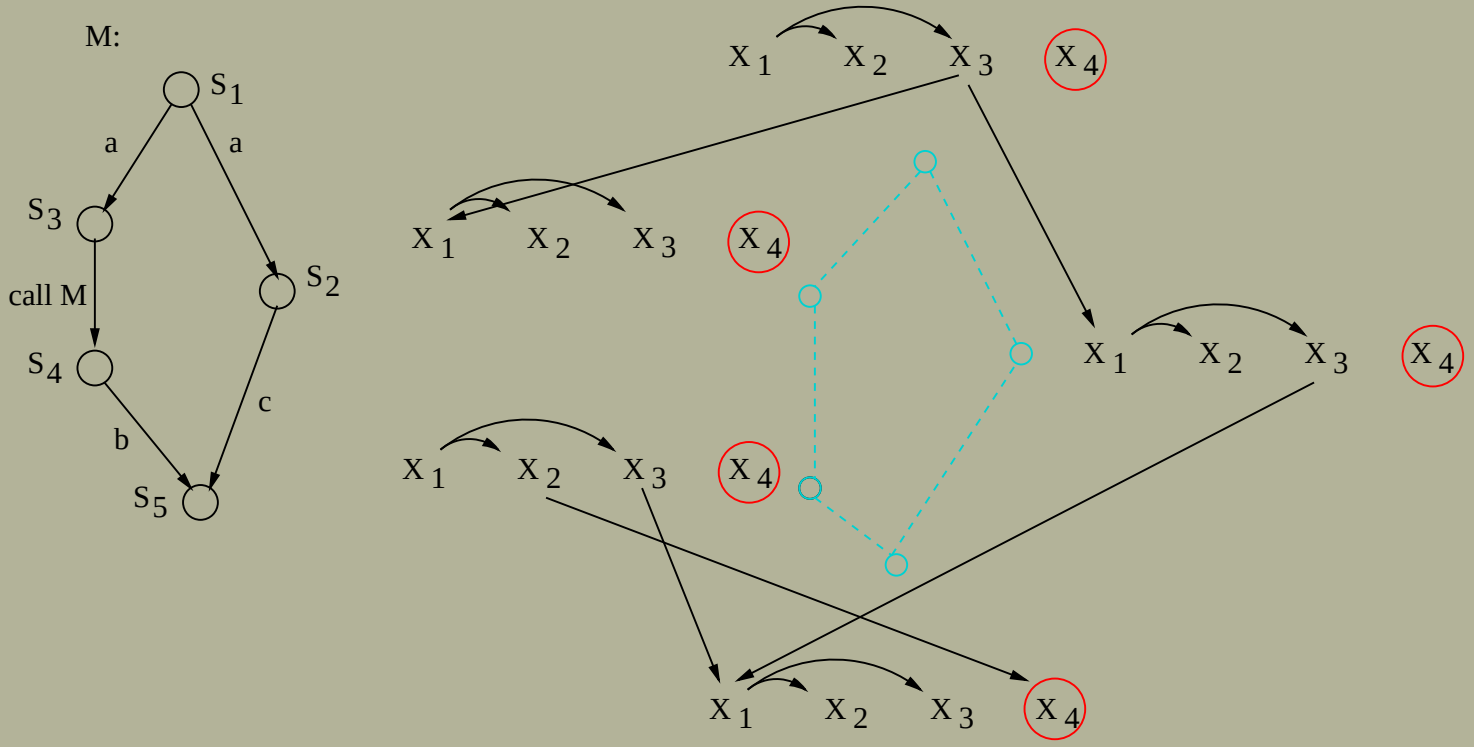
$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

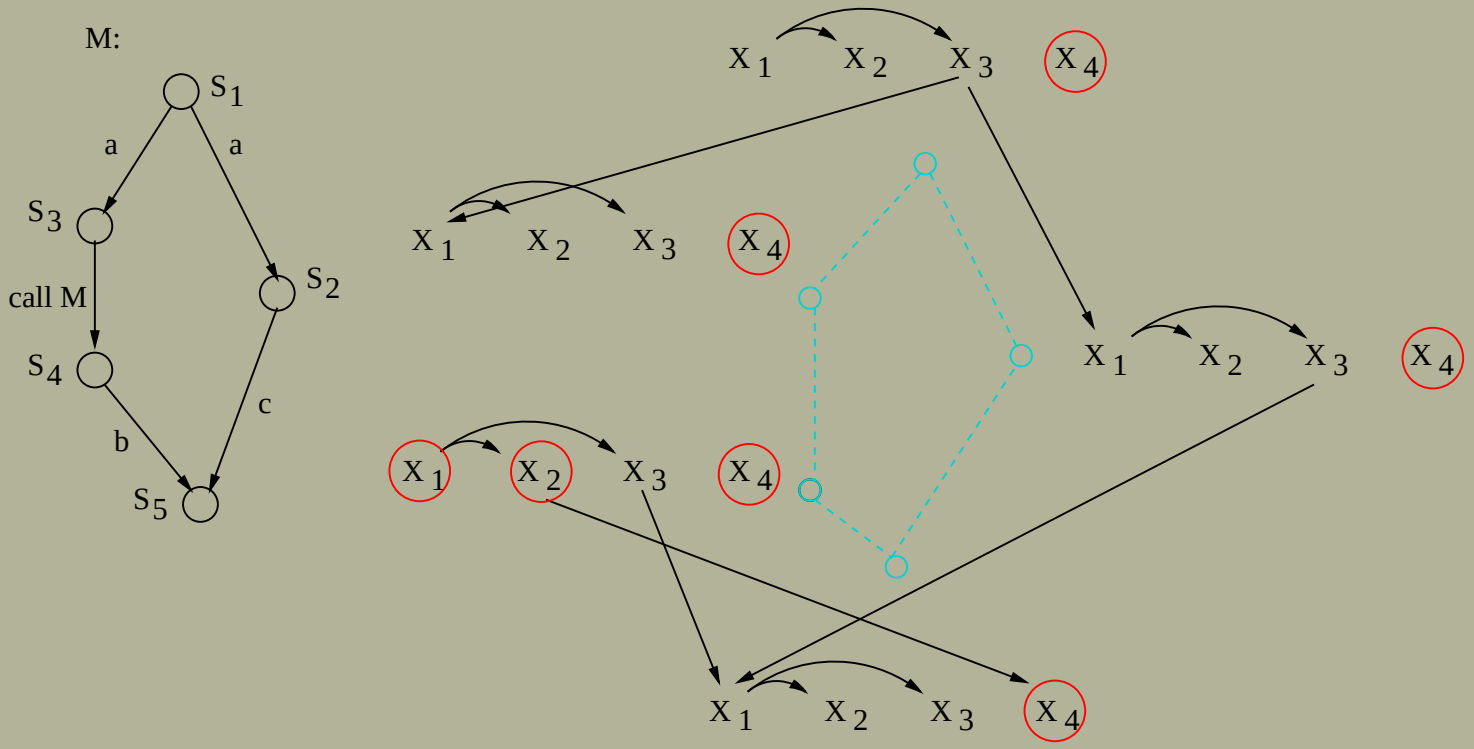
$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

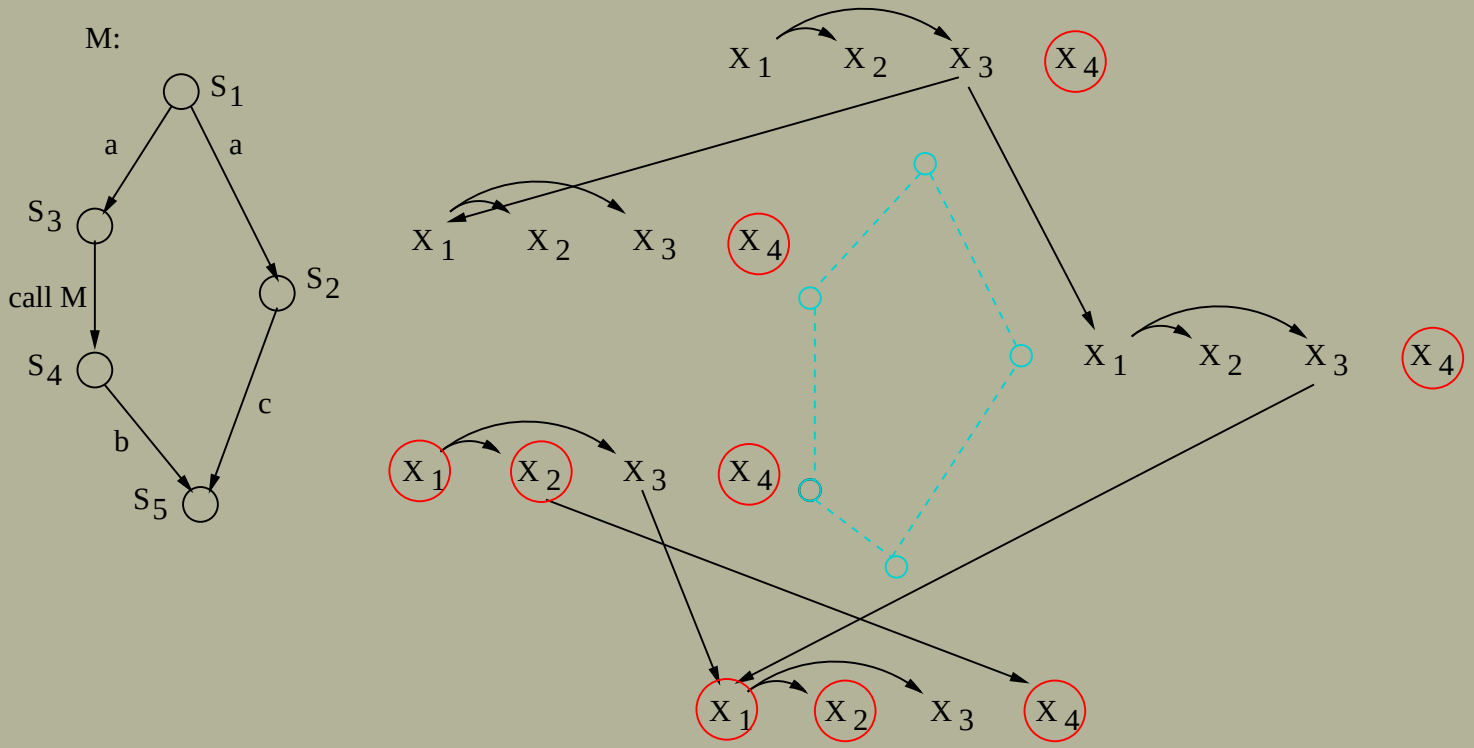
$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

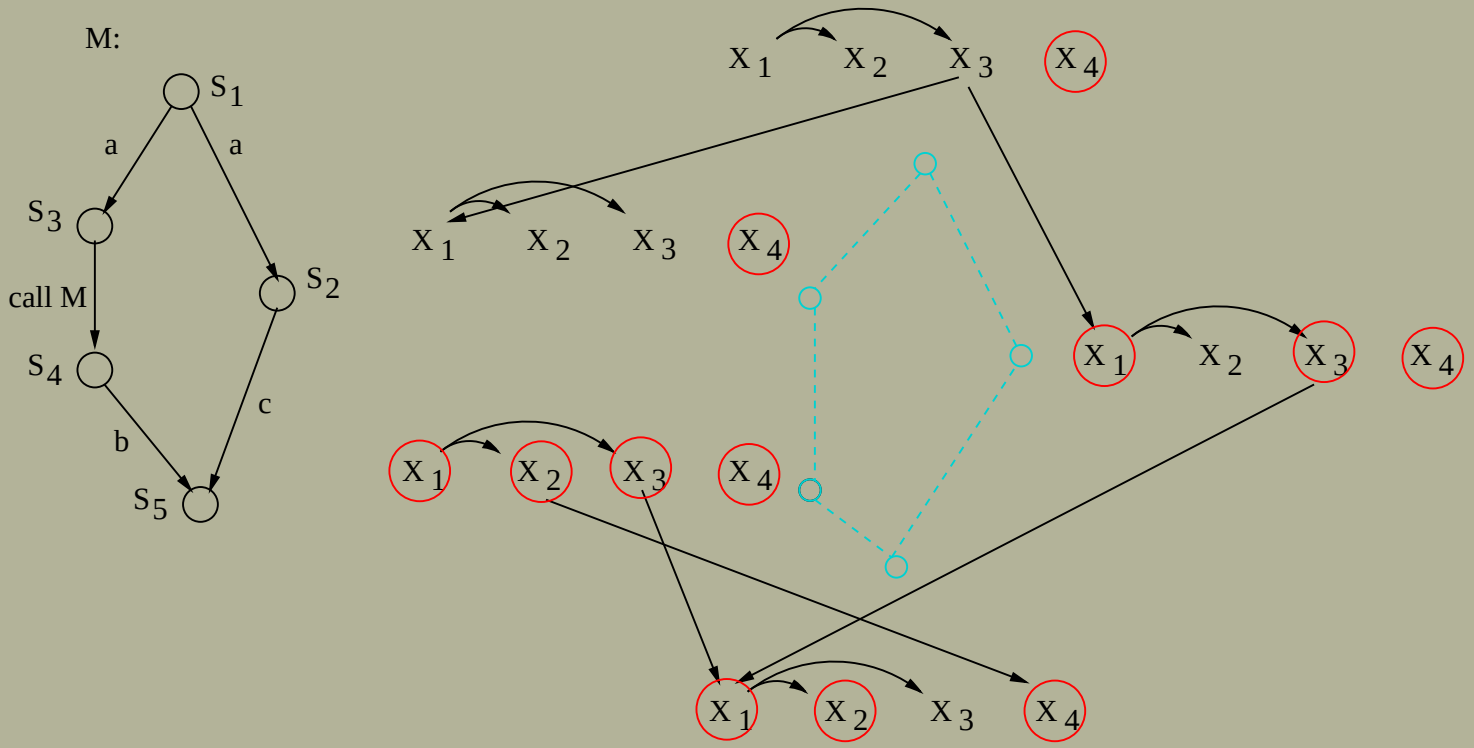
$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

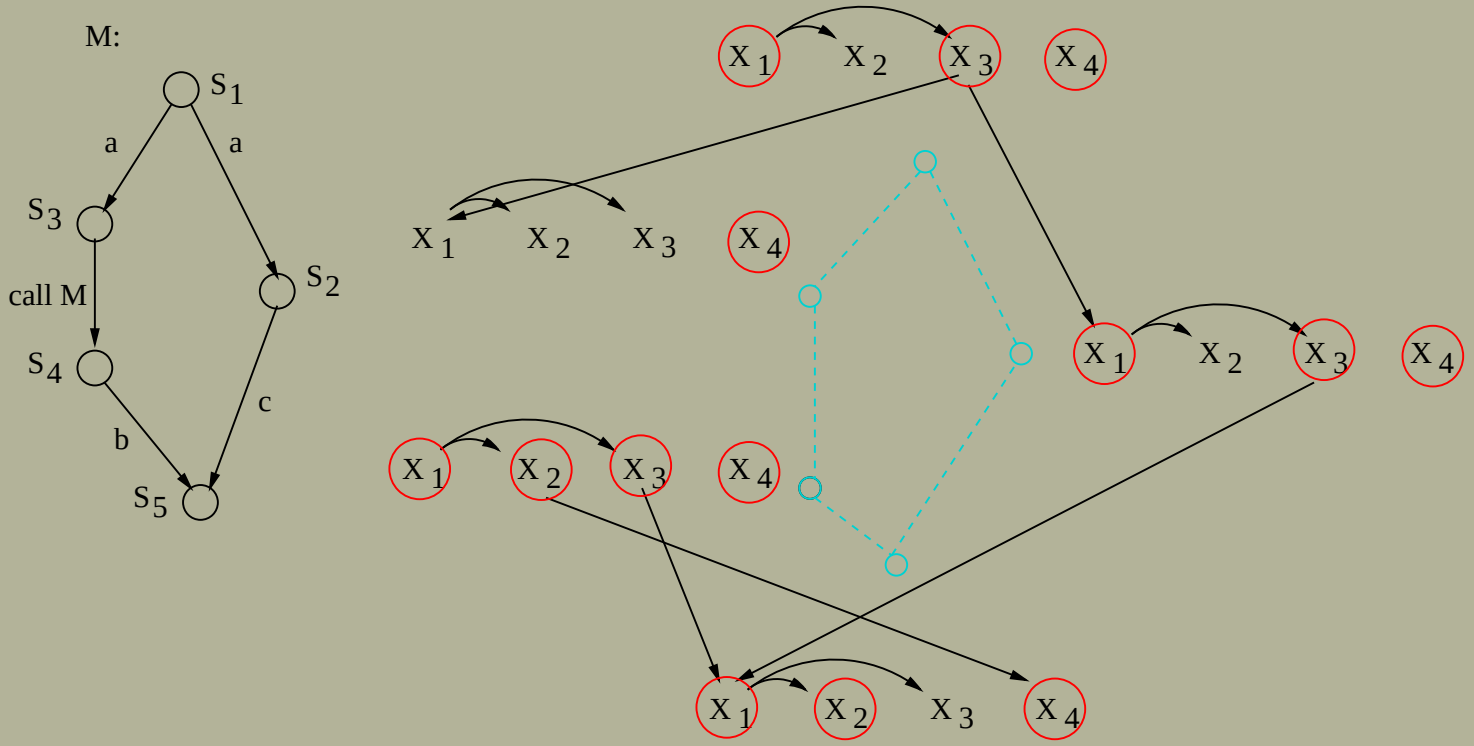
$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

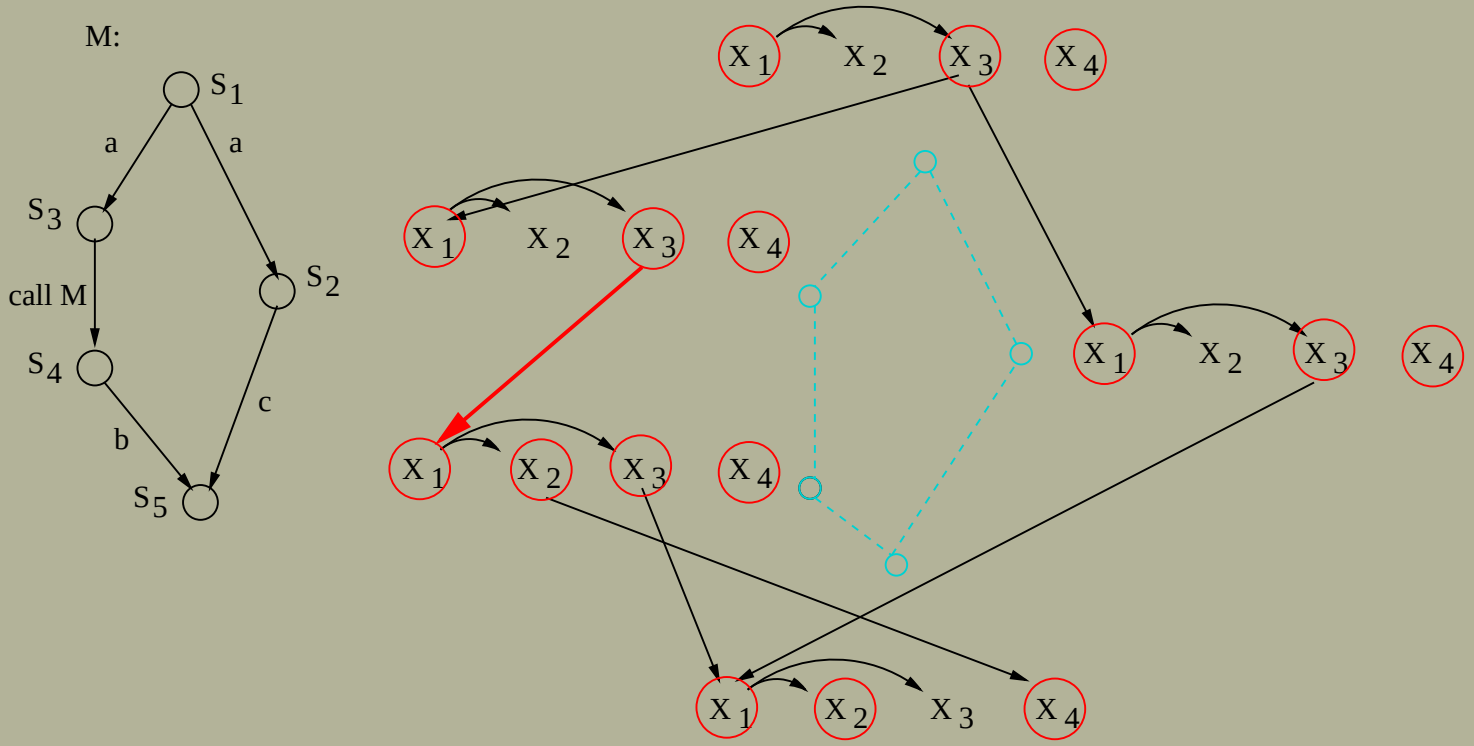
$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



Example

$$X = \mu X.(\langle b \rangle \mathtt{tt} \vee \langle - \rangle X)$$

$$X_1 = X_2 \vee X_3 \quad X_2 = \langle b \rangle X_4 \quad X_3 = \langle - \rangle X_1 \quad X_4 = \mathtt{tt}$$



$$X1(s)X_4 \implies s \models X$$

References

- Model checking for context-free processes
  Burkart, Steffen – CONCUR'92

- Model checking the full modal mu-calculus for infinite sequential processes
  Burkart, Steffen – TCS'99