Verification of Infinite-state Systems

Javier Esparza

Software Reliability and Security Group

Institute for Formal Methods in Computer Science

University of Stuttgart

Software model checking

Challenge: develop model-checking techniques for ‘higher-level’ software.

Three main research questions:

Software model checking

Challenge: develop model-checking techniques for ‘higher-level’ software.

Three main research questions:

Integration of the techniques in the system development process.

Software model checking

Challenge: develop model-checking techniques for ‘higher-level’ software.

Three main research questions:

Integration of the techniques in the system development process.

- PathStar [Holzmann, Smith, IEEE Trans. on Soft. Eng.]:
  Checking Lucent’s PathStar access server.
- Slam [Ball, Rajamani, POPL’02.]: Checking Windows XP drivers.

Software model checking

Challenge: develop model-checking techniques for ‘higher-level’ software.

Three main research questions:

Integration of the techniques in the system development process.

- PathStar [Holzmann, Smith, IEEE Trans. on Soft. Eng.]:
  Checking Lucent’s PathStar access server.
- Slam [Ball, Rajamani, POPL’02.]: Checking Windows XP drivers.

Automatic extraction of formal models from code.

Software model checking

Challenge: develop model-checking techniques for ‘higher-level’ software.

Three main research questions:

Integration of the techniques in the system development process.

- PathStar [Holzmann, Smith, IEEE Trans. on Soft. Eng.]:
  Checking Lucent’s PathStar access server.
- Slam [Ball, Rajamani, POPL’02.]: Checking Windows XP drivers.

Automatic extraction of formal models from code.

- Work of the abstract interpretation and static analysis community.
- Bandera [Corbett,Dwyer,Hatcliff et al., ICSE’00]:
  From Java code to model-checkable models through abstraction/static analysis.

Software model checking

Challenge: develop model-checking techniques for ‘higher-level’ software.

Three main research questions:

Integration of the techniques in the system development process.

- PathStar [Holzmann, Smith, IEEE Trans. on Soft. Eng.]:
  Checking Lucent’s PathStar access server.
- Slam [Ball, Rajamani, POPL’02.]: Checking Windows XP drivers.

Automatic extraction of formal models from code.

- Work of the abstract interpretation and static analysis community.
- Bandera [Corbett,Dwyer,Hatcliff et al., ICSE’00]:
  From Java code to model-checkable models through abstraction/static analysis.

Exploration of infinite-state spaces.

Integration in the system development process

PathStar

Checking a telephone switch.

- One system
- Verification interleaved with design (300 versions)
- Highly concurrent code
- Complex specification (80/200 properties)

Slam

Checking Windows XP drivers.

- Many systems
- Post-mortem verification
- Sequential code
- Simple specification (i.e.,correct locking/unlocking)

Sources of infinity in software systems

Data manipulation: integers, lists, trees, more general pointer structures, . . .

Control structures: procedures , process creation, . . .

Asynchronous communication: unbounded FIFO queues.

Parameters: number of processes, duration of delays . . .

Real-time: discrete or dense domains.

Current approach of (most of) the ISMC community

Model data abstractions of the program by means of extended automata or equivalent models.

Current approach of (most of) the ISMC community

Model data abstractions of the program by means of extended automata or equivalent models.

Using the automata theoretic-approach to model checking, reduce the verification problem to reachability or repeated reachability problems.
(See Moshe Vardi's course.)

Current approach of (most of) the ISMC community

Model data abstractions of the program by means of extended automata or equivalent models.

Using the automata theoretic-approach to model checking, reduce the verification problem to reachability or repeated reachability problems.
(See Moshe Vardi's course.)

Develop algorithms or semi-algorithms for these problems using symbolic search, accelerations, and learning.
(See this course.)

Current approach of (most of) the ISMC community

Model data abstractions of the program by means of extended automata or equivalent models.

Using the automata theoretic-approach to model checking, reduce the verification problem to reachability or repeated reachability problems.
(See Moshe Vardi's course.)

Develop algorithms or semi-algorithms for these problems using symbolic search, accelerations, and learning.
(See this course.)

Reintroduce the abstracted data incrementally by means of predicate abstraction and counterexample-guided abstraction refinement.
(See Orna Grumberg's course.)

Extended automata: Syntax

Extended automaton $=$ automaton whose transitions are
guarded by and operate on data structures.

An extended automaton is a tuple $E = (X, Q, T, G, A)$ where

- $X = \{x_1, \ldots, x_n\}$ is a finite set of variables over sets $V_1, \ldots, V_n$ of values,
- Q is a finite set of control states,
- $T \subseteq Q \times Q$ is a set of transitions or rules,
- G associates to each transition a guard
  (a predicate over X, the condition under which the transition can be taken),
- A associates to each transition an action
  (a possibly nondeterministic assignment to X)

Notation for transitions: $q \xrightarrow[a]{g} q'$, where g guard and a action.

Remark: variables over finite sets of values can be encoded into the states.

Extended automata: Semantics

A configuration is a tuple $\langle q, v_1, \ldots, v_n \rangle$, where

- q is a state, and
- $v_1, \ldots, v_n$ is a valuation of $x_1, \ldots, x_n$ (i.e., $v_i \in V_i$ for every $1 \leq i \leq n$).

The transition system $\mathcal{T}_E$ of an extended automaton E has:

- the set of all configurations as nodes, and
- an edge $\langle q, v_1, \ldots, v_n \rangle \longrightarrow \langle q', v_1', \ldots, v_n' \rangle$ iff E has a transition $q \xrightarrow[a]{g} q'$ such that
  - $v_1, \ldots, v_n$ satisfies the guard g, and
  - $v_1', \ldots, v_n'$ is one of the possible results of applying a to $v_1, \ldots, v_n$.

Some classes of extended automata

Automata	Variables	Transition
Timed automata	clocks (reals)	$q \xrightarrow[c_2:=0]{c_1\geq 2} q'$
Pushdown automata	stack	$q \xrightarrow[a/ba]{top=a} q'$
(Ext. of) Petri nets	counters (integers)	$q \xrightarrow[x_2:=x_2+x_3]{x_1=0} q'$
FIFO automata	queues	$q \xrightarrow[l_2?a]{l_1\neq\epsilon} q'$

Networks of extended automata

A network of extended automata (or just a network) is a tuple $\langle E_1, \ldots, E_m \rangle$ of extended automata over the same set of variables X.

The asynchronous product of a network $\langle E_1, \ldots, E_m \rangle$ is the extended automaton having

- the set $Q = Q_1 \times \ldots \times Q_m$ as states, where $Q_1, \ldots, Q_m$ are the sets of states of $E_1, \ldots, E_m$, and
- for every $i \in \{1, \ldots, m\}$, every state $\langle q_1, q_2, \ldots, q_m \rangle \in Q$ and every transition $q_i \xrightarrow[a]{g} q_i'$ of E_i, a transition

$$\langle q_1, \ldots, q_{i-1}, q_i, q_{i+1}, \ldots, q_m \rangle \xrightarrow[a]{g} \langle q_1, \ldots, q_{i-1}, q_i', q_{i+1}, \ldots, q_m \rangle$$

The reachability problem

Let c, c' be two configurations of an extended automaton E. We say that c' is reachable from c if there is a path in $\mathcal{T}_E$ leading from c to c'.

We consider the following problem:

- Given: An extended automaton E, a set I of initial configurations,
  a set D of dangerous configurations.
- Decide: Is some dangerous configuration reachable
  from some initial configuration ?

The sets I and D may be infinite.

Symbolic search

A general framework for the reachability problem

Let $post(C)$ denote the immediate successors of a (possibly infinite!) set C of configurations

<u>Forward symbolic search</u>

Initialize $C := I$

Iterate $C := C \cup post(C)$ until

$C \cap D \neq \emptyset$; return “reachable”, or

a fixpoint is reached; return “non-reachable”

Backward search: exchange I and D, replace $post$ by pre.

Question: when is symbolic search effective?

(Forward) Symbolic search effective if . . .

1. each $C \in \mathcal{C}$ has a symbolic finite representation,

2. $I \in \mathcal{C}$,

3. if $C \in \mathcal{C}$, then $C \cup post(C) \in \mathcal{C}$ (and effectively computable),

4. emptiness of $C \cap D$ is decidable,

5. $C_1 = C_2$ is decidable (to check if fixpoint has been reached),, and

6. any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ reaches a fixpoint after finitely many steps.

Remarks

Similar conditions for backward search.

The shape of I is determined by the model.

The shape of D is determined by the specification.

This asymmetry can make one of the two searches far more useful than the other.

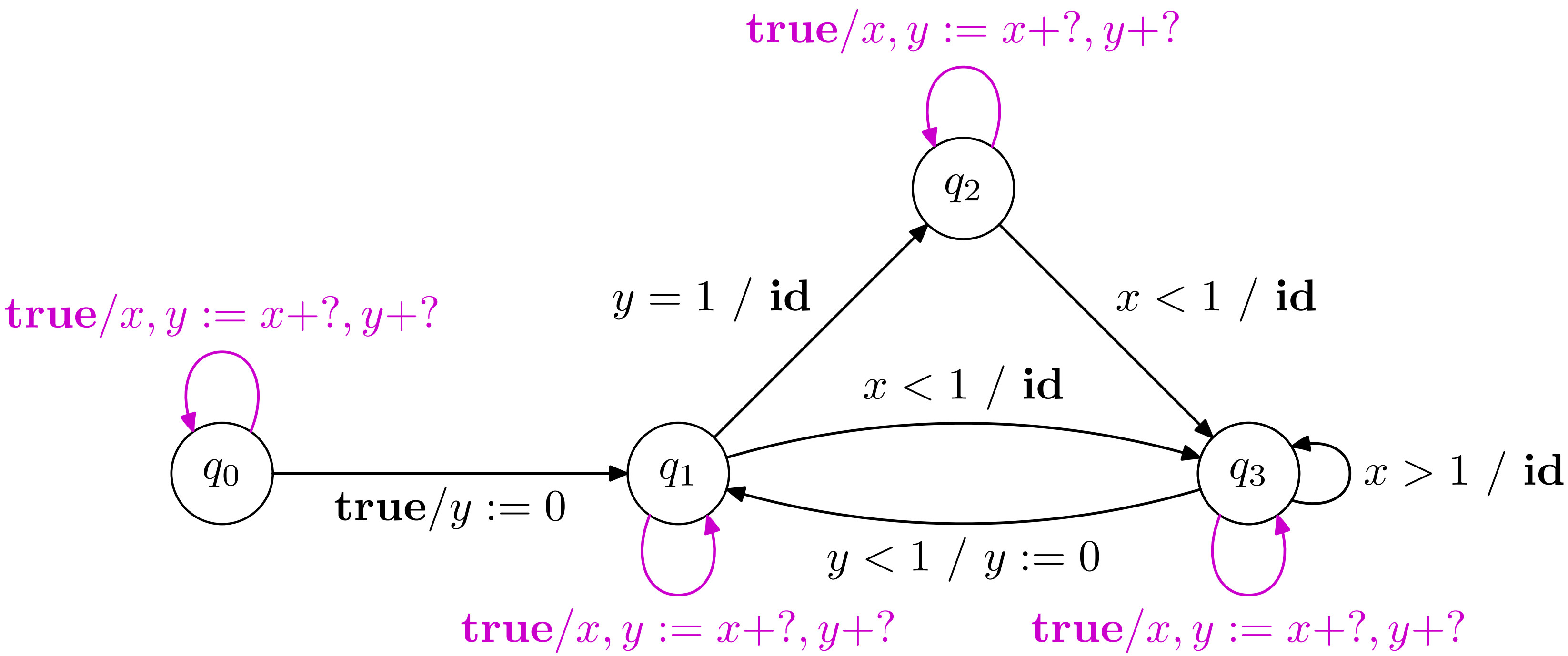
Program for the rest of the course

We consider four classes of systems, and use them to illustrate four different techniques to obtain an effective symbolic search.

- Timed automata: Finite partitions.
- Broadcast protocols: Well quasi-orders.
- Pushdown automata: Accelerations.
- (Lossy) channel systems: Learning.

Timed automata

Timed automata

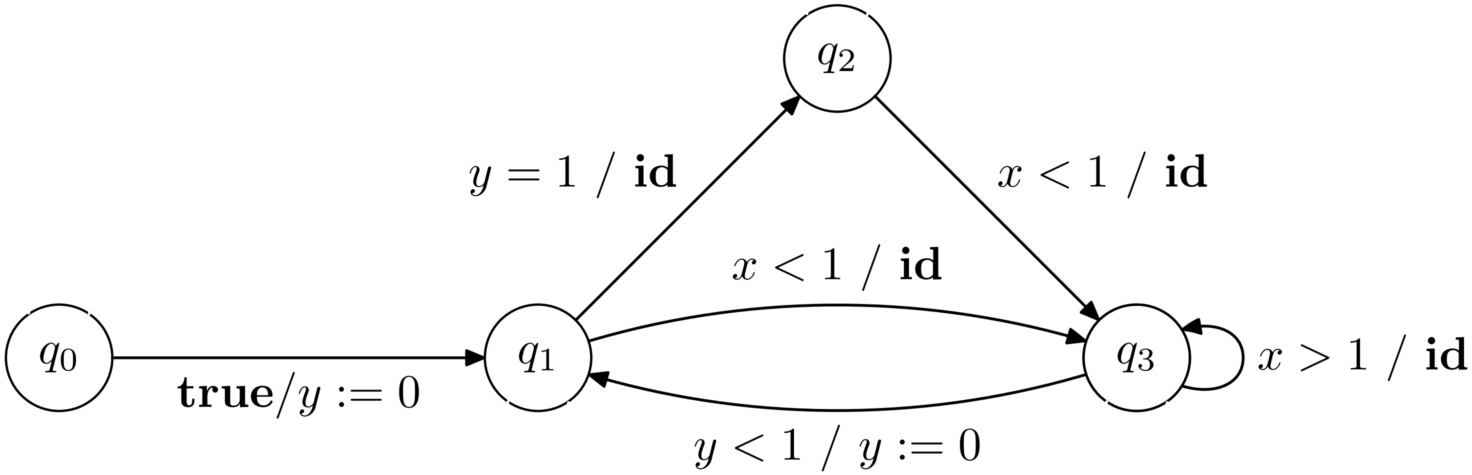


Automata extended with clocks (non-negative real variables).

Time-elapse transitions: self-loops, no guard, the action adds an arbitrary positive real to all clocks (same for all).

Location-switch transitions: guarded by boolean combination of comparisons with integer bounds, the action resets a subset of clocks.

Timed automata



Automata extended with clocks (non-negative real variables).

Time-elapse transitions: self-loops, no guard, the action adds an arbitrary positive real to all clocks (same for all).

Location-switch transitions: guarded by boolean combination of comparisons with integer bounds, the action resets a subset of clocks.

Case study: Fischer's mutex protocol

A simplified version (so that the analysis can be visualized in one slide).

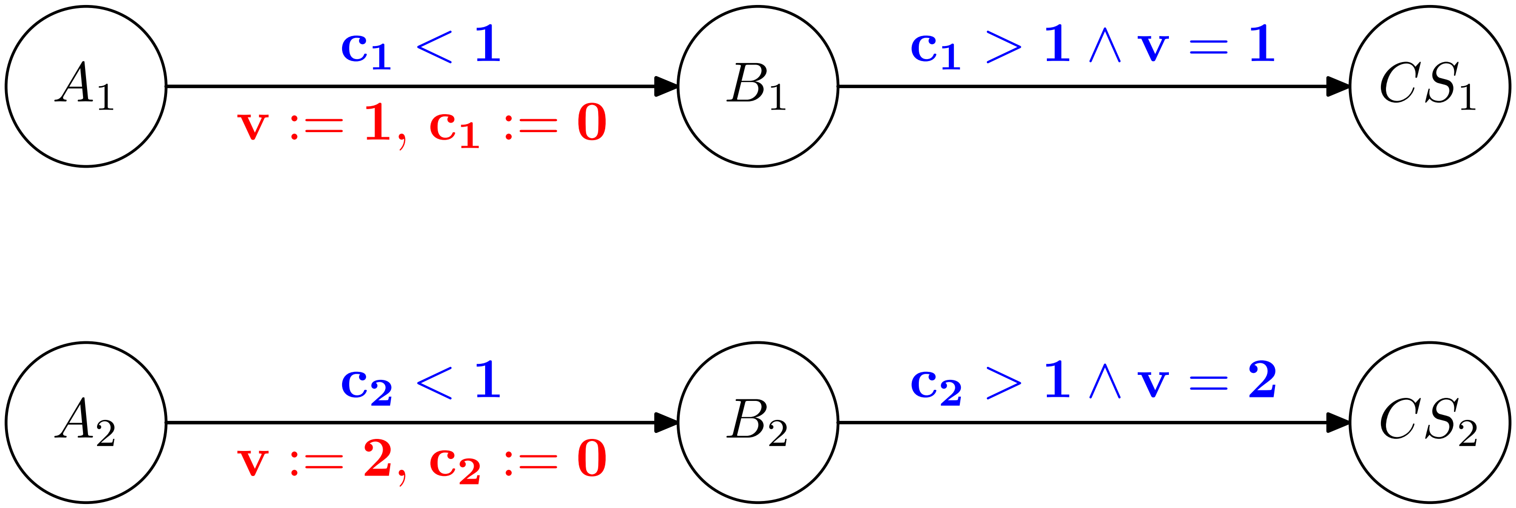
var `v:{1,2}` **init** `1;`

delay `< 1;`	**delay** `< 1;`
`v:= 1;`	`v:= 2;`
delay `> 1;`	**delay** `> 1;`
if `v = 1` **then goto** `cs1`	**if** `v = 2` **then goto** `cs2`

Model

$\mathbf{var}\ v : \{1, 2\}\ \mathbf{init}\ 1$

$\mathbf{var}\ c_1, c_2 : \text{clock}\ \mathbf{init}\ 0$



Network of 2 timed automata.

Equivalent to one single automaton with 9 states.

Symbolic search for timed automata

The set I of initial configurations is usually of the form

$$\{\langle q, 0, \ldots, 0\rangle \mid q \in Q_I\}$$

The set D of dangerous final configurations is usually of the form

$$\{\langle q, t_1, \ldots, t_n\rangle \mid q \in Q_D \text{ and } t_1, \ldots, t_n \geq 0\}$$

Question: Is reachability decidable for I and D of this form?

Regions

Consider a timed automaton with clocks $x_1, \ldots, x_n$.

Let max be the maximal constant appearing in the syntactic description of the automaton

Regions

Consider a timed automaton with clocks $x_1, \ldots, x_n$.

Let max be the maximal constant appearing in the syntactic description of the automaton

Let Γ be the set of all constraints of the form

$$x_i \leq k \qquad \text{or} \qquad x_i \geq k \qquad \text{or} \qquad x_i - x_j \leq k$$

where $k \in \{0, 1, \ldots, max\}$.

Regions

Consider a timed automaton with clocks $x_1, \ldots, x_n$.

Let max be the maximal constant appearing in the syntactic description of the automaton

Let Γ be the set of all constraints of the form

$$x_i \le k \qquad \text{or} \qquad x_i \ge k \qquad \text{or} \qquad x_i - x_j \le k$$

where $k \in \{0, 1, \ldots, max\}$.

Two configurations $\langle q, \mathbf{t} \rangle$ and $\langle r, \mathbf{u} \rangle$ are equivalent, denoted by $\langle q, \mathbf{t} \rangle \sim \langle r, \mathbf{u} \rangle$, if

- $q = r$, and
- for every constraint $\gamma \in \Gamma$: $\mathbf{t}$ satisfies γ iff $\mathbf{u}$ satisfies γ.

Regions

Consider a timed automaton with clocks $x_1, \ldots, x_n$.

Let max be the maximal constant appearing in the syntactic description of the automaton

Let Γ be the set of all constraints of the form

$$x_i \leq k \qquad \text{or} \qquad x_i \geq k \qquad \text{or} \qquad x_i - x_j \leq k$$

where $k \in \{0, 1, \ldots, max\}$.

Two configurations $\langle q, \mathbf{t} \rangle$ and $\langle r, \mathbf{u} \rangle$ are equivalent, denoted by $\langle q, \mathbf{t} \rangle \sim \langle r, \mathbf{u} \rangle$, if

- $q = r$, and
- for every constraint $\gamma \in \Gamma$: $\mathbf{t}$ satisfies γ iff $\mathbf{u}$ satisfies γ.

An equivalence class of configurations is called a region.

Characterizing regions

Given a real number z, let $\lfloor z \rfloor$ denote its integer and $\underline{z}$ its fractional part.

$\langle q, \mathbf{t} \rangle \sim \langle r, \mathbf{u} \rangle$ holds iff $q = r$ and for every $i, j \in \{0, 1, \ldots, max\}$:

(a) $\lfloor t_i \rfloor = \lfloor u_i \rfloor$ or $t_i > max$ and $u_i > max$,

(because $k - 1 \leq t_i \leq k$ iff $k - 1 \leq u_i \leq k$ for all $k \in \{1, \ldots, max\}$)

(b) if $t_i, u_i \leq max$, then $\underline{t_i} = 0$ iff $\underline{u_i} = 0$,

(because $k \leq t_i \leq k$ iff $k \leq u_i \leq k$ for all $k \in \{0, \ldots, max\}$))

(c) if $t_i, u_i, t_j, u_j \leq max$, then $\underline{t_i} < \underline{t_j}$ iff $\underline{u_i} < \underline{u_j}$.

(because of (a), (b), and $t_i - t_j \leq 0$ iff $u_i - u_j \leq 0$)

Example: $\langle q \quad 3.2 \quad 4.7 \quad 3.5 \rangle \sim \langle q \quad 3.7 \quad 4.9 \quad 3.8 \rangle$

$\langle q \quad 3.2 \quad 4.7 \quad 3.5 \rangle \not\sim \langle q \quad 3.2 \quad 4.7 \quad 3.9 \rangle$

Two observations

The number of regions is bounded by $(2max + 2)^n \cdot n! \cdot 2^n$ (exercise).

- Exponential in both the number of clocks n and in the maximal constant max when written in binary.

Two observations

The number of regions is bounded by $(2max + 2)^n \cdot n! \cdot 2^n$ (exercise).

- Exponential in both the number of clocks n and in the maximal constant max when written in binary.

Two equivalent configurations enable exactly the same transitions.

- Because they satisfy exactly the same guards.

Effectiveness of forward and backward search

We choose $\mathcal{C}$ as the powerset of the set of regions.

Theorem [Alur, Dill, TCS 1994]:
Both forward and backward search satisfy conditions (1) - (6).

Proof for forward search in the next slides, for backward search analogous.

Proof

1. A region can be finitely represented by
   the set of constraints it satisfies (by definition).

Proof

1. A region can be finitely represented by
   the set of constraints it satisfies (by definition). $\surd$

Proof

1. A region can be finitely represented by
   the set of constraints it satisfies (by definition). √

2. The set *I* of initial configurations is a union of regions.

Proof

1. A region can be finitely represented by
   the set of constraints it satisfies (by definition). $\surd$

2. The set I of initial configurations is a union of regions.

   $(0, \ldots, 0)$ is the only time-vector satisfying $x_i \leq 0$ for $i \in \{1, \ldots, n\}$,
   and so $\{\langle q, 0, \ldots, 0\rangle\}$ is a region for each state q.

Proof

1. A region can be finitely represented by
   the set of constraints it satisfies (by definition). $\surd$

2. The set I of initial configurations is a union of regions. $\surd$

   $(0, \ldots, 0)$ is the only time-vector satisfying $x_i \leq 0$ for $i \in \{1, \ldots, n\}$,
   and so $\{\langle q, 0, \ldots, 0\rangle\}$ is a region for each state q.

Proof

1. A region can be finitely represented by

   the set of constraints it satisfies (by definition).

2. The set I of initial configurations is a union of regions.

   $(0,\ldots,0)$ is the only time-vector satisfying $x_i \leq 0$ for $i \in \{1,\ldots,n\}$, and so $\{\langle q, 0, \ldots, 0\rangle\}$ is a region for each state q.

3. If C is the union of a set of regions, then so is $C \cup post(C)$.

Proof

1. A region can be finitely represented by
   the set of constraints it satisfies (by definition). $\surd$

2. The set I of initial configurations is a union of regions. $\surd$

   $(0, \ldots, 0)$ is the only time-vector satisfying $x_i \leq 0$ for $i \in \{1, \ldots, n\}$, and so $\{\langle q, 0, \ldots, 0\rangle\}$ is a region for each state q.

3. If C is the union of a set of regions, then so is $C \cup post(C)$.

   It suffices to prove that if C is a region then $post(C)$ is a union of regions.

Proof

1. A region can be finitely represented by

   the set of constraints it satisfies (by definition). $\surd$

2. The set I of initial configurations is a union of regions. $\surd$

   $(0, \ldots, 0)$ is the only time-vector satisfying $x_i \leq 0$ for $i \in \{1, \ldots, n\}$, and so $\{\langle q, 0, \ldots, 0\rangle\}$ is a region for each state q.

3. If C is the union of a set of regions, then so is $C \cup post(C)$.

   It suffices to prove that if C is a region then $post(C)$ is a union of regions. Take $\langle r, \mathbf{u}\rangle \in post(C)$ and $\langle r, \mathbf{u}'\rangle \sim \langle r, \mathbf{u}\rangle$. We show $\langle r, \mathbf{u}'\rangle \in post(C)$.

Proof

1. A region can be finitely represented by
   the set of constraints it satisfies (by definition). $\surd$

2. The set I of initial configurations is a union of regions. $\surd$

$(0, \ldots, 0)$ is the only time-vector satisfying $x_i \leq 0$ for $i \in \{1, \ldots, n\}$, and so $\{\langle q, 0, \ldots, 0\rangle\}$ is a region for each state q.

3. If C is the union of a set of regions, then so is $C \cup post(C)$.

It suffices to prove that if C is a region then $post(C)$ is a union of regions. Take $\langle r, \mathbf{u}\rangle \in post(C)$ and $\langle r, \mathbf{u}'\rangle \sim \langle r, \mathbf{u}\rangle$. We show $\langle r, \mathbf{u}'\rangle \in post(C)$. Since $\langle r, \mathbf{u}\rangle \in post(C)$, there is $\langle q, \mathbf{t}\rangle \in C$ such that $\langle q, \mathbf{t}\rangle \longrightarrow \langle r, \mathbf{u}\rangle$. We consider the cases of time-elapse and location-switch transitions separately.

Time-elapse transitions ("proof by example"):

$$\langle \; q \quad t_1 \quad t_2 \quad t_3 \; \rangle \xrightarrow{\lfloor\tau\rfloor + \underline{\tau}} \langle \; r \quad u_1 \quad u_2 \quad u_3 \; \rangle$$

$$\sim$$

$$\langle \; r \quad u_1' \quad u_2' \quad u_3' \; \rangle$$

Time-elapse transitions ("proof by example"):

$$\begin{array}{ccc} 0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 & & 0 = \underline{u_2} < \underline{u_3} < \underline{u_1} < 1 \\ \langle\ q \quad t_1 \quad t_2 \quad t_3\ \rangle & \xrightarrow{\lfloor\tau\rfloor + \underline{\tau}} & \langle\ r \quad u_1 \quad u_2 \quad u_3\ \rangle \\ & & \sim \\ & & \langle\ r \quad u'_1 \quad u'_2 \quad u'_3\ \rangle \end{array}$$

Time-elapse transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_3} < \underline{u_1} < 1$$

$$\langle\ q \quad t_1 \quad t_2 \quad t_3\ \rangle \xrightarrow{\lfloor\tau\rfloor + \underline{\tau}} \langle\ r \quad u_1 \quad u_2 \quad u_3\ \rangle$$

$$\langle\ q \quad 3.1 \quad 1.5 \quad 2.7\ \rangle \xrightarrow{1 + 0.5} \langle\ r \quad 4.7 \quad 3.0 \quad 4.3\ \rangle$$

$$\sim$$

$$\langle\ r \quad u_1' \quad u_2' \quad u_3'\ \rangle$$

Time-elapse transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_3} < \underline{u_1} < 1$$

$$\langle\; q \quad t_1 \quad t_2 \quad t_3 \;\rangle \xrightarrow{\lfloor\tau\rfloor + \underline{\tau}} \langle\; r \quad u_1 \quad u_2 \quad u_3 \;\rangle$$

$$\langle\; q \quad 3.1 \quad 1.5 \quad 2.7 \;\rangle \xrightarrow{1 + 0.5} \langle\; r \quad 4.7 \quad 3.0 \quad 4.3 \;\rangle$$

$$\sim$$

$$\langle\; r \quad u'_1 \quad u'_2 \quad u'_3 \;\rangle$$

$$0 = \underline{u'_2} < \underline{u'_3} < \underline{u'_1} < 1$$

Time-elapse transitions (“proof by example”):

$$
\begin{array}{ccc}
\begin{array}{c} 0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \\ \langle\ q \quad t_1 \quad t_2 \quad t_3\ \rangle \\ \langle\ q \quad 3.1 \quad 1.5 \quad 2.7\ \rangle \end{array}
&
\begin{array}{c} \xrightarrow{\lfloor \tau \rfloor + \underline{\tau}} \\ \xrightarrow{1 + 0.5} \end{array}
&
\begin{array}{c} 0 = \underline{u_2} < \underline{u_3} < \underline{u_1} < 1 \\ \langle\ r \quad u_1 \quad u_2 \quad u_3\ \rangle \\ \langle\ r \quad 4.7 \quad 3.0 \quad 4.3\ \rangle \\ \sim \\ \langle\ r \quad 4.8 \quad 3.0 \quad 4.7\ \rangle \\ \langle\ r \quad u'_1 \quad u'_2 \quad u'_3\ \rangle \\ 0 = \underline{u'_2} < \underline{u'_3} < \underline{u'_1} < 1 \end{array}
\end{array}
$$

Time-elapse transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_3} < \underline{u_1} < 1$$

$$\langle\ q \quad t_1 \quad t_2 \quad t_3\ \rangle \xrightarrow{\lfloor\tau\rfloor + \underline{\tau}} \langle\ r \quad u_1 \quad u_2 \quad u_3\ \rangle$$

$$\langle\ q \quad 3.1 \quad 1.5 \quad 2.7\ \rangle \xrightarrow{1 + 0.5} \langle\ r \quad 4.7 \quad 3.0 \quad 4.3\ \rangle$$

$$\sim \qquad\qquad\qquad\qquad \sim$$

$$\langle\ r \quad 4.8 \quad 3.0 \quad 4.7\ \rangle$$

$$\langle\ q \quad t'_1 \quad t'_2 \quad t'_3\ \rangle \xrightarrow{\lfloor\tau\rfloor + \delta} \langle\ r \quad u'_1 \quad u'_2 \quad u'_3\ \rangle$$

$$0 < \underline{t'_1} < \underline{t'_2} < \underline{t'_3} < 1 \qquad \underline{u'_3} < \delta < \underline{u'_1} \qquad 0 = \underline{u'_2} < \underline{u'_3} < \underline{u'_1} < 1$$

Time-elapse transitions ("proof by example"):

$$
\begin{array}{ccc}
0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 & & 0 = \underline{u_2} < \underline{u_3} < \underline{u_1} < 1 \\
\langle\ q \quad t_1 \quad t_2 \quad t_3\ \rangle & \xrightarrow{\lfloor \tau \rfloor + \underline{\tau}} & \langle\ r \quad u_1 \quad u_2 \quad u_3\ \rangle \\
\langle\ q \quad 3.1 \quad 1.5 \quad 2.7\ \rangle & \xrightarrow{1 + 0.5} & \langle\ r \quad 4.7 \quad 3.0 \quad 4.3\ \rangle \\
\sim & & \sim \\
\langle\ q \quad 3.05 \quad 1.25 \quad 2.95\ \rangle & \xrightarrow{1 + 0.75} & \langle\ r \quad 4.8 \quad 3.0 \quad 4.7\ \rangle \\
\langle\ q \quad t_1' \quad t_2' \quad t_3'\ \rangle & \xrightarrow{\lfloor \tau \rfloor + \delta} & \langle\ r \quad u_1' \quad u_2' \quad u_3'\ \rangle \\
0 < \underline{t_1'} < \underline{t_2'} < \underline{t_3'} < 1 & \underline{u_3'} < \delta < \underline{u_1'} & 0 = \underline{u_2'} < \underline{u_3'} < \underline{u_1'} < 1
\end{array}
$$

Time-elapse transitions ("proof by example"): $\surd$

$$
\begin{array}{ccccccccccccc}
 & \multicolumn{5}{c}{0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1} & & & \multicolumn{5}{c}{0 = \underline{u_2} < \underline{u_3} < \underline{u_1} < 1} \\
\langle & q & t_1 & t_2 & t_3 & \rangle & \xrightarrow{\lfloor \tau \rfloor + \underline{\tau}} & \langle & r & u_1 & u_2 & u_3 & \rangle \\
\langle & q & 3.1 & 1.5 & 2.7 & \rangle & \xrightarrow{1 + 0.5} & \langle & r & 4.7 & 3.0 & 4.3 & \rangle \\
 & & & \sim & & & & & & & \sim & & \\
\langle & q & 3.05 & 1.25 & 2.95 & \rangle & \xrightarrow{1 + 0.75} & \langle & r & 4.8 & 3.0 & 4.7 & \rangle \\
\langle & q & t'_1 & t'_2 & t'_3 & \rangle & \xrightarrow{\lfloor \tau \rfloor + \delta} & \langle & r & u'_1 & u'_2 & u'_3 & \rangle \\
 & \multicolumn{5}{c}{0 < \underline{t'_1} < \underline{t'_2} < \underline{t'_3} < 1} & \underline{u'_3} < \delta < \underline{u'_1} & & \multicolumn{5}{c}{0 = \underline{u'_2} < \underline{u'_3} < \underline{u'_1} < 1}
\end{array}
$$

Location-switch transitions ("proof by example"):

$$\langle \; r \quad t_1 \quad t_2 \quad t_3 \; \rangle \xrightarrow{x_2 := 0} \langle \; r \quad t_1 \quad 0 \quad t_3 \; \rangle$$

$$\sim$$

$$\langle \; r \quad u'_1 \quad 0 \quad u'_3 \; \rangle$$

Location-switch transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_1} < \underline{u_3} < 1$$

$$\langle \; r \quad t_1 \quad t_2 \quad t_3 \; \rangle \xrightarrow{x_2 := 0} \langle \; r \quad t_1 \quad 0 \quad t_3 \; \rangle$$

$$\sim$$

$$\langle \; r \quad u_1' \quad 0 \quad u_3' \; \rangle$$

Location-switch transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_1} < \underline{u_3} < 1$$

$$\langle \; r \quad t_1 \quad t_2 \quad t_3 \; \rangle \xrightarrow{x_2 := 0} \langle \; r \quad t_1 \quad 0 \quad t_3 \; \rangle$$

$$\langle \; r \quad 3.1 \quad 1.5 \quad 2.7 \; \rangle \xrightarrow{x_2 := 0} \langle \; r \quad 3.1 \quad 0 \quad 2.7 \; \rangle$$

$$\sim$$

$$\langle \; r \quad u'_1 \quad 0 \quad u'_3 \; \rangle$$

Location-switch transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_1} < \underline{u_3} < 1$$

$$\langle\ r \quad t_1 \quad t_2 \quad t_3\ \rangle \xrightarrow{x_2 := 0} \langle\ r \quad t_1 \quad 0 \quad t_3\ \rangle$$

$$\langle\ r \quad 3.1 \quad 1.5 \quad 2.7\ \rangle \xrightarrow{x_2 := 0} \langle\ r \quad 3.1 \quad 0 \quad 2.7\ \rangle$$

$$\sim$$

$$\langle\ r \quad u'_1 \quad 0 \quad u'_3\ \rangle$$

$$0 = \underline{u'_2} < \underline{u'_1} < \underline{u'_3} < 1$$

Location-switch transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_1} < \underline{u_3} < 1$$

$$\langle \; r \quad t_1 \quad t_2 \quad t_3 \; \rangle \xrightarrow{x_2 := 0} \langle \; r \quad t_1 \quad 0 \quad t_3 \; \rangle$$

$$\langle \; r \quad 3.1 \quad 1.5 \quad 2.7 \; \rangle \xrightarrow{x_2 := 0} \langle \; r \quad 3.1 \quad 0 \quad 2.7 \; \rangle$$

$$\sim$$

$$\langle \; r \quad 3.3 \quad 0 \quad 2.4 \; \rangle$$

$$\langle \; r \quad u'_1 \quad 0 \quad u'_3 \; \rangle$$

$$0 = \underline{u'_2} < \underline{u'_1} < \underline{u'_3} < 1$$

Location-switch transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_1} < \underline{u_3} < 1$$

$$\langle\ r\quad t_1\quad t_2\quad t_3\ \rangle \xrightarrow{x_2 := 0} \langle\ r\quad t_1\quad 0\quad t_3\ \rangle$$

$$\langle\ r\quad 3.1\quad 1.5\quad 2.7\ \rangle \xrightarrow{x_2 := 0} \langle\ r\quad 3.1\quad 0\quad 2.7\ \rangle$$

$$\sim \qquad\qquad\qquad\qquad \sim$$

$$\langle\ r\quad 3.3\quad 0\quad 2.4\ \rangle$$

$$\langle\ r\quad t'_1\quad t'_2\quad t'_3\ \rangle \xrightarrow{x_2 := 0} \langle\ r\quad u'_1\quad 0\quad u'_3\ \rangle$$

$$0 < \underline{t'_1} = \underline{u'_1} < \underline{t'_2} < \underline{t'_3} = \underline{u'_3} < 1 \qquad 0 = \underline{u'_2} < \underline{u'_1} < \underline{u'_3} < 1$$

Location-switch transitions ("proof by example"):

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_1} < \underline{u_3} < 1$$

$$\langle\ r \quad t_1 \quad t_2 \quad t_3\ \rangle \xrightarrow{x_2 := 0} \langle\ r \quad t_1 \quad 0 \quad t_3\ \rangle$$

$$\langle\ r \quad 3.1 \quad 1.5 \quad 2.7\ \rangle \xrightarrow{x_2 := 0} \langle\ r \quad 3.1 \quad 0 \quad 2.7\ \rangle$$

$$\sim \qquad\qquad\qquad\qquad \sim$$

$$\langle\ r \quad 3.3 \quad 1.35 \quad 2.4\ \rangle \xrightarrow{x_2 := 0} \langle\ r \quad 3.3 \quad 0 \quad 2.4\ \rangle$$

$$\langle\ r \quad t_1' \quad t_2' \quad t_3'\ \rangle \xrightarrow{x_2 := 0} \langle\ r \quad u_1' \quad 0 \quad u_3'\ \rangle$$

$$0 < \underline{t_1'} = \underline{u_1'} < \underline{t_2'} < \underline{t_3'} = \underline{u_3'} < 1 \qquad\qquad 0 = \underline{u_2'} < \underline{u_1'} < \underline{u_3'} < 1$$

Location-switch transitions ("proof by example"): $\surd$

$$0 < \underline{t_1} < \underline{t_2} < \underline{t_3} < 1 \qquad\qquad 0 = \underline{u_2} < \underline{u_1} < \underline{u_3} < 1$$

$$\langle\ r\quad t_1\quad t_2\quad t_3\ \rangle \xrightarrow{x_2 := 0} \langle\ r\quad t_1\quad 0\quad t_3\ \rangle$$

$$\langle\ r\quad 3.1\quad 1.5\quad 2.7\ \rangle \xrightarrow{x_2 := 0} \langle\ r\quad 3.1\quad 0\quad 2.7\ \rangle$$

$$\sim \qquad\qquad\qquad\qquad \sim$$

$$\langle\ r\quad 3.3\quad 1.35\quad 2.4\ \rangle \xrightarrow{x_2 := 0} \langle\ r\quad 3.3\quad 0\quad 2.4\ \rangle$$

$$\langle\ r\quad t_1'\quad t_2'\quad t_3'\ \rangle \xrightarrow{x_2 := 0} \langle\ r\quad u_1'\quad 0\quad u_3'\ \rangle$$

$$0 < \underline{t_1'} = \underline{u_1'} < \underline{t_2'} < \underline{t_3'} = \underline{u_3'} < 1 \qquad\qquad 0 = \underline{u_2'} < \underline{u_1'} < \underline{u_3'} < 1$$

4. Emptiness of $C \cap D$ is decidable.

4. Emptiness of $C \cap D$ is decidable.

Just check if C contains some configuration with some state of Q_D as first element.

4. Emptiness of $C \cap D$ is decidable. $\surd$

Just check if C contains some configuration with some state of Q_D as first element.

4. Emptiness of $C \cap D$ is decidable. $\surd$

Just check if C contains some configuration with some state of Q_D as first element.

5. $C_1 = C_2$ is decidable.

4. Emptiness of $C \cap D$ is decidable. $\surd$

Just check if C contains some configuration with some state of Q_D as first element.

5. $C_1 = C_2$ is decidable.

A region is represented by the constraints it satisfies.
Two regions are equal iff their representations are equal.
Two sets of regions are equal iff they contain the same regions.

4. Emptiness of $C \cap D$ is decidable. $\surd$

Just check if C contains some configuration with some state of Q_D as first element.

5. $C_1 = C_2$ is decidable. $\surd$

A region is represented by the constraints it satisfies.
Two regions are equal iff their representations are equal.
Two sets of regions are equal iff they contain the same regions.

4. Emptiness of $C \cap D$ is decidable. $\surd$

Just check if C contains some configuration with some state of Q_D as first element.

5. $C_1 = C_2$ is decidable. $\surd$

A region is represented by the constraints it satisfies.
Two regions are equal iff their representations are equal.
Two sets of regions are equal iff they contain the same regions.

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint.

4. Emptiness of $C \cap D$ is decidable. $\surd$

Just check if C contains some configuration with some state of Q_D as first element.

5. $C_1 = C_2$ is decidable. $\surd$

A region is represented by the constraints it satisfies.
Two regions are equal iff their representations are equal.
Two sets of regions are equal iff they contain the same regions.

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint.

Follows from the fact that the set of regions is finite.

4. Emptiness of $C \cap D$ is decidable. $\surd$

Just check if C contains some configuration with some state of Q_D as first element.

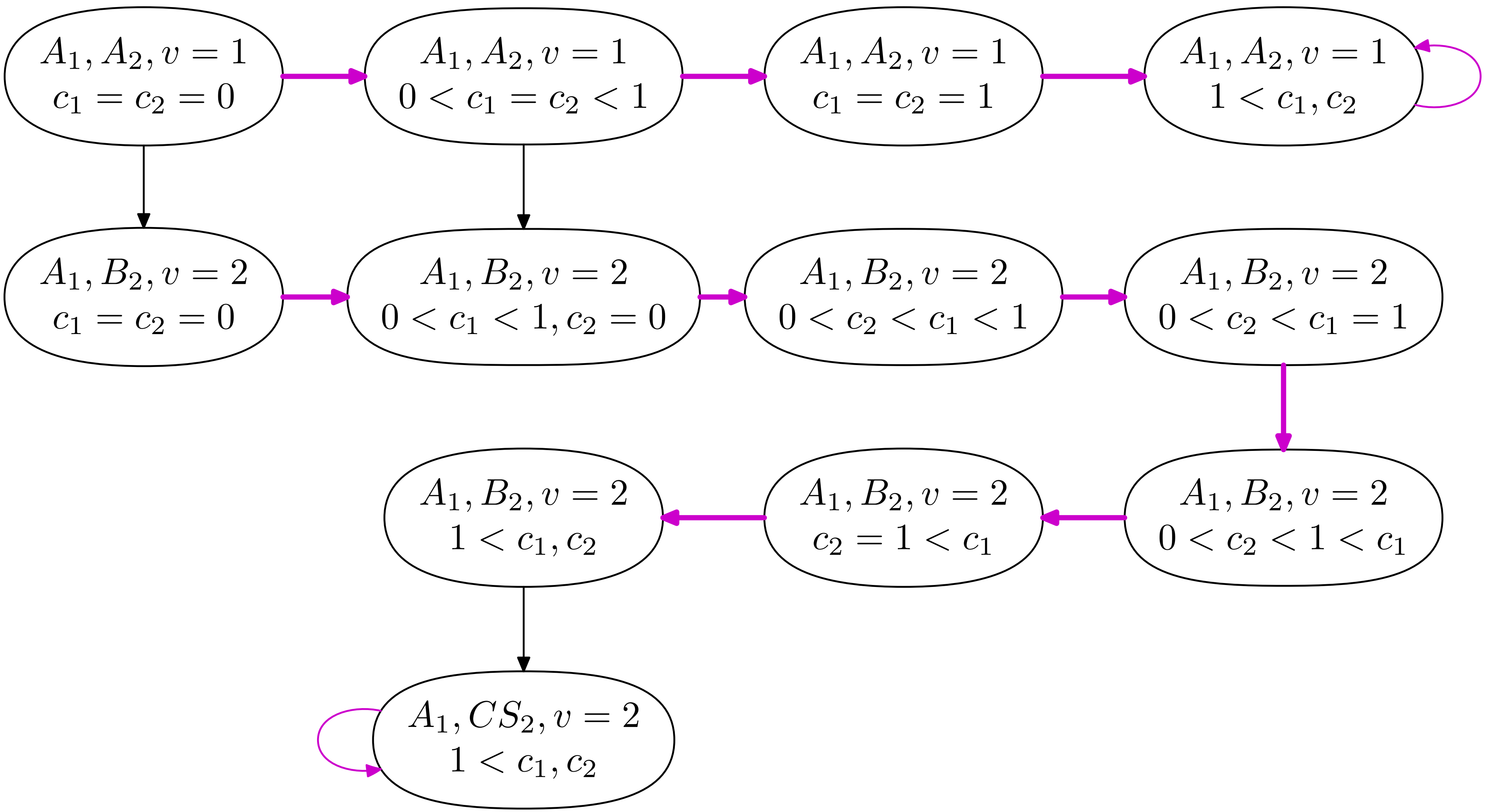
5. $C_1 = C_2$ is decidable. $\surd$

A region is represented by the constraints it satisfies.
Two regions are equal iff their representations are equal.
Two sets of regions are equal iff they contain the same regions.

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint. $\surd$

Follows from the fact that the set of regions is finite.

(One half of) The region graph of Fischer's protocol



Complexity of the reachability problem

The reachability problem is PSPACE-complete.

Reason: exponential dependence in the number of clocks or the size of max is unavoidable.

The problem remains PSPACE-hard if the constants or the number of clocks (but not both) are bounded.

Repeated reachability for timed automata

A control state is repeatedly reachable if some non-zeno infinite execution containing infinitely many location-switch transitions visits the control state infinitely often.

The repeated reachability problem can be solved easily using the region graph.

To know more

Tutorial slides by Rajeev Alur, available at
http://www.cis.upenn.edu/ alur/talks.html

Check the publications of: Alur, Asarin, Bouyer, Courcoubetis, Dill, Henzinger, Laroussinie, Larsen, Maler, Sifakis, Wilke . . . .

UPPAAL is a popular tool for verification of timed automata,
http://www.uppaal.com/

Broadcast protocols

Broadcast protocols

Introduced by Emerson and Namjoshi in LICS '98.

All processes execute the same algorithm, i.e., all finite automata are identical.

Processes are indistinguishable (no IDs).

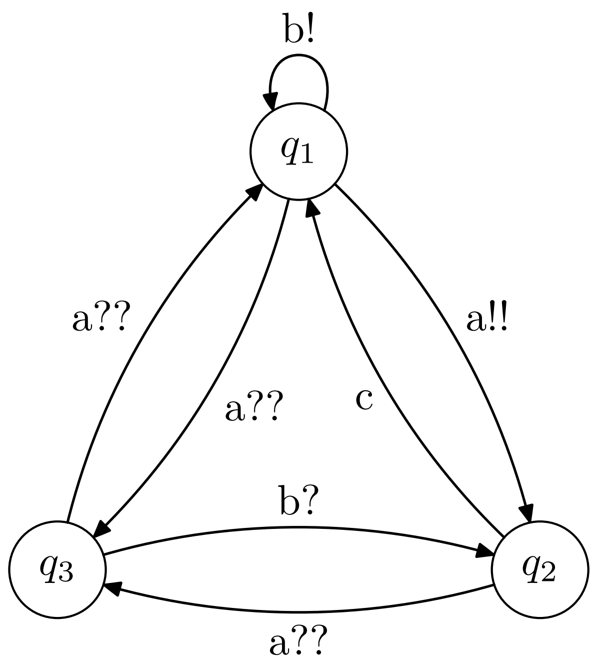
Communication mechanisms:

Rendezvous: two processes exchange a message and move to new states.

Broadcasts: a process sends a message to all others,
all processes move to new states.

We introduce syntax and semantics and show translation into extended automata.

Syntax



a!!	:	broadcast a message along (channel) *a*
a??	:	receive a broadcasted message along *a*
b!	:	send a message to one process along *b*
b?	:	receive a message from one process along *b*
c	:	change state without communicating with anybody

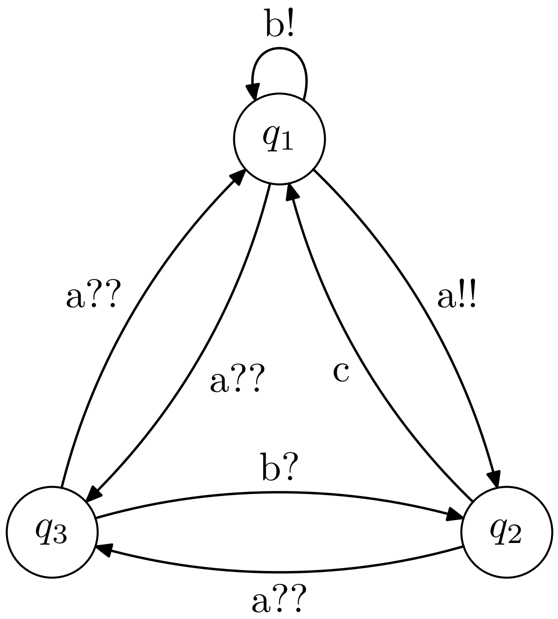
Semantics

The global state of a broadcast protocol is completely determined by the number of processes in each state.

Configuration: mapping $c\colon Q \longrightarrow \mathbb{N}$

represented by the vector $(c(q_1), \ldots, c(q_n))$.

Semantics for a given initial configuration: finite transition system with configurations as nodes.



$$
\begin{array}{rcll}
(3,1,2) & \longrightarrow & (4,0,2) & \text{(silent move } c\text{)} \\
(3,1,2) & \longrightarrow & (3,2,1) & \text{(rendezvous } b\text{)} \\
(3,1,2) & \longrightarrow & (2,1,3) & \text{(broadcast } a\text{)} \\[1em]
(185,3425,17) & \longrightarrow & (17,1,3609) & \text{(broadcast } a\text{)}
\end{array}
$$

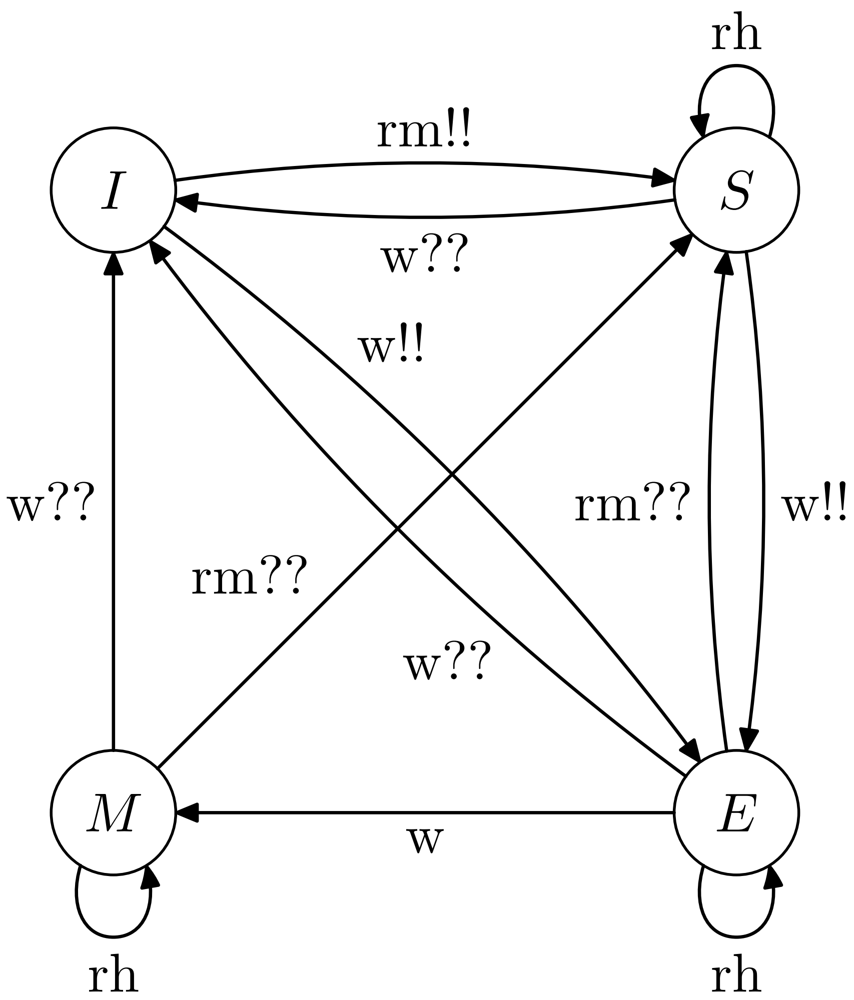
Parametrized configuration: partial mapping $p : Q \longrightarrow \mathbb{N}$.

- Intuition: “configuration with holes”.
- Formally: set of configurations (total mappings matching p).

Infinite transition system of the broadcast protocol:

- Fix an initial parametrized configuration p_0.
- Take the union of all finite transition systems for each configuration $c \in p_0$.

Case study: A MESI cache-coherence protocol



rh	:	read hit
rm	:	read miss
w	:	write hit/write miss

Broadcast protocols as extended automata

We translate the MESI-protocol into an extended automaton.

We take:

- One (non-negative) integer variable per state of the protocol: m, e, s, i.
- One single control state q.
- One transition $q \xrightarrow[a]{g} q$ for each send transition or silent move of the protocol, see next slide.

A configuration $(n_1, \ldots, n_k)$ of a broadcast protocol corresponds to the configuration $\langle q, n_1, \ldots, n_k \rangle$ of the extended automaton.

Transition	Guard	Action
$I \xrightarrow{rm!!} S$	$i \geq 1$	$m' = m \quad e' = 0 \quad s' = m + e + s + 1 \quad i' = i - 1$
$I \xrightarrow{w!!} E$	$i \geq 1$	$m' = 0 \quad e' = 1 \quad s' = 0 \quad i' = m + e + s + i - 1$
$S \xrightarrow{w!!} E$	$s \geq 1$	$m' = 0 \quad e' = 1 \quad s' = 0 \quad i' = m + e + s + i - 1$
$S \xrightarrow{rh} S$	$s \geq 1$	$m' = m \quad e' = e \quad s' = s \quad i' = i$
$E \xrightarrow{w} M$	$e \geq 1$	$m' = m + 1 \quad e' = e - 1 \quad s' = s \quad i' = i$
$E \xrightarrow{rh} E$	$e \geq 1$	$m' = m \quad e' = e \quad s' = s \quad i' = i$
$M \xrightarrow{rh} M$	$m \geq 1$	$m' = m \quad e' = e \quad s' = s \quad i' = i$

Reachability in broadcast protocols

Typical set I of initial configurations: parametrized configuration.

Typical set D of final configurations: upward-closed sets.

- U is an upward-closed set of configurations if

$$c \in U \text{ and } c' \geq c \text{ implies } c' \in U$$

  where $\geq$ is the pointwise order on $\mathbb{N}^n$.

- Example: states M and S of MESI protocol should be mutually exclusive

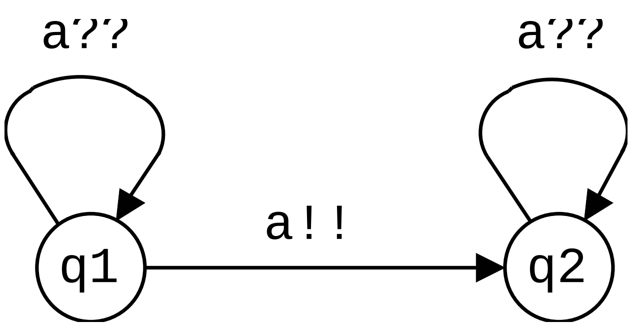
$$D = \{(m, e, s, i) \mid m \geq 1 \wedge s \geq 1\}$$

Question: Is reachability decidable if I is a parametric configuration and D is an upward-closed set?

First try: Forward search

Since $I \in \mathcal{C}$ is required by condition (2), the family $\mathcal{C}$ must contain all parametrized configurations.

Satisfies (1) - (5) but not (6). Termination fails in very simple cases.



$$(\sqcup, 0) \xrightarrow{a} (\sqcup, 1) \xrightarrow{a} (\sqcup, 2) \xrightarrow{a} \ldots$$

Second try: Backward search

Since $D \in \mathcal{C}$ is required by condition (2), the family $\mathcal{C}$ must contain all upward-closed sets.

Theorem [Abdulla *et al.*, I&C 160, 2000], [E. *et al.*, LICS'99]
Backward search satisfies conditions (1) - (6).

Proof in the next slides.

Proof

1. An upward-closed set can be finitely represented by its set of minimal elements w.r.t. the pointwise order $\leq$

Proof

1. An upward-closed set can be finitely represented by its set of minimal elements w.r.t. the pointwise order $\leq$

- An upward-closed set is determined by its minimal elements
- Any subset of $\mathbb{N}^k$ has finitely many minimal elements

Proof

1. An upward-closed set can be finitely represented by
   its set of minimal elements w.r.t. the pointwise order $\leq$

- An upward-closed set is determined by its minimal elements
- Any subset of $\mathbb{N}^k$ has finitely many minimal elements

  Every infinite sequence $c_1, c_2, c_3, \ldots$ of vectors of $\mathbb{N}^k$ contains a non-decreasing infinite subsequence $c_{i_1} \leq c_{i_2} \leq c_{i_3} \ldots$ (Dickson's lemma)

Proof

1. An upward-closed set can be finitely represented by its set of minimal elements w.r.t. the pointwise order $\leq$

- An upward-closed set is determined by its minimal elements
- Any subset of $\mathbb{N}^k$ has finitely many minimal elements

  Every infinite sequence $c_1, c_2, c_3, \ldots$ of vectors of $\mathbb{N}^k$ contains a non-decreasing infinite subsequence $c_{i_1} \leq c_{i_2} \leq c_{i_3} \ldots$ (Dickson's lemma)

  Assume some $X \subseteq \mathbb{N}^k$ has infinitely many minimal elements.
  Enumerate them in a sequence $m_1, m_2 \ldots$.

Proof

1. An upward-closed set can be finitely represented by

   its set of minimal elements w.r.t. the pointwise order $\leq$

- An upward-closed set is determined by its minimal elements
- Any subset of $\mathbb{N}^k$ has finitely many minimal elements

  Every infinite sequence $c_1, c_2, c_3, \ldots$ of vectors of $\mathbb{N}^k$ contains a non-decreasing infinite subsequence $c_{i_1} \leq c_{i_2} \leq c_{i_3} \ldots$ (Dickson's lemma)

  Assume some $X \subseteq \mathbb{N}^k$ has infinitely many minimal elements. Enumerate them in a sequence $m_1, m_2 \ldots$.

  By Dickson's lemma, $m_i \leq m_j$ for some $i < j$.

Proof

1. An upward-closed set can be finitely represented by its set of minimal elements w.r.t. the pointwise order $\leq$

- An upward-closed set is determined by its minimal elements
- Any subset of $\mathbb{N}^k$ has finitely many minimal elements

  Every infinite sequence $c_1, c_2, c_3, \ldots$ of vectors of $\mathbb{N}^k$ contains a non-decreasing infinite subsequence $c_{i_1} \leq c_{i_2} \leq c_{i_3} \ldots$ (Dickson's lemma)

  Assume some $X \subseteq \mathbb{N}^k$ has infinitely many minimal elements.
  Enumerate them in a sequence $m_1, m_2 \ldots$.

  By Dickson's lemma, $m_i \leq m_j$ for some $i < j$.

  But then m_j is not minimal.

Proof

1. An upward-closed set can be finitely represented by its set of minimal elements w.r.t. the pointwise order $\leq$

- An upward-closed set is determined by its minimal elements
- Any subset of $\mathbb{N}^k$ has finitely many minimal elements

Every infinite sequence $c_1, c_2, c_3, \ldots$ of vectors of $\mathbb{N}^k$ contains a non-decreasing infinite subsequence $c_{i_1} \leq c_{i_2} \leq c_{i_3} \ldots$ (Dickson's lemma)

Assume some $X \subseteq \mathbb{N}^k$ has infinitely many minimal elements. Enumerate them in a sequence $m_1, m_2 \ldots$.

By Dickson's lemma, $m_i \leq m_j$ for some $i < j$.

But then m_j is not minimal.

Contradiction.

Proof

1. An upward-closed set can be finitely represented by its set of minimal elements w.r.t. the pointwise order $\leq$ $\surd$

- An upward-closed set is determined by its minimal elements
- Any subset of $\mathbb{N}^k$ has finitely many minimal elements

Every infinite sequence $c_1, c_2, c_3, \ldots$ of vectors of $\mathbb{N}^k$ contains a non-decreasing infinite subsequence $c_{i_1} \leq c_{i_2} \leq c_{i_3} \ldots$ (Dickson's lemma)

Assume some $X \subseteq \mathbb{N}^k$ has infinitely many minimal elements.
Enumerate them in a sequence $m_1, m_2 \ldots$.

By Dickson's lemma, $m_i \leq m_j$ for some $i < j$.

But then m_j is not minimal.

Contradiction.

2. *D* is upward-closed √

2. *D* is upward-closed $\surd$

3. If C is upward-closed then so is $C \cup pre(C)$.

2. D is upward-closed $\checkmark$

3. If C is upward-closed then so is $C \cup pre(C)$.

Since union of upward-closed sets is upward-closed, it suffices to prove that $pre(C)$ is upward-closed.

2. D is upward-closed $\surd$

3. If C is upward-closed then so is $C \cup pre(C)$.

Since union of upward-closed sets is upward-closed, it suffices to prove that $pre(C)$ is upward-closed.
Take $c \in pre(C)$ and $c' \geq c$. We show $c' \in pre(C)$.
Key idea: “adding more processes to a configuration cannot disable any transition”.

2. D is upward-closed $\surd$

3. If C is upward-closed then so is $C \cup pre(C)$.

Since union of upward-closed sets is upward-closed, it suffices to prove that $pre(C)$ is upward-closed.
Take $c \in pre(C)$ and $c' \geq c$. We show $c' \in pre(C)$.
Key idea: "adding more processes to a configuration cannot disable any transition".

$$\begin{array}{ccc} c & \longrightarrow & d \in C \\ \leq & & \\ c' & & \end{array}$$

2. D is upward-closed $\surd$

3. If C is upward-closed then so is $C \cup pre(C)$.

Since union of upward-closed sets is upward-closed, it suffices to prove that $pre(C)$ is upward-closed.
Take $c \in pre(C)$ and $c' \geq c$. We show $c' \in pre(C)$.
Key idea: “adding more processes to a configuration cannot disable any transition”.

$$\begin{array}{ccc} c & \rightarrow & d \in C \\ \leq & & \\ c' & \rightarrow & d' \end{array}$$

2. D is upward-closed $\surd$

3. If C is upward-closed then so is $C \cup pre(C)$.

Since union of upward-closed sets is upward-closed, it suffices to prove that $pre(C)$ is upward-closed.
Take $c \in pre(C)$ and $c' \geq c$. We show $c' \in pre(C)$.
Key idea: "adding more processes to a configuration cannot disable any transition".

$$\begin{array}{ccc} c & \longrightarrow & d \in C \\ \leq & & \leq \\ c' & \longrightarrow & d' \end{array}$$

2. D is upward-closed $\surd$

3. If C is upward-closed then so is $C \cup pre(C)$. $\surd$

Since union of upward-closed sets is upward-closed, it suffices to prove that $pre(C)$ is upward-closed.
Take $c \in pre(C)$ and $c' \geq c$. We show $c' \in pre(C)$.
Key idea: "adding more processes to a configuration cannot disable any transition".

$$\begin{array}{ccc} c & \rightarrow & d \in C \\ \leq & & \leq \\ c' & \rightarrow & d' \in C \end{array}$$

4. $C \cap I$ is decidable. $\surd$

4. $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

4. $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

6. Any chain $U_1 \subseteq U_2 \subseteq U_3 \ldots$ of upward-closed sets reaches a fixpoint after finitely many steps.

4. $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

6. Any chain $U_1 \subseteq U_2 \subseteq U_3 \ldots$ of upward-closed sets reaches a fixpoint after finitely many steps.

Assume this is not the case: $U_1 \subset U_2 \subset U_3 \ldots$

4. $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

6. Any chain $U_1 \subseteq U_2 \subseteq U_3 \ldots$ of upward-closed sets reaches a fixpoint after finitely many steps.

Assume this is not the case: $U_1 \subset U_2 \subset U_3 \ldots$

Pick some minimal element $m_1 \in U_1$.
Pick for every $i > 1$ some minimal element $m_i \notin U_1 \cup \ldots \cup U_{i-1} = U_{i-1}$.
Consider the sequence $m_1, m_2, m_3, \ldots$

4. $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

6. Any chain $U_1 \subseteq U_2 \subseteq U_3 \ldots$ of upward-closed sets reaches a fixpoint after finitely many steps.

Assume this is not the case: $U_1 \subset U_2 \subset U_3 \ldots$

Pick some minimal element $m_1 \in U_1$.
Pick for every $i > 1$ some minimal element $m_i \notin U_1 \cup \ldots \cup U_{i-1} = U_{i-1}$.
Consider the sequence $m_1, m_2, m_3, \ldots$

Let i, j be any two indices satisfying $i < j$.
Since $m_j \notin U_i$, we have $m_i \not\leq m_j$ by upward-closedness of U_i.

4. $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

6. Any chain $U_1 \subseteq U_2 \subseteq U_3 \ldots$ of upward-closed sets reaches a fixpoint after finitely many steps.

Assume this is not the case: $U_1 \subset U_2 \subset U_3 \ldots$

Pick some minimal element $m_1 \in U_1$.
Pick for every $i > 1$ some minimal element $m_i \notin U_1 \cup \ldots \cup U_{i-1} = U_{i-1}$.
Consider the sequence $m_1, m_2, m_3, \ldots$

Let i, j be any two indices satisfying $i < j$.
Since $m_j \notin U_i$, we have $m_i \not\leq m_j$ by upward-closedness of U_i.

Contradiction to Dickson's lemma.

4. $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

6. Any chain $U_1 \subseteq U_2 \subseteq U_3 \ldots$ of upward-closed sets reaches a fixpoint after finitely many steps. $\surd$

Assume this is not the case: $U_1 \subset U_2 \subset U_3 \ldots$

Pick some minimal element $m_1 \in U_1$.
Pick for every $i > 1$ some minimal element $m_i \notin U_1 \cup \ldots \cup U_{i-1} = U_{i-1}$.
Consider the sequence $m_1, m_2, m_3, \ldots$

Let i, j be any two indices satisfying $i < j$.
Since $m_j \notin U_i$, we have $m_i \not\leq m_j$ by upward-closedness of U_i.

Contradiction to Dickson's lemma.

Complexity

Consider the sequences $C = c_1, c_2, c_3, \ldots$, where $c_i \in \mathbb{N}^k$ for all $i \geq 1$, that satisfy:

- $c_1 \leq (1, \ldots, 1)$, and
- $|c_i(j) - c_{i+1}(j)| \leq 1$ for every $i \geq 1, 1 \leq j \leq k$.

By Dickson's lemma any such sequence contains indices i, j such that $c_i \leq c_j$.

Let $J(C)$ be the smallest j for which such an i exist.

Let $G(k)$ be the maximum over all C's of the index $J(C)$.

How fast can G grow?

Complexity

Consider the sequences $C = c_1, c_2, c_3, \ldots$, where $c_i \in \mathbb{N}^k$ for all $i \geq 1$, that satisfy:

- $c_1 \leq (1, \ldots, 1)$, and
- $|c_i(j) - c_{i+1}(j)| \leq 1$ for every $i \geq 1, 1 \leq j \leq k$.

By Dickson's lemma any such sequence contains indices i, j such that $c_i \leq c_j$.

Let $J(C)$ be the smallest j for which such an i exist.

Let $G(k)$ be the maximum over all C's of the index $J(C)$.

How fast can G grow?

Theorem [Mayr,Meyer, JACM '81]: The function G is non-primitive recursive.

Complexity

Consider the sequences $C = c_1, c_2, c_3, \ldots$, where $c_i \in \mathbb{N}^k$ for all $i \geq 1$, that satisfy:

- $c_1 \leq (1, \ldots, 1)$, and
- $|c_i(j) - c_{i+1}(j)| \leq 1$ for every $i \geq 1, 1 \leq j \leq k$.

By Dickson's lemma any such sequence contains indices i, j such that $c_i \leq c_j$.

Let $J(C)$ be the smallest j for which such an i exist.

Let $G(k)$ be the maximum over all C's of the index $J(C)$.

How fast can G grow?

Theorem [Mayr,Meyer, JACM '81]: The function G is non-primitive recursive.

Backward search may need a non-primitive recursive number of iterations.

However: Still useful in practice!

Application to the MESI-protocol

Are the states M and S mutually exclusive?

Check if the upward-closed set with minimal element

$$m = 1,\ e = 0,\ s = 1,\ i = 0$$

can be reached from the initial parametrized configuration

$$m = 0,\ e = 0,\ s = 0,\ i = \sqcup$$

Proceed as follows:

Application to the MESI-protocol

Are the states M and S mutually exclusive?

Check if the upward-closed set with minimal element

$$m = 1,\ e = 0,\ s = 1,\ i = 0$$

can be reached from the initial parametrized configuration

$$m = 0,\ e = 0,\ s = 0,\ i = \sqcup$$

Proceed as follows:

$$D: \quad m \geq 1 \wedge s \geq 1$$

Application to the MESI-protocol

Are the states M and S mutually exclusive?

Check if the upward-closed set with minimal element

$$m = 1,\ e = 0,\ s = 1,\ i = 0$$

can be reached from the initial parametrized configuration

$$m = 0,\ e = 0,\ s = 0,\ i = \sqcup$$

Proceed as follows:

$$\begin{aligned} D: &\quad m \geq 1 \land s \geq 1 \\ D \cup pre(D): &\quad (m \geq 1 \land s \geq 1) \lor \\ &\quad (m = 0 \land e = 1 \land s \geq 1) \end{aligned}$$

Application to the MESI-protocol

Are the states M and S mutually exclusive?

Check if the upward-closed set with minimal element

$$m = 1,\ e = 0,\ s = 1,\ i = 0$$

can be reached from the initial parametrized configuration

$$m = 0,\ e = 0,\ s = 0,\ i = \sqcup$$

Proceed as follows:

$$\begin{aligned}
D\text{:} &\quad m \geq 1 \wedge s \geq 1 \\
D \cup pre(D)\text{:} &\quad (m \geq 1 \wedge s \geq 1) \vee \\
&\quad (m = 0 \wedge e = 1 \wedge s \geq 1) \\
D \cup pre(D) \cup pre^2(D)\text{:} &\quad D \cup pre(D)
\end{aligned}$$

Case studies

Other cache-coherence protocols: Berkeley RISC, Illinois, Xerox PARC Dragon, DEC Firefly, Futurebus +, etc.

[Delzanno, FMSD'03]:

- Model extended with more complicated guards.
- Termination guarantee gets lost.
- Upward-closed sets represented by linear constraints.
- Backward-search algorithm must be refined: Possibly more iterations, but each iteration has lower complexity.

[Emerson,Kahlon, CHARME'03,TACAS'03]:

- Restricted models still able to model the cache-coherence protocols.
- Much faster algorithms.

Symbolic search for other models

Lossy channel systems [Abdulla and Jonsson, I&C ’93], [Abdulla et al, CAV’98].

- Configuration: $\langle q, \mathbf{w} \rangle$, where q state and $\mathbf{w} = (w_1, \ldots, w_n)$ vector of words representing the current queue contents
- Family $\mathcal{C}$: upward-closed sets with respect to the subsequence order
  $abba \leq bbaabaaabbabb$
  Dickson’s lemma $\rightarrow$ Higman’s lemma
- Backward search satisfies (1) - (6).

Timed Petri nets [Abdulla and Nylén, ICATPN’01].

- Configuration: $\langle q, B \rangle$, where B finite bag of vectors of reals.
- Family $\mathcal{C}$: existential zones.

Repeated reachability in broadcast protocols

The following problem is undecidable:

Given: a broadcast protocol,

an initial parametrized configuration $p = (\sqcup, 0, \ldots, 0)$

To decide: is there an integer n such that the transition system

with $(n, 0, \ldots, 0)$ as initial configuration

has an infinite computation ?

Can be reformulated as a repeated reachability problem where
$I = (\sqcup, 0, \ldots, 0)$ and D = set of all configurations.

Pushdown automata

Pushdown automata

Automata extended with one stack.

Transitions:

- Guards: check the topmost symbol in the stack.
- Actions: replace the topmost symbol by a fixed word.
- Notation: $\langle p, \gamma \rangle \hookrightarrow \langle p', v \rangle$
- Normalization: $|v| \leq 2$.

We use P, Γ, Δ for the sets of control states, stack symbols, and rules, respectively.

Configurations: pairs $\langle p, w \rangle$, where p is a control state and w is a word.
(Stack, topmost symbol is the first letter.)

PDAs as models of sequential programs

Programs determined by:

- Control flow: assignments, conditionals, loops ,
  procedure calls with parameters/return values.
- Local variables of each procedure.
- Global variables.

State space determined by:

- Program pointer.
- Values of global variables.
- Values of local variables (of current procedure).
- Activation records (return addresses, copies of locals).

Interpretation of a configuration $\langle q, \gamma v \rangle$:

q holds values of global variables.

γ holds (program pointer, values of local variables).

v holds stack of (return address, saved locals).

Restriction: finite datatypes.

Correspondence between statements and rules:

$\langle q, \gamma \rangle \hookrightarrow \langle q', \gamma' \rangle$	simple statement
$\langle q, \gamma \rangle \hookrightarrow \langle q', \gamma' \gamma'' \rangle$	procedure call
$\langle q, \gamma \rangle \hookrightarrow \langle q', \epsilon \rangle$	return statement

Case study: Drawing skylines

```
void m() {
    if (?) {
        s(); right();
        if (?) m();
    } else {
        up(); m(); down();
    }
}
```

```
void s() {
    if (?) return;
    up(); m(); down();
}

main() {
  s();
}
```

Model

`void s() {`	**var** `st`:**stack** `of` $\{s_0, \ldots, s_5, \ldots\}$
s_0`: if (?)` s_1`: return;`	$\langle p, s_0 \rangle \hookrightarrow \langle p, s_2 \rangle \quad \langle p, s_0 \rangle \hookrightarrow \langle p, \epsilon \rangle$
s_2`: up();`	$\langle p, s_2 \rangle \hookrightarrow \langle p, up_0\, s_3 \rangle$
s_3`: m();`	$\langle p, s_3 \rangle \hookrightarrow \langle p, m_0\, s_4 \rangle$
s_4`: down();` s_5`:`	$\langle p, s_4 \rangle \hookrightarrow \langle p, down_0\, s_5 \rangle \quad \langle p, s_5 \rangle \hookrightarrow \langle p, \epsilon \rangle$
`}`	

Symbolic reachability in pushdown automata

A set of configurations C is regular if for every control point p, the set $\{w \in \Gamma^* \mid \langle p, w \rangle \in C\}$ is regular.

Typically, I and D are regular sets of configurations.
(Even very simple ones, like $\langle p, \Gamma^* \rangle$.)

Family $\mathcal{C}$: regular sets

Backward search: Do conditions (1) - (6) hold ?

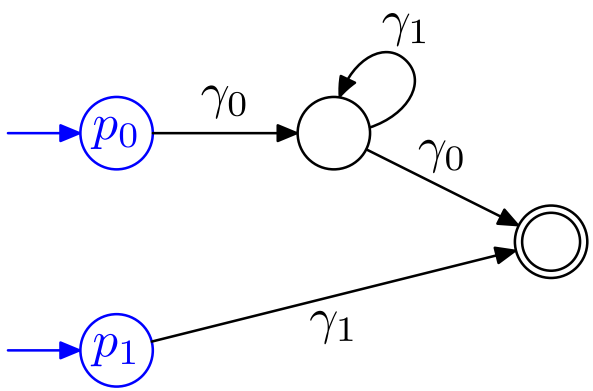
1. Each regular set can be finitely represented by a NFA. $\surd$

NFA for a pushdown system:

- P as set of initial states and Γ as alphabet.
- $\langle p, v \rangle$ recognized if $p \xrightarrow{v} q$ for some final state q.

Example: $P = \{p_0, p_1\}$ and $\Gamma = \{\gamma_0, \gamma_1\}$

Automaton coding the set $\langle p_0, \gamma_0\gamma_1^*\gamma_0 \rangle \cup \langle p_1, \gamma_1 \rangle$:



2. $F \in \mathcal{C}$ $\surd$

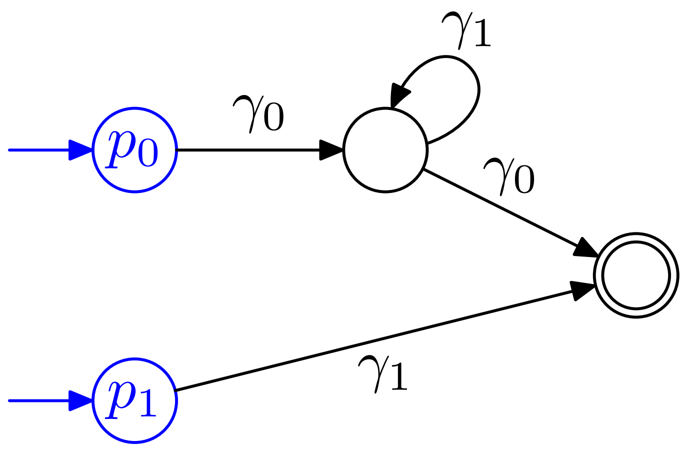
2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

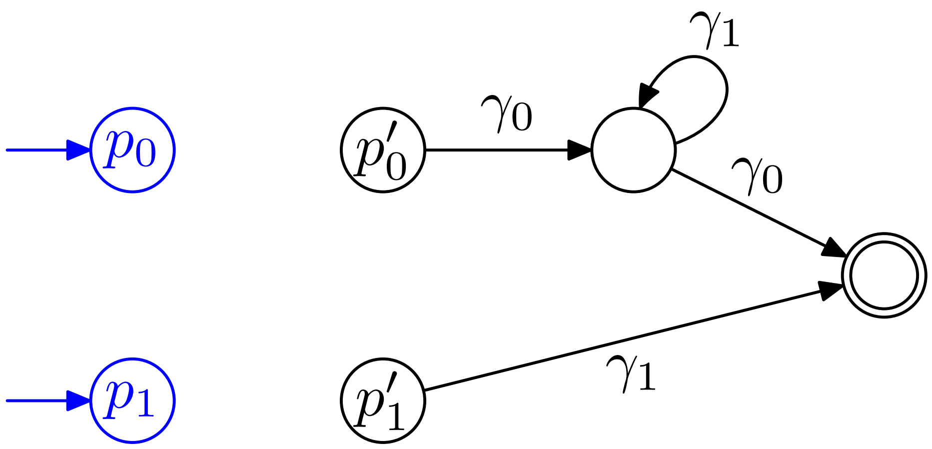
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \ \}$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

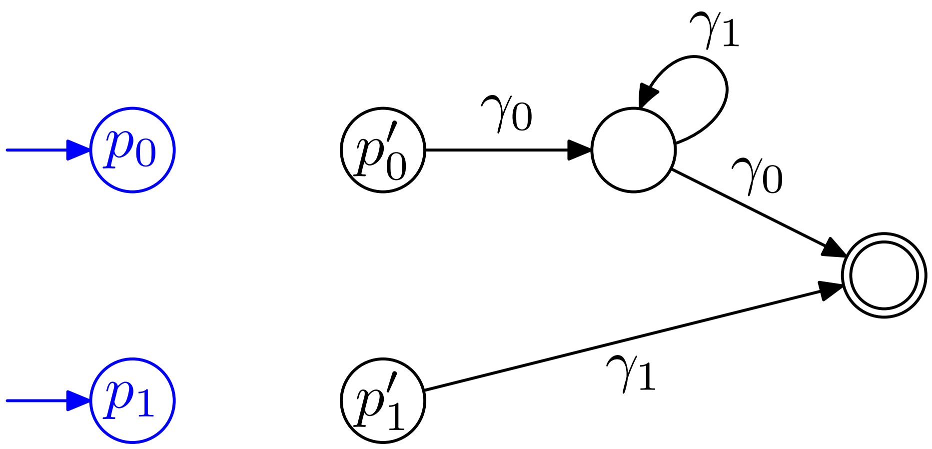
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \;,\; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \;,\; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

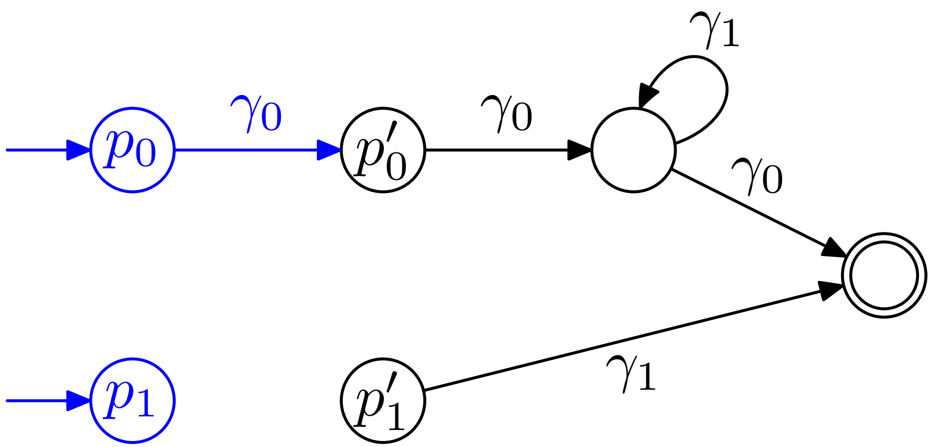
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

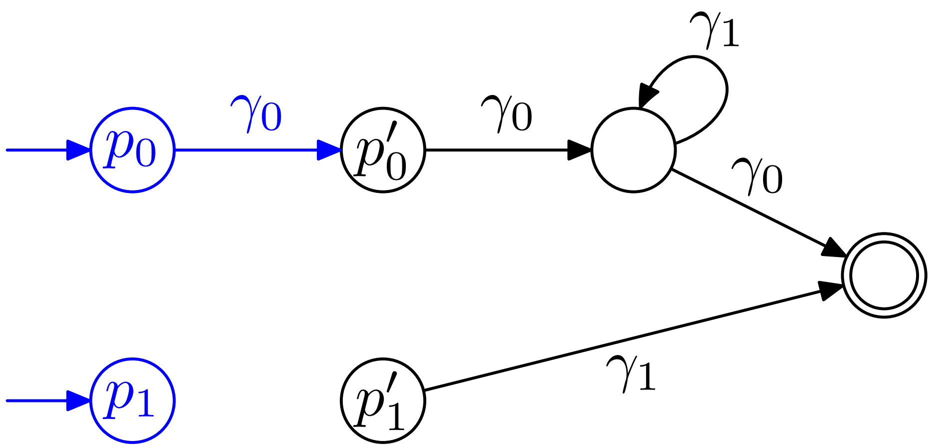
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

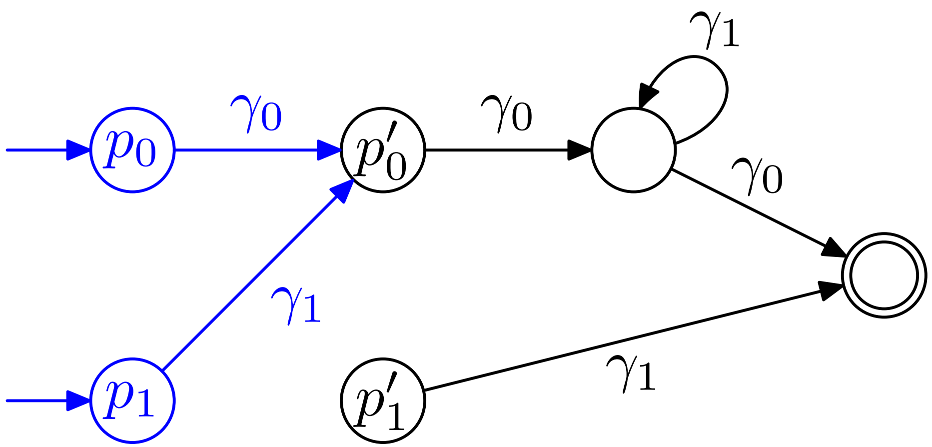
$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

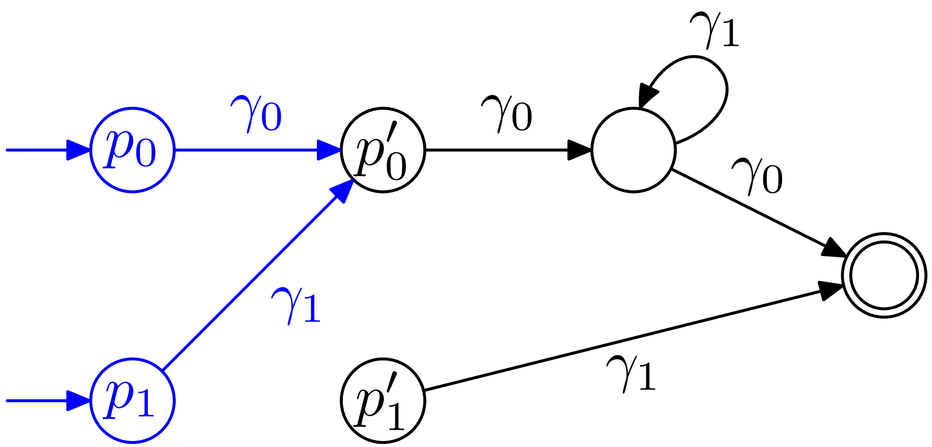
$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\checkmark$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

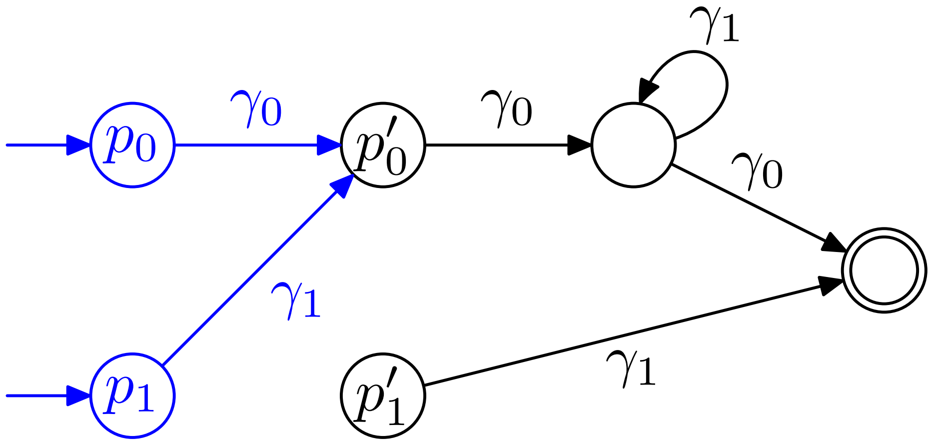
$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1\gamma_0 \rangle$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

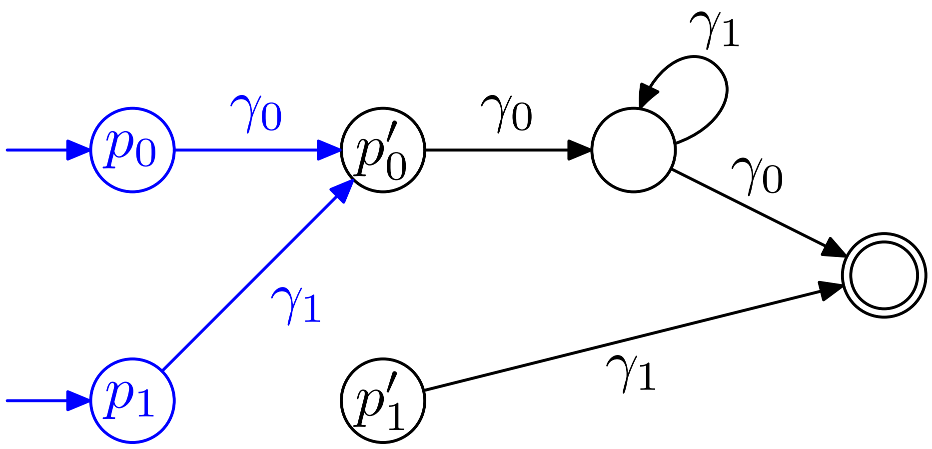
$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

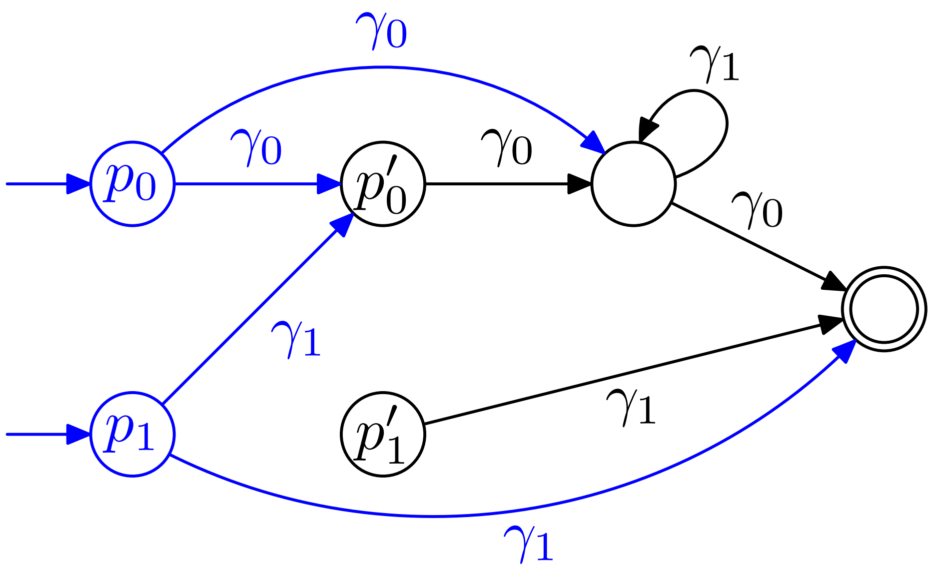
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \,,\; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \,,\; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

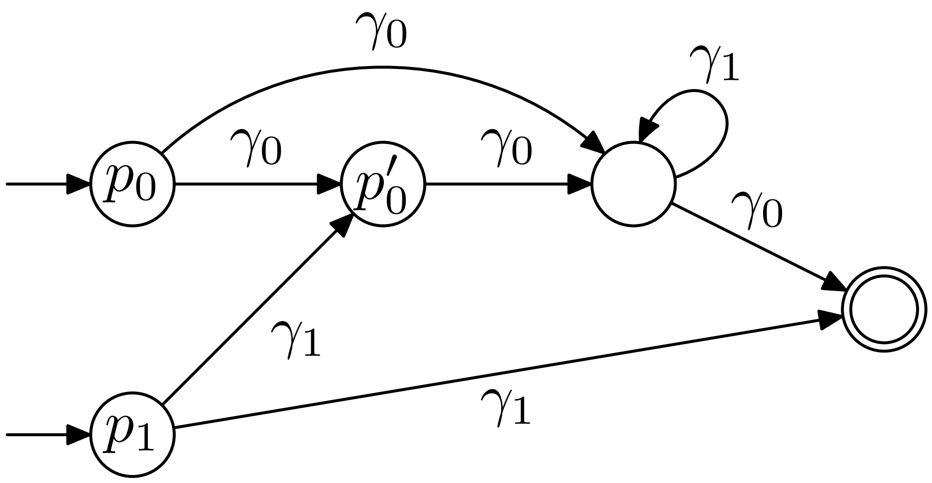
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

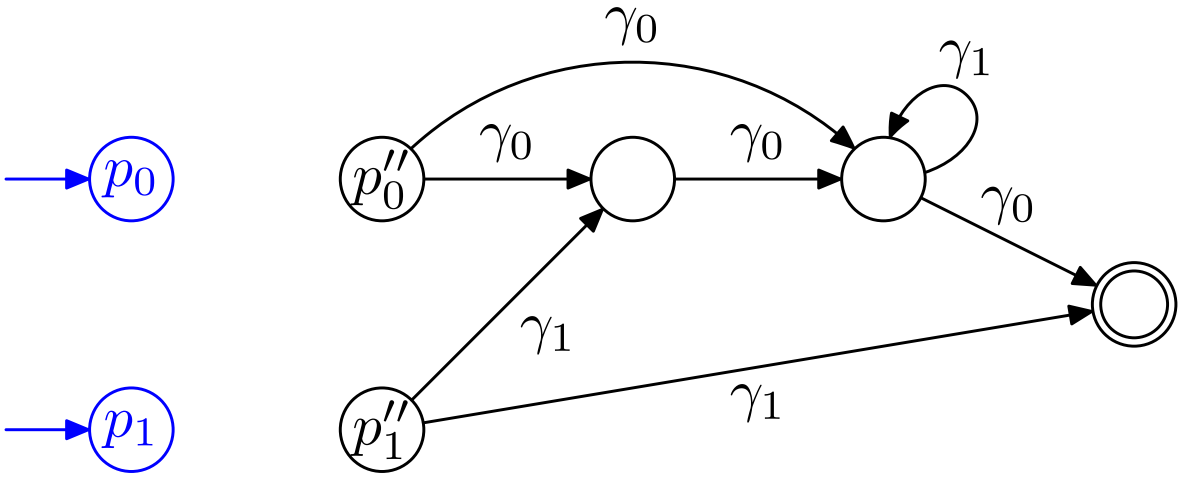
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



2. $\color{red}{F} \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

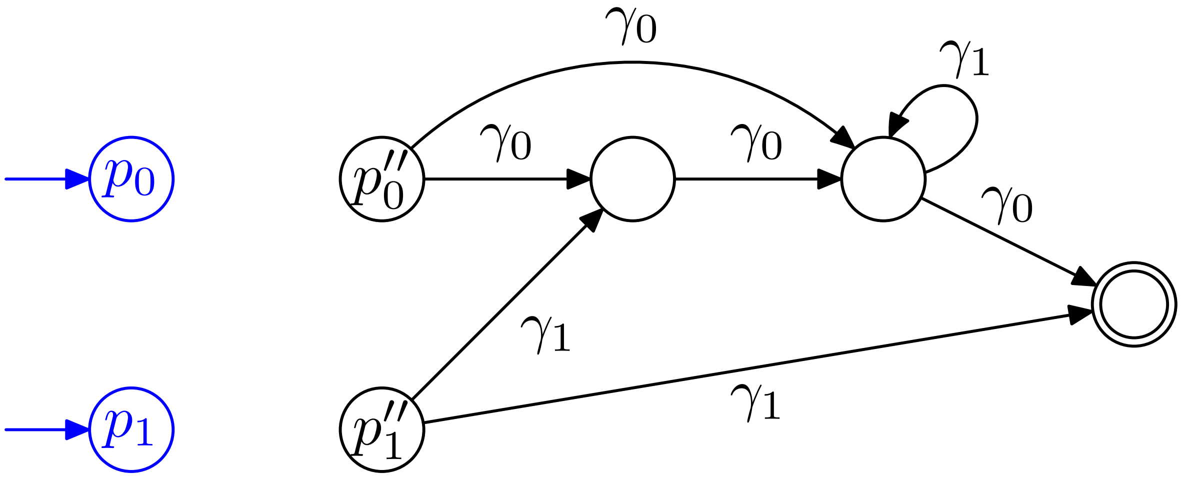
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \ \}$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

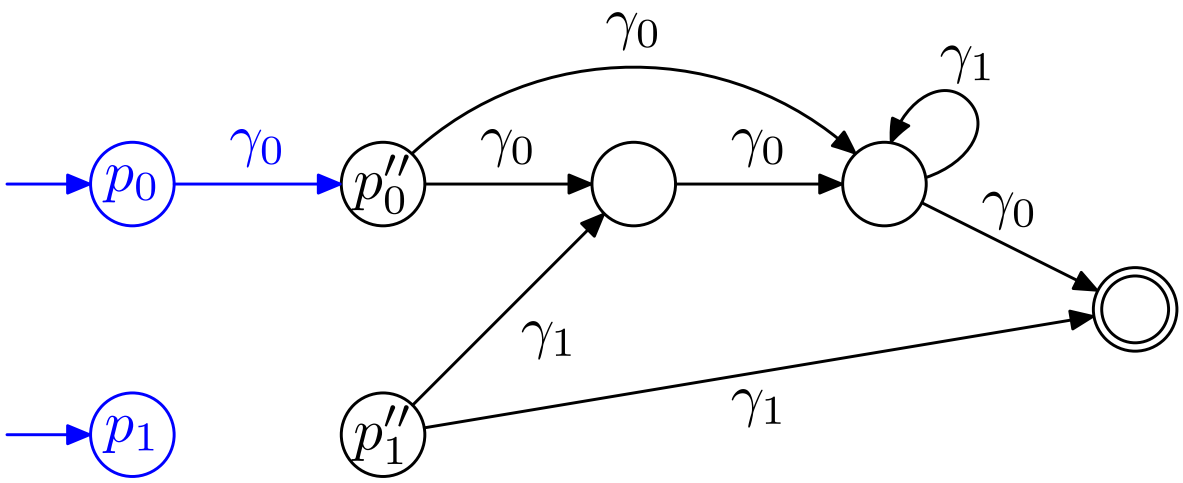
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

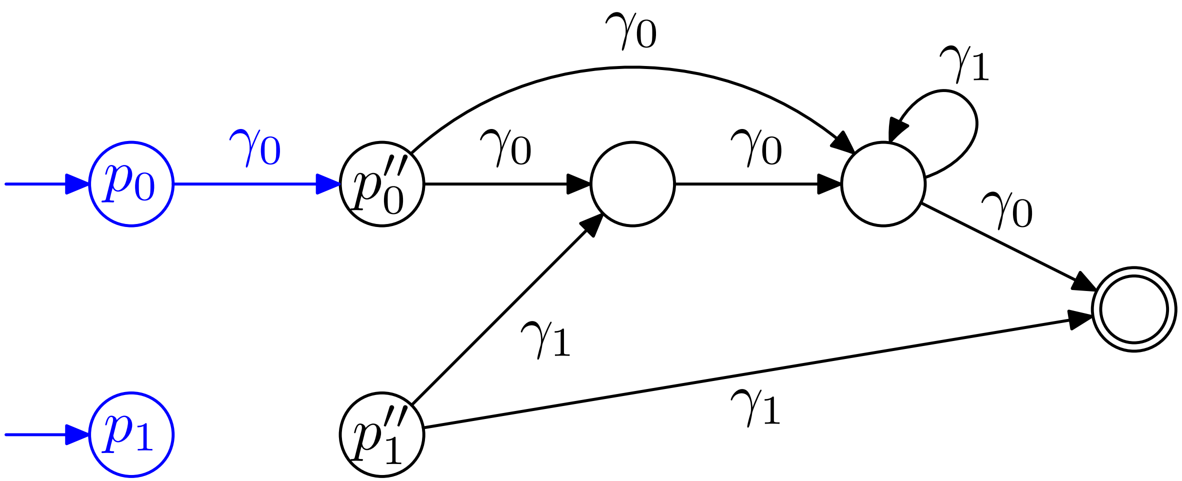
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

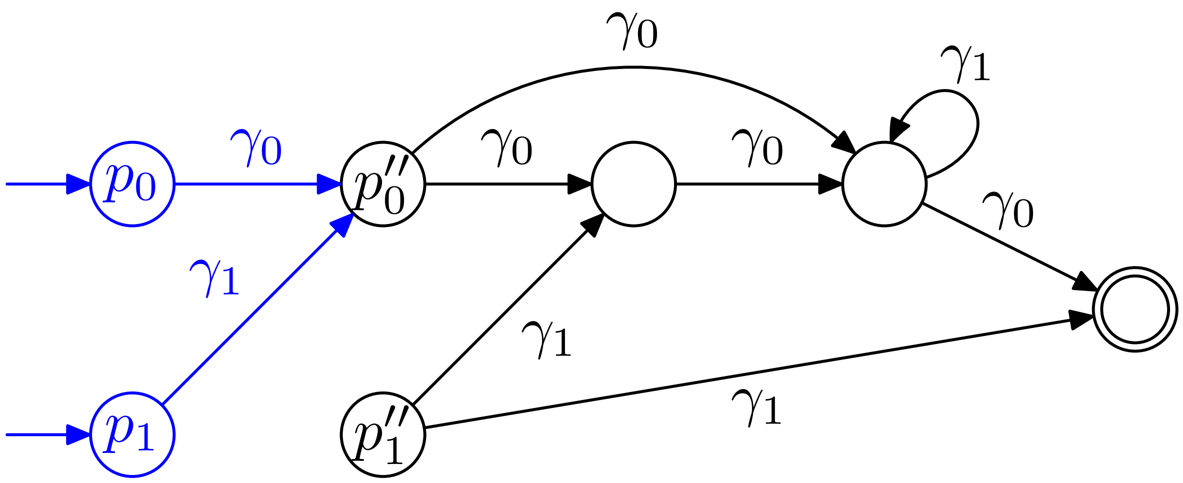
$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

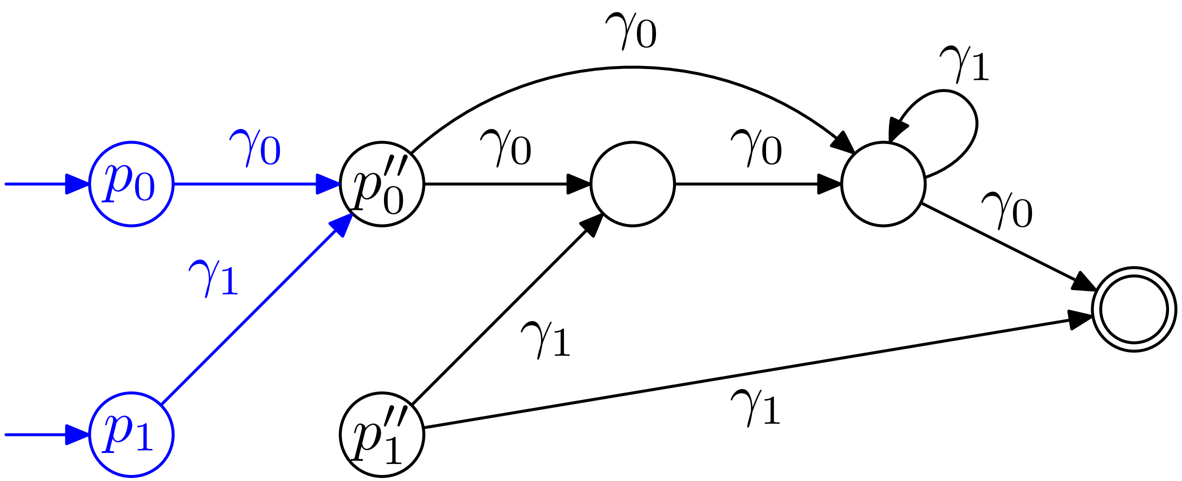
$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

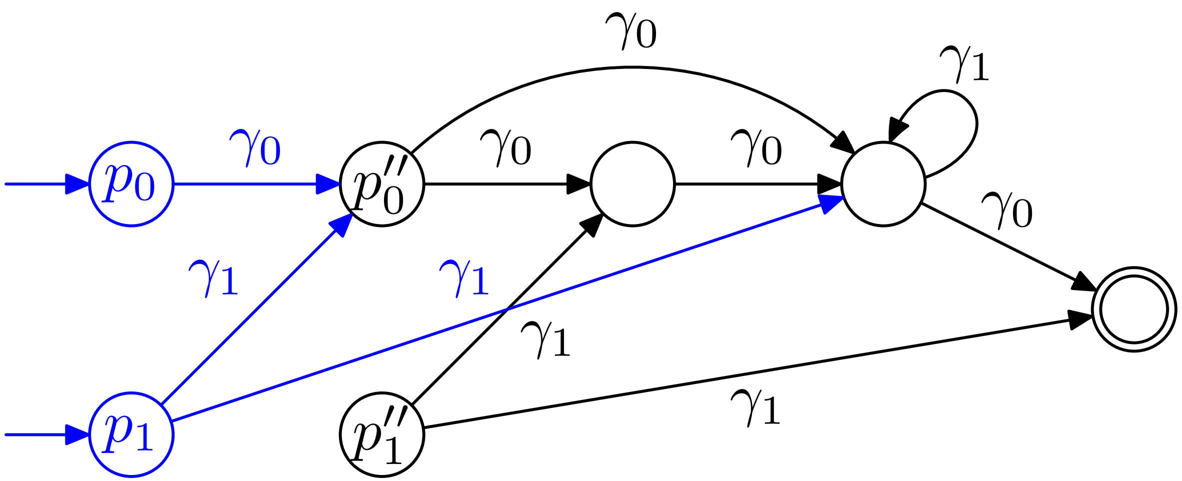
$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1\gamma_0 \rangle$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

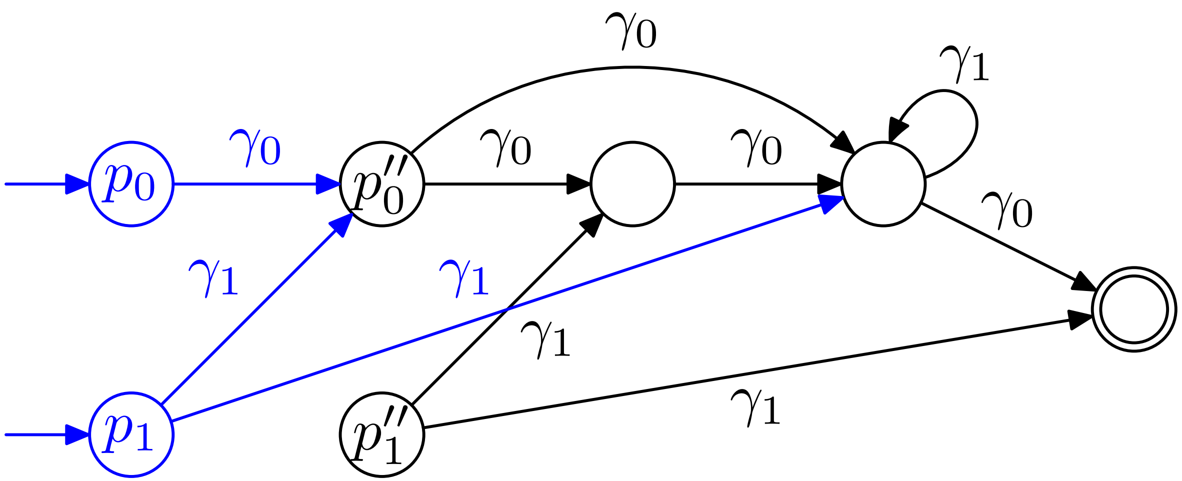
$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

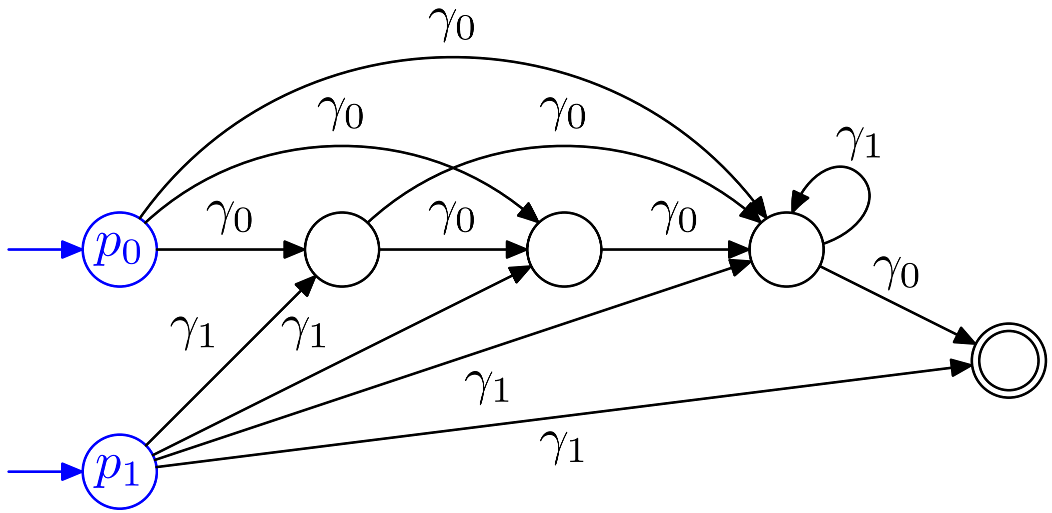
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \ \}$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$.

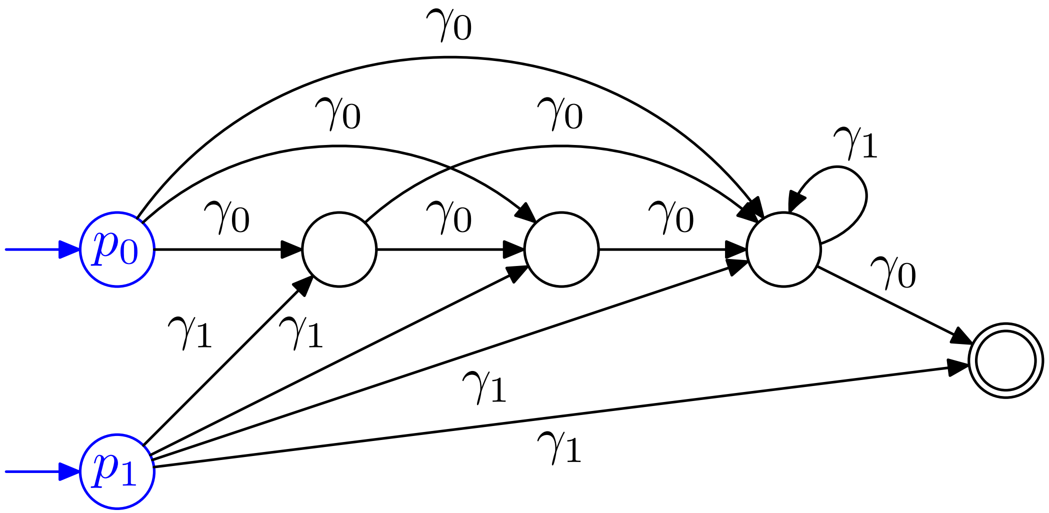
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



2. $F \in \mathcal{C}$ $\surd$

3. If $C \in \mathcal{C}$, then $C \cup pre(C) \in \mathcal{C}$. $\surd$

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



4. Emptiness of $C \cap I$ is decidable. $\surd$

4. Emptiness of $C \cap I$ is decidable. $\surd$

5. $C_1 = C_2$ is decidable. $\surd$

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint.

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint.

$P = \{p_0, p_1\}$, $\Gamma = \{\gamma_0, \gamma_1\}$

$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle , \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle , \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint.

$P = \{p_0, p_1\}$, $\Gamma = \{\gamma_0, \gamma_1\}$

$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle , \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle , \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$

$C_0 \;=\; D \qquad = \langle p_0, \gamma_0 \gamma_1^* \gamma_0 \rangle \cup \langle p_1, \gamma_1 \rangle$

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint.

$P = \{p_0, p_1\}$, $\Gamma = \{\gamma_0, \gamma_1\}$

$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \,,\, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \,,\, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$

$$\begin{aligned}
C_0 &= D & &= \langle p_0, \gamma_0 \gamma_1^* \gamma_0 \rangle \cup \langle p_1, \gamma_1 \rangle \\
C_1 &= C_0 \cup pre(C_0) & &= \langle p_0, (\gamma_0 + \gamma_0^2) \gamma_1^* \gamma_0 \rangle \cup \\
& & & \quad \langle p_1, \gamma_1 (\epsilon + \gamma_0) \gamma_1^* (\epsilon + \gamma_0) \rangle
\end{aligned}$$

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint.

$P = \{p_0, p_1\}$, $\Gamma = \{\gamma_0, \gamma_1\}$

$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \ \rangle \}$

$$\begin{array}{rcl l l}
C_0 & = & D & = & \langle p_0, \gamma_0 \gamma_1^* \gamma_0 \rangle \ \cup \ \langle p_1, \gamma_1 \rangle \\
C_1 & = & C_0 \cup pre(C_0) & = & \langle p_0, (\gamma_0 + \gamma_0^2) \gamma_1^* \gamma_0 \rangle \cup \\
 & & & & \langle p_1, \gamma_1 (\epsilon + \gamma_0) \gamma_1^* (\epsilon + \gamma_0) \rangle \\
 & \ldots & & & \\
C_i & = & C_{i-1} \cup pre(C_{i-1}) & = & \langle p_0, (\gamma_0 + \ldots + \gamma_0^{i+1}) \gamma_1^* \gamma_0 \rangle \cup \\
 & & & & \langle p_1, \gamma_1 (\epsilon + \gamma_0 + \ldots + \gamma_0^i) \gamma_1^* (\epsilon + \gamma_0) \rangle \\
 & \ldots & & &
\end{array}$$

6. Any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ eventually reaches a fixpoint. FAILS!

$P = \{p_0, p_1\}$, $\Gamma = \{\gamma_0, \gamma_1\}$

$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \ \rangle \}$

$$
\begin{array}{rcl c l}
C_0 & = & D & = & \langle p_0, \gamma_0 \gamma_1^* \gamma_0 \rangle \ \cup \ \langle p_1, \gamma_1 \rangle \\
C_1 & = & C_0 \cup pre(C_0) & = & \langle p_0, (\gamma_0 + \gamma_0^2) \gamma_1^* \gamma_0 \rangle \cup \\
 & & & & \langle p_1, \gamma_1 (\epsilon + \gamma_0) \gamma_1^* (\epsilon + \gamma_0) \rangle \\
 & \ldots & & & \\
C_i & = & C_{i-1} \cup pre(C_{i-1}) & = & \langle p_0, (\gamma_0 + \ldots + \gamma_0^{i+1}) \gamma_1^* \gamma_0 \rangle \cup \\
 & & & & \langle p_1, \gamma_1 (\epsilon + \gamma_0 + \ldots + \gamma_0^{i}) \gamma_1^* (\epsilon + \gamma_0) \rangle \\
 & \ldots & & &
\end{array}
$$

However, the fixpoint

$$\begin{aligned} pre^*(D) &= \langle p_0, \gamma_0^+ \gamma_1^* \gamma_0 \rangle \cup \\ & \quad \langle p_1, \gamma_1 \gamma_0^* \gamma_1^* (\epsilon + \gamma_0) \rangle \end{aligned}$$

is regular.

How can we compute it?

Accelerations

By definition, $pre(D) = \bigcup_{i \geq 0} C_i$
where $C_0 = D$ and $C_{i+1} = C_i \cup pre(C_i)$ for every $i \geq 0$

If convergence fails, try to compute an acceleration :
a sequence $D_0 \subseteq D_1 \subseteq D_2 \ldots$ such that

(a) $\forall i \geq 0 \colon C_i \subseteq D_i$

(b) $\forall i \geq 0 \colon D_i \subseteq \bigcup_{j \geq 0} C_j = pre(D)$

Property (a) ensures capture of (at least) the whole set $pre(D)$

Property (b) ensures that only elements of $pre(D)$ are captured

The acceleration guarantees termination if

(c) $\exists i \geq 0 \colon D_{i+1} = D_i$

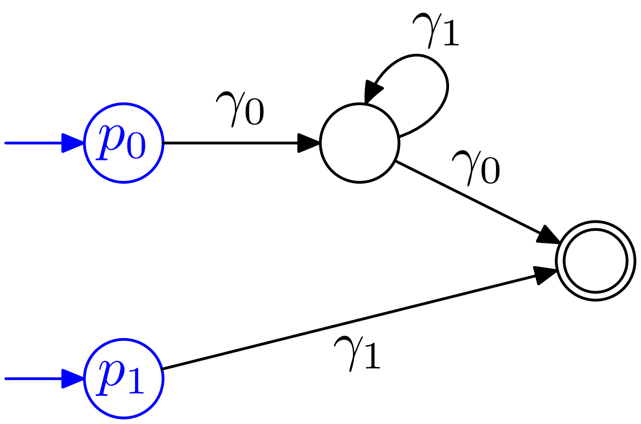
An acceleration for pushdown automata

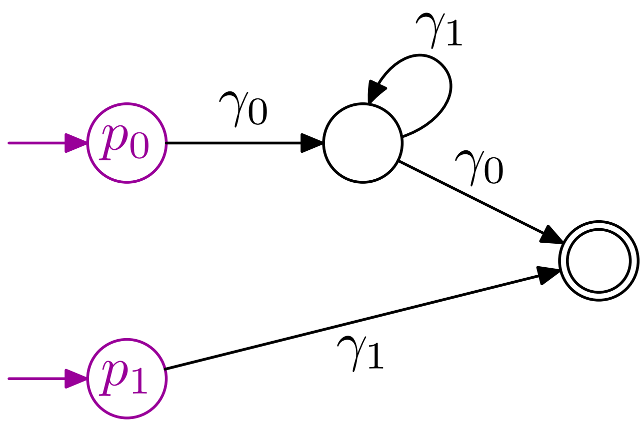
Idea: reuse the same states

An acceleration for pushdown automata

Idea: reuse the same states

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$

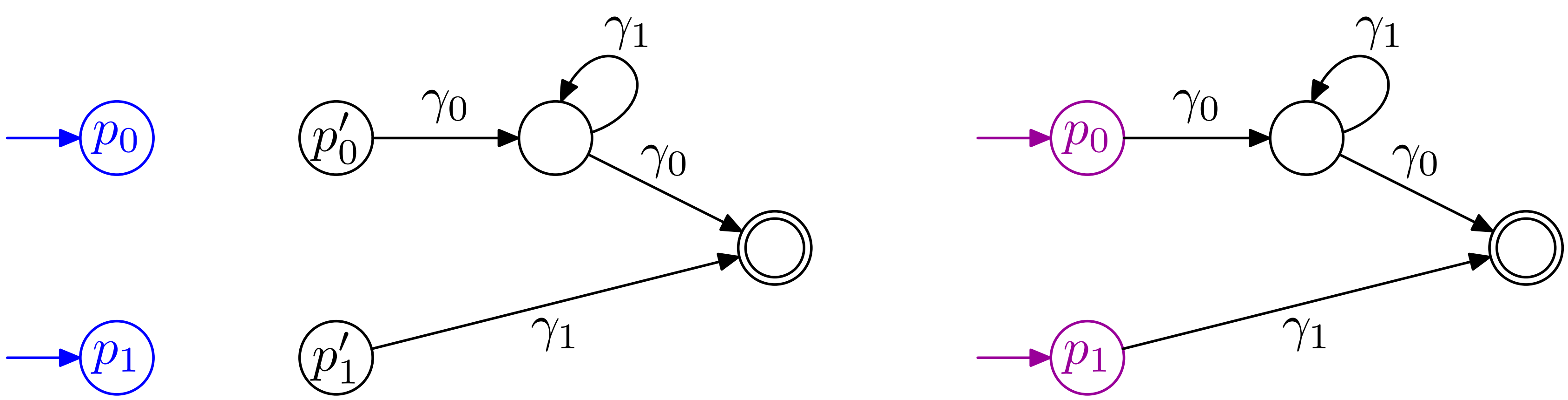




An acceleration for pushdown automata

Idea: reuse the same states

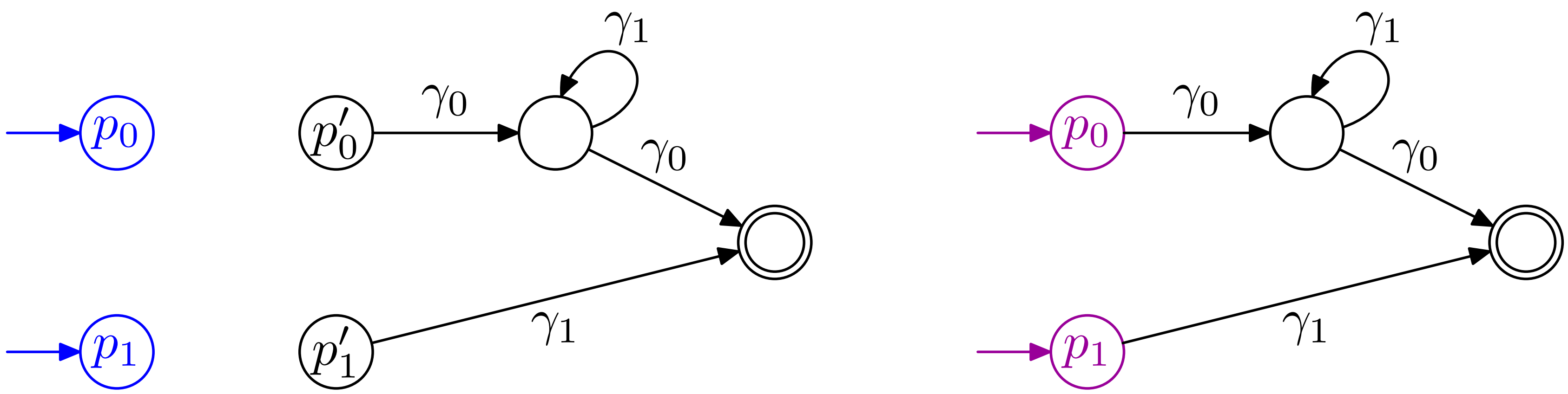
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \;,\; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \;,\; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



An acceleration for pushdown automata

Idea: reuse the same states

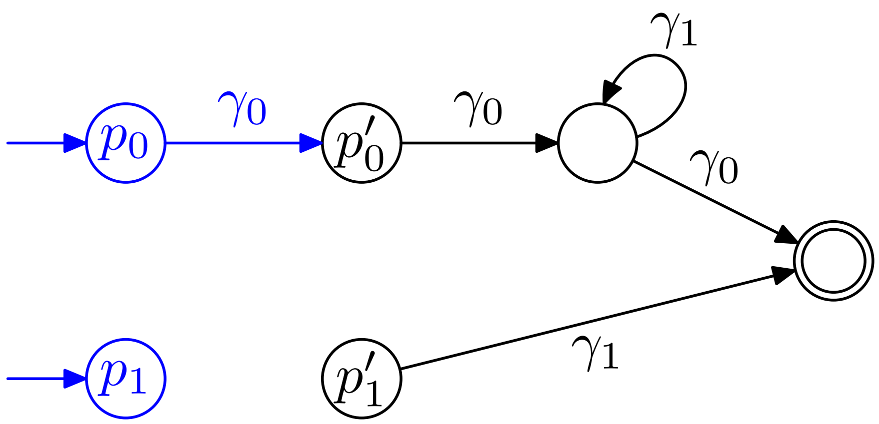
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

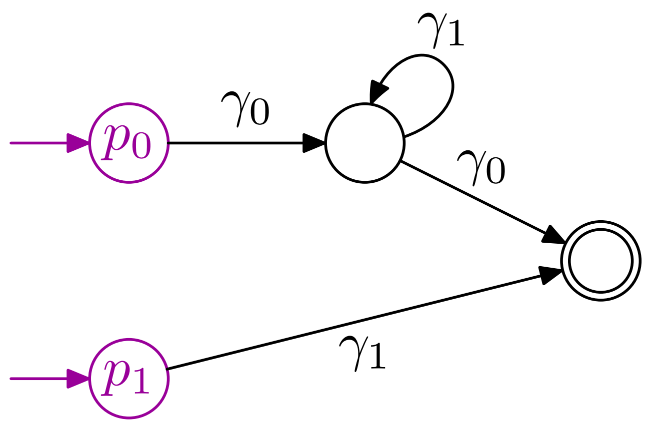


An acceleration for pushdown automata

Idea: reuse the same states

$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

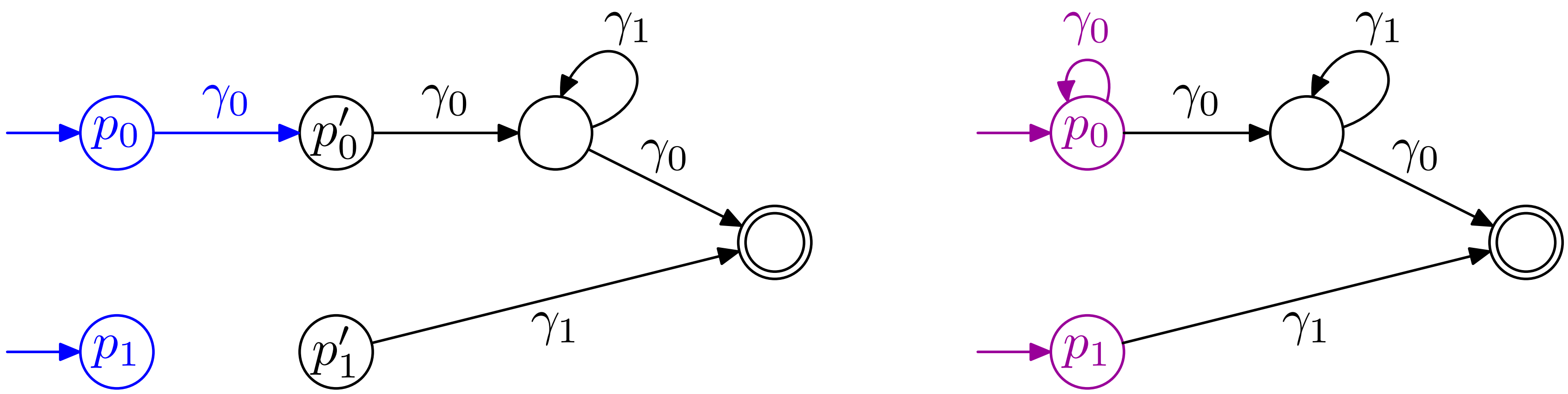




An acceleration for pushdown automata

Idea: reuse the same states

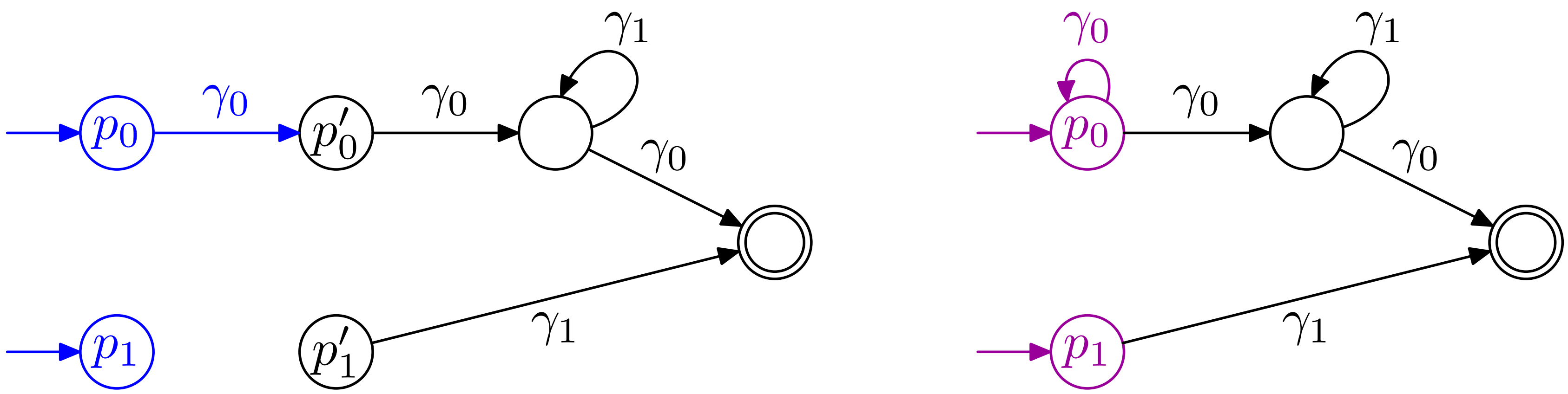
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



An acceleration for pushdown automata

Idea: reuse the same states

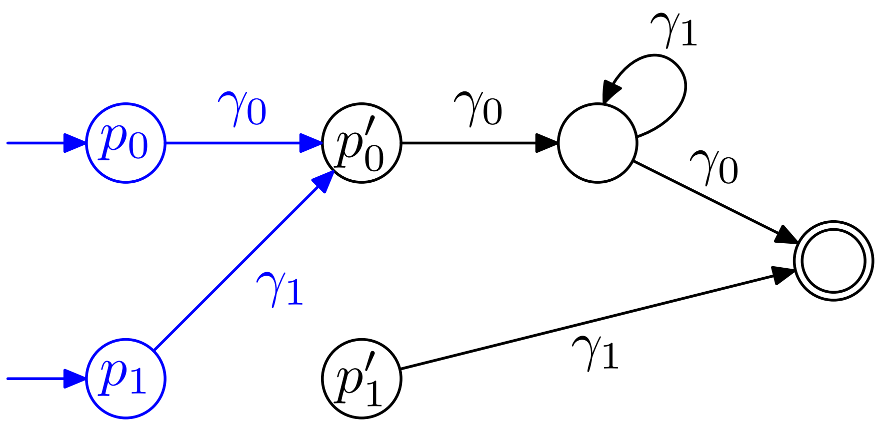
$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

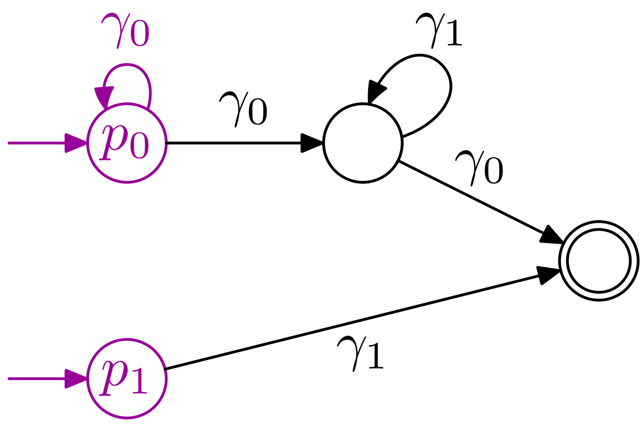


An acceleration for pushdown automata

Idea: reuse the same states

$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

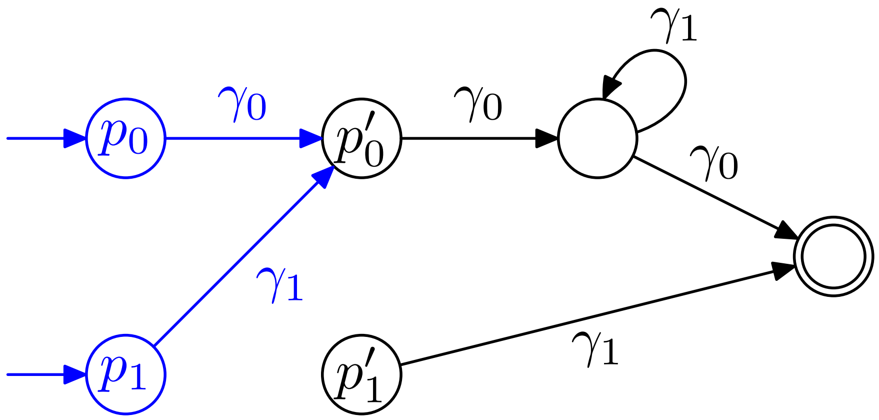


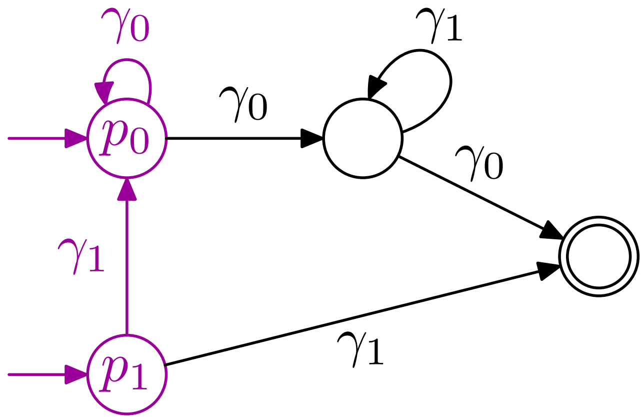


An acceleration for pushdown automata

Idea: reuse the same states

$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

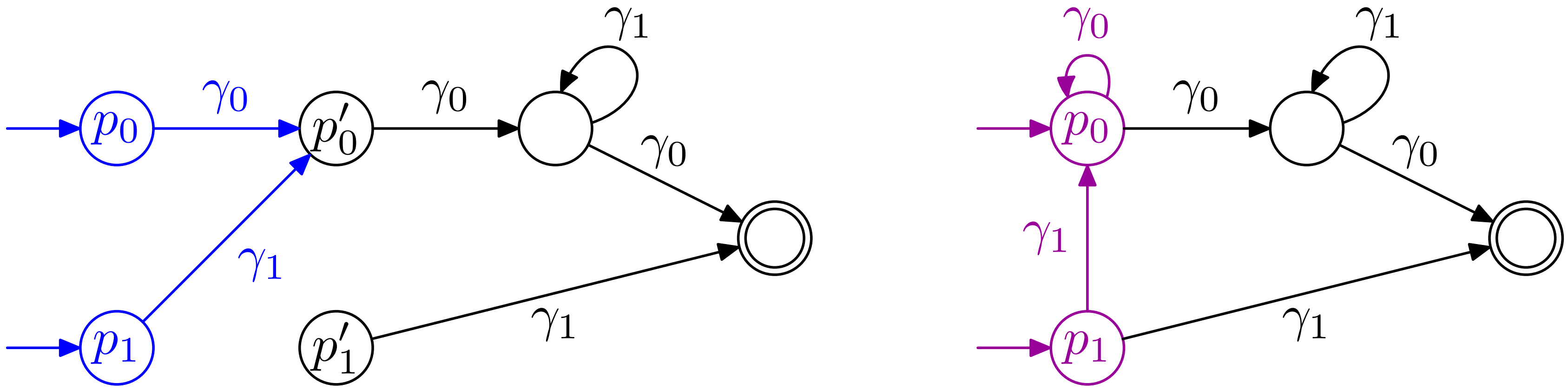




An acceleration for pushdown automata

Idea: reuse the same states

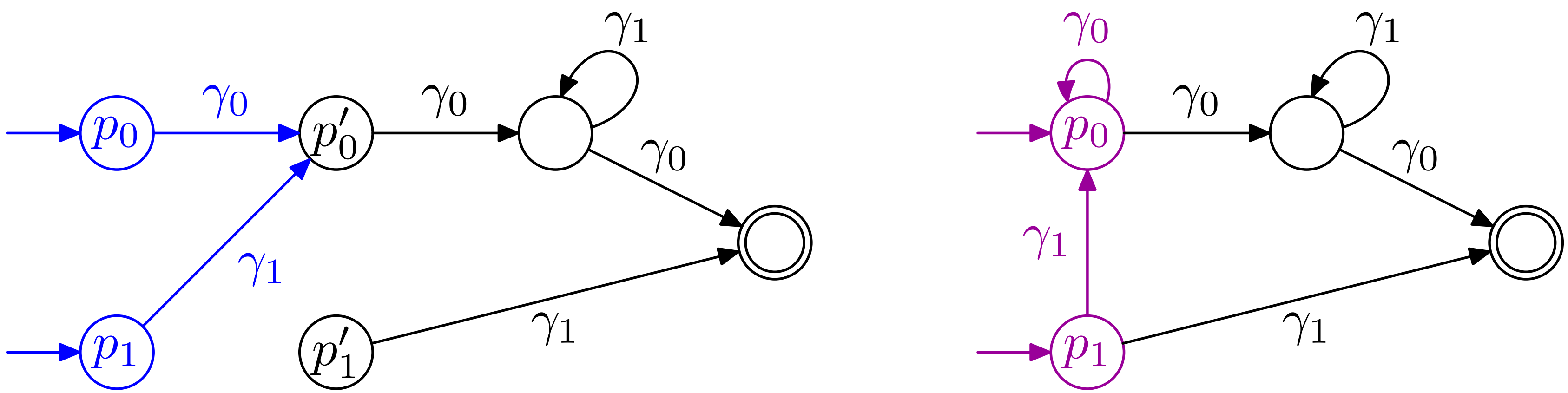
$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1\gamma_0 \rangle$



An acceleration for pushdown automata

Idea: reuse the same states

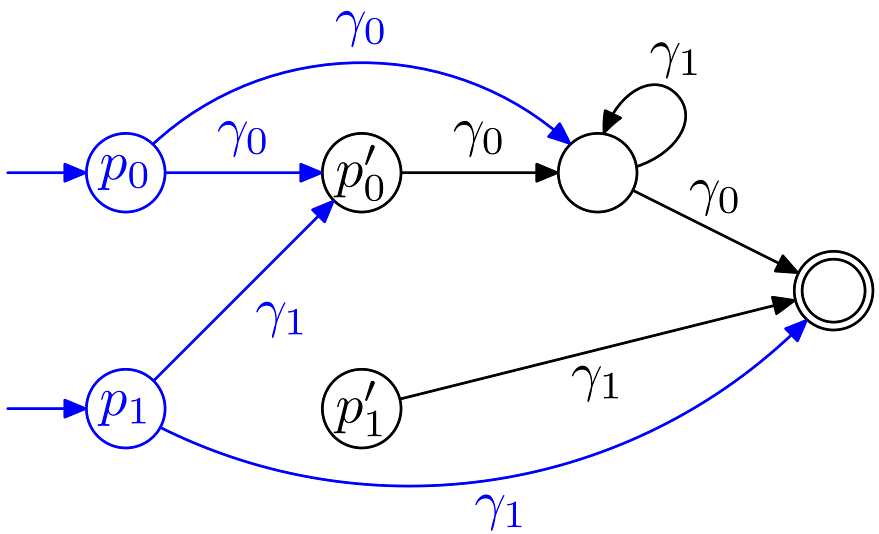
$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$

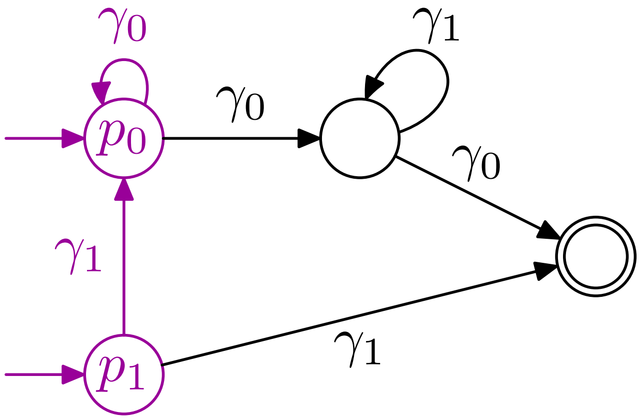


An acceleration for pushdown automata

Idea: reuse the same states

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \ \rangle \}$$

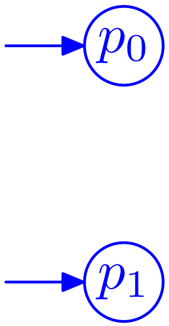


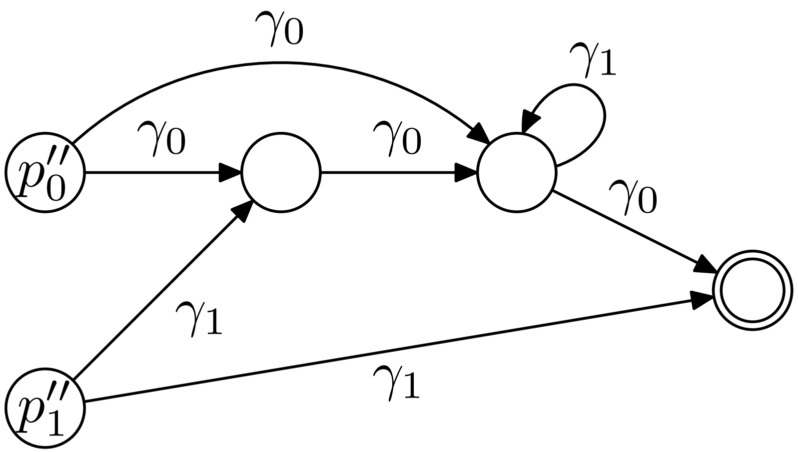


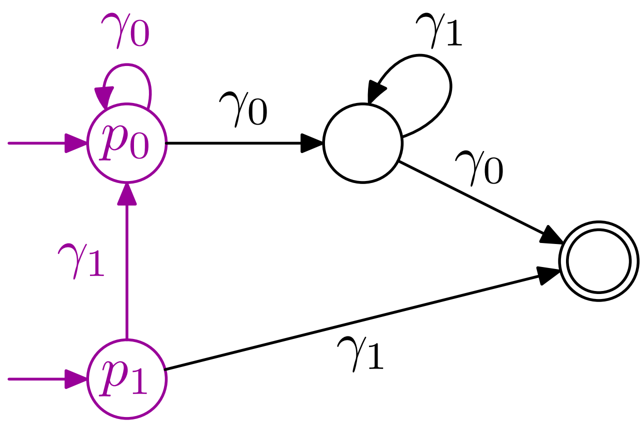
An acceleration for pushdown automata

Idea: reuse the same states

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \; , \; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \; , \; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



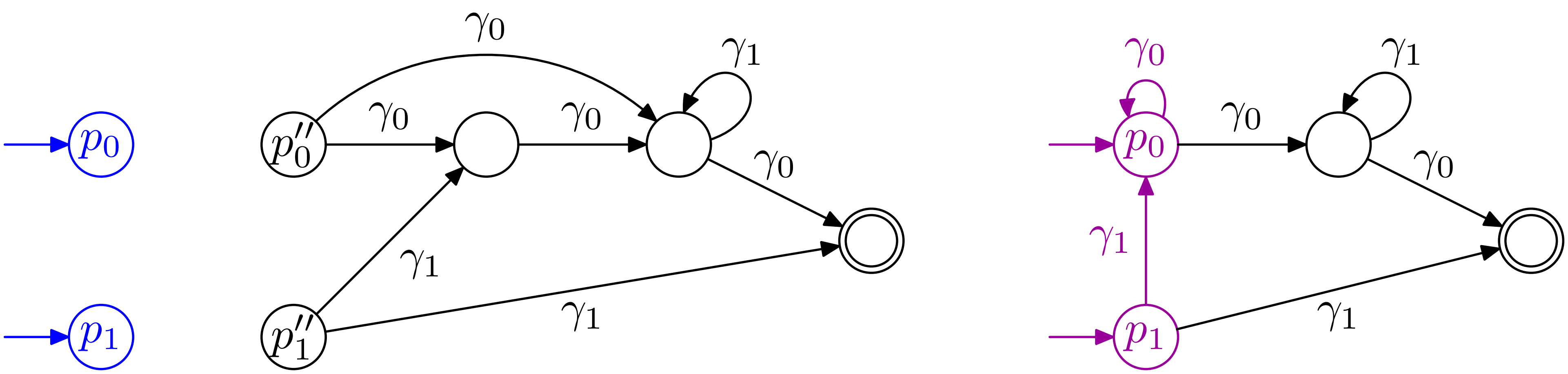




An acceleration for pushdown automata

Idea: reuse the same states

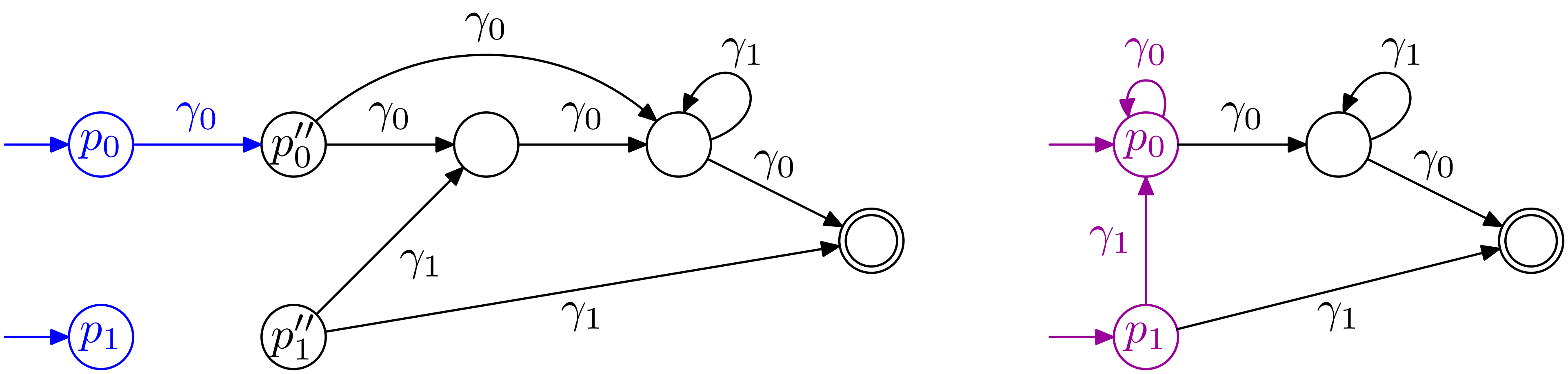
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



An acceleration for pushdown automata

Idea: reuse the same states

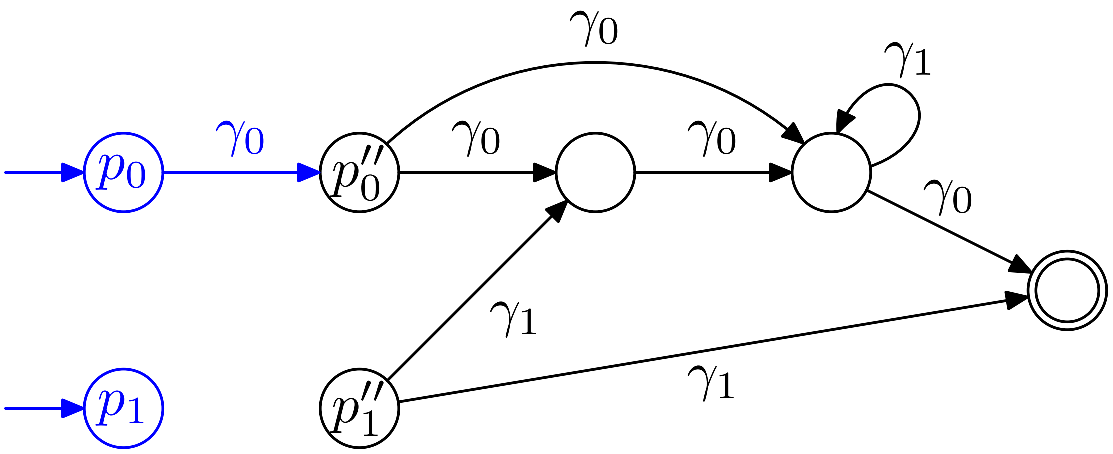
$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

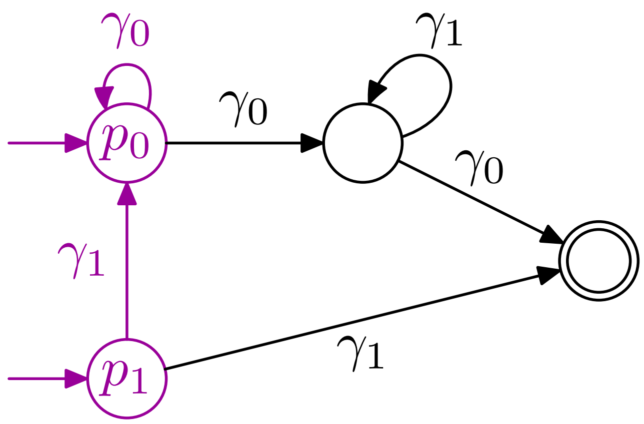


An acceleration for pushdown automata

Idea: reuse the same states

$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

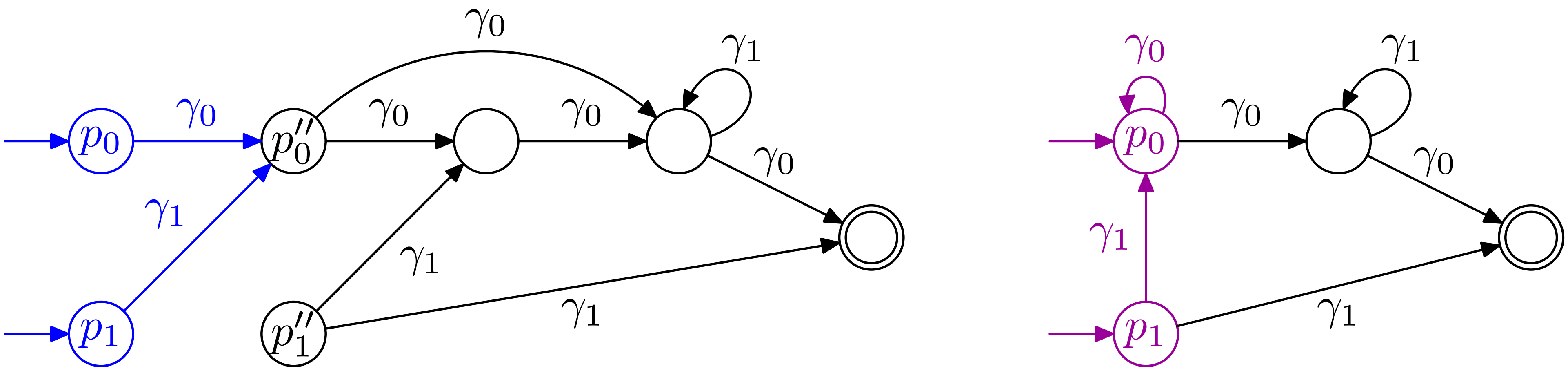




An acceleration for pushdown automata

Idea: reuse the same states

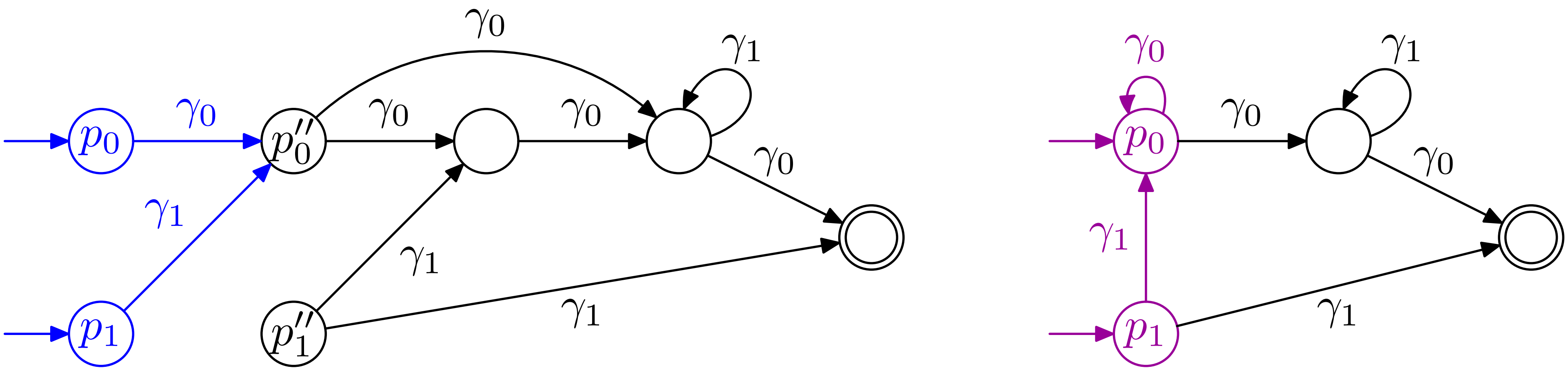
$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$



An acceleration for pushdown automata

Idea: reuse the same states

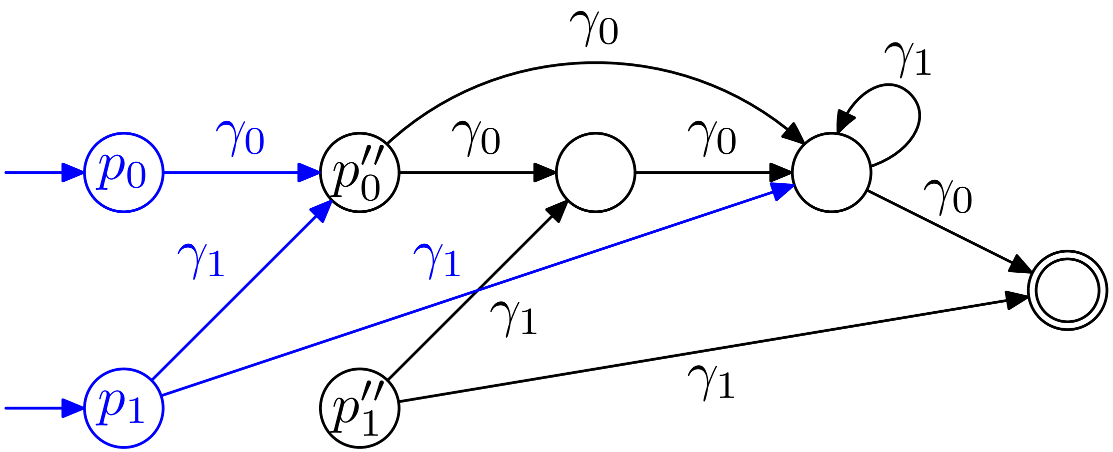
$$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle$$

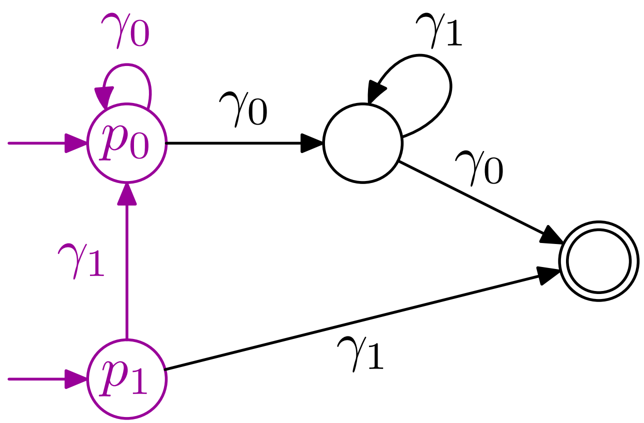


An acceleration for pushdown automata

Idea: reuse the same states

$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1\gamma_0 \rangle$

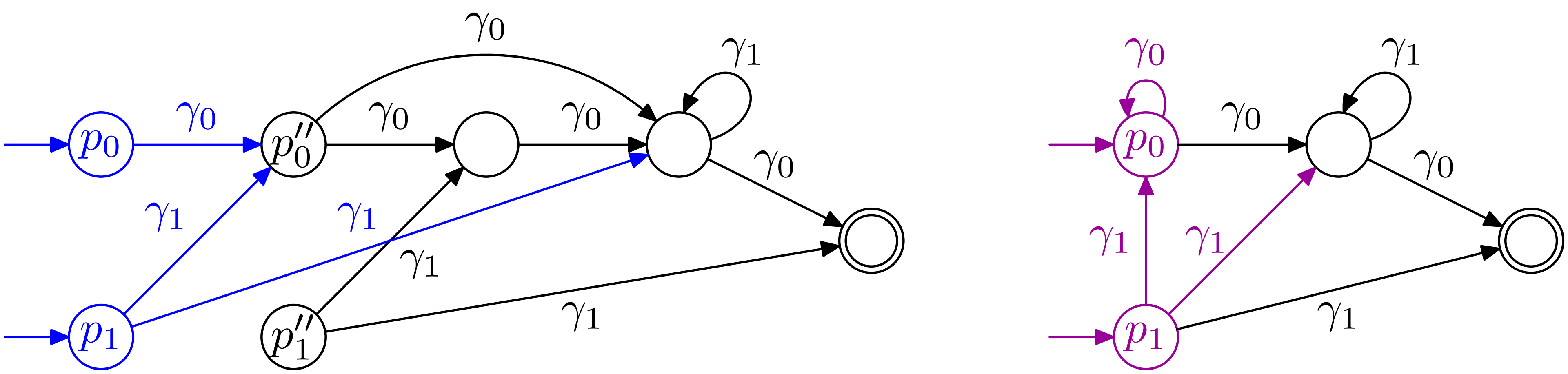




An acceleration for pushdown automata

Idea: reuse the same states

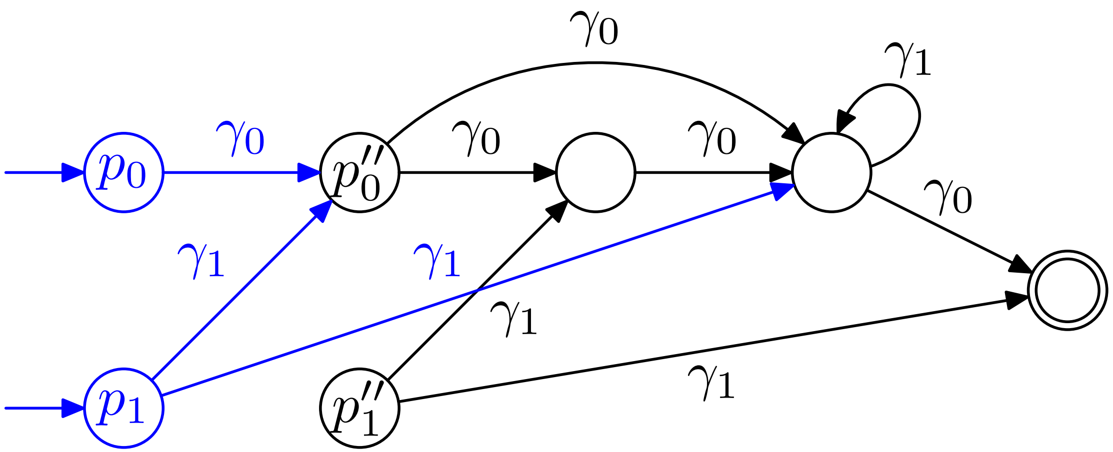
$\langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1\gamma_0 \rangle$

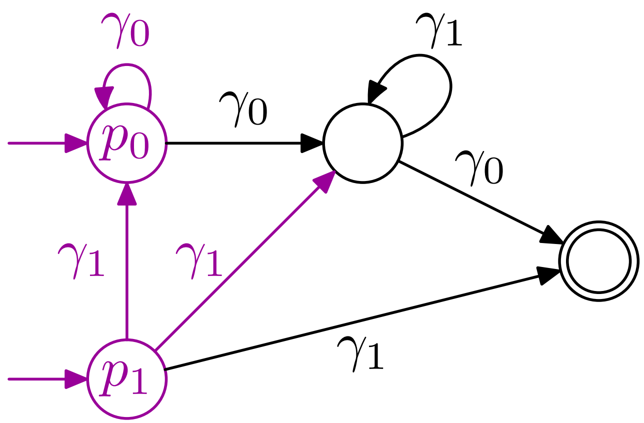


An acceleration for pushdown automata

Idea: reuse the same states

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$

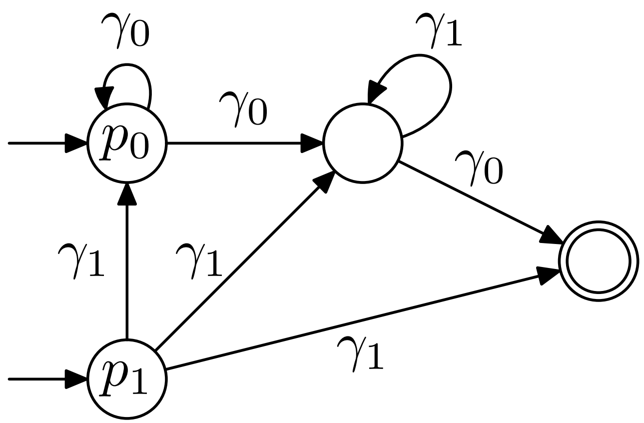




An acceleration for pushdown automata

Idea: reuse the same states

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \; , \; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \; , \; \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \, \rangle \}$$



But does it work . . . ?

All predecessors are computed, and termination guaranteed

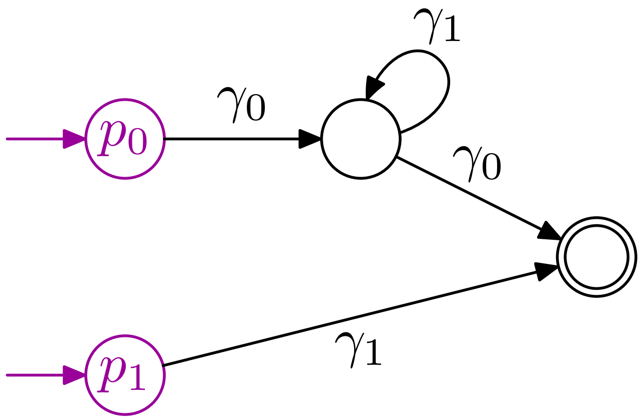
But: we might be adding non-predecessors

But does it work . . . ?

All predecessors are computed, and termination guaranteed

But: we might be adding non-predecessors

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \ , \ \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$

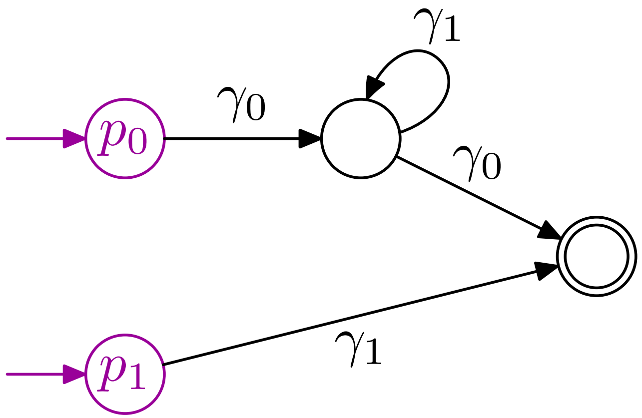


But does it work . . . ?

All predecessors are computed, and termination guaranteed

But: we might be adding non-predecessors

$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

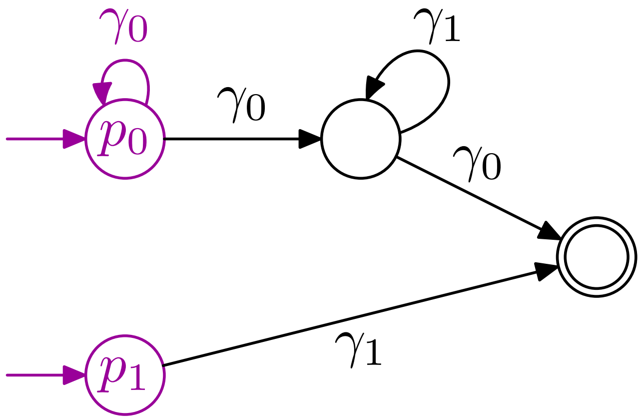


But does it work . . . ?

All predecessors are computed, and termination guaranteed

But: we might be adding non-predecessors

$$\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$$

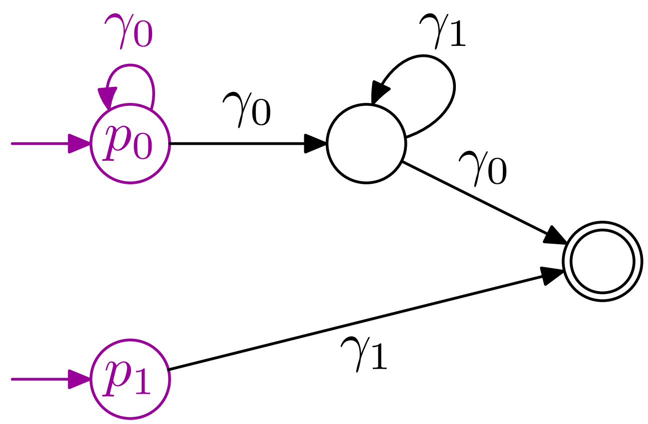


But does it work . . . ?

All predecessors are computed, and termination guaranteed

But: we might be adding non-predecessors

$$\Delta = \{ \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle \, , \, \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_0 \rangle \}$$



Fortunately: correct if initial states have no incoming arcs.

The proof (1/4)

Input: Pushdown automaton (P, Γ, Δ), NFA $\mathcal{A} = (Q, \Gamma, \rightarrow_0, P, F)$ recognizing a regular set C.

Precondition: No transition of $\mathcal{A}$ leads to an initial state.

Output: NFA $\mathcal{A}_{pre^*} = (Q, \Gamma, \rightarrow, P, F)$.

Postcondition: $\mathcal{A}_{pre^*}$ recognizes $pre^*(C)$.

Algorithm: Add new transitions according to the following saturation rule

If $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ and $p' \xrightarrow{w} q$ in the current automaton, add a transition (p, γ, q).

The proof (2/4)

Goal: show that $\mathcal{A}_{pre^*}$ only recognizes words of *pre**(*C*).

(Showing that it recognizes all words of *pre**(*C*) is easy.)

The proof (2/4)

Goal: show that $\mathcal{A}_{pre^*}$ only recognizes words of $pre^*(C)$.

(Showing that it recognizes all words of $pre^*(C)$ is easy.)

Notation: $\xrightarrow[i]{}$ denotes the transition relation after adding i transitions to $\mathcal{A}$.

The proof (2/4)

Goal: show that $\mathcal{A}_{pre^*}$ only recognizes words of $pre^*(C)$.

(Showing that it recognizes all words of $pre^*(C)$ is easy.)

Notation: $\xrightarrow[i]{}$ denotes the transition relation after adding i transitions to $\mathcal{A}$.

We show: If $p \xrightarrow[i]{w} q$, then $\langle p, w\rangle \Rightarrow^* \langle p', w'\rangle$ for some $\langle p', w'\rangle$ such that $p' \xrightarrow[0]{w'} q$; moreover, if q initial, then $w' = \epsilon$.

The proof (2/4)

Goal: show that $\mathcal{A}_{pre^*}$ only recognizes words of $pre^*(C)$.

(Showing that it recognizes all words of $pre^*(C)$ is easy.)

Notation: $\xrightarrow[i]{}$ denotes the transition relation after adding i transitions to $\mathcal{A}$.

We show: If $p \xrightarrow[i]{w} q$, then $\langle p, w\rangle \Rightarrow^* \langle p', w'\rangle$ for some $\langle p', w'\rangle$ such that $p' \xrightarrow[0]{w'} q$; moreover, if q initial, then $w' = \epsilon$.

Proof by induction on i. Basis $i = 1$ is easy.

The proof (2/4)

Goal: show that $\mathcal{A}_{pre^*}$ only recognizes words of $pre^*(C)$.

(Showing that it recognizes all words of $pre^*(C)$ is easy.)

Notation: $\xrightarrow[i]{}$ denotes the transition relation after adding i transitions to $\mathcal{A}$.

We show: If $p \xrightarrow[i]{w} q$, then $\langle p, w \rangle \Rightarrow^* \langle p', w' \rangle$ for some $\langle p', w' \rangle$ such that $p' \xrightarrow[0]{w'} q$; moreover, if q initial, then $w' = \epsilon$.

Proof by induction on i. Basis $i = 1$ is easy.

$i > 1$. Let (p_1, γ, q') be the i-th transition added to $\mathcal{A}$ (p_1 initial state!).

The proof (2/4)

Goal: show that $\mathcal{A}_{pre^*}$ only recognizes words of $pre^*(C)$.

(Showing that it recognizes all words of $pre^*(C)$ is easy.)

Notation: $\xrightarrow[i]{}$ denotes the transition relation after adding i transitions to $\mathcal{A}$.

We show: If $p \xrightarrow[i]{w} q$, then $\langle p, w \rangle \Rightarrow^* \langle p', w' \rangle$ for some $\langle p', w' \rangle$ such that $p' \xrightarrow[0]{w'} q$; moreover, if q initial, then $w' = \epsilon$.

Proof by induction on i. Basis $i = 1$ is easy.

$i > 1$. Let (p_1, γ, q') be the i-th transition added to $\mathcal{A}$ (p_1 initial state!).
Let j be the number of times that (p_1, γ, q') is used in $p \xrightarrow[i]{w} q$.

The proof (2/4)

Goal: show that $\mathcal{A}_{pre^*}$ only recognizes words of $pre^*(C)$.

(Showing that it recognizes all words of $pre^*(C)$ is easy.)

Notation: $\xrightarrow[i]{}$ denotes the transition relation after adding i transitions to $\mathcal{A}$.

We show: If $p \xrightarrow[i]{w} q$, then $\langle p, w \rangle \Rightarrow^* \langle p', w' \rangle$ for some $\langle p', w' \rangle$ such that $p' \xrightarrow[0]{w'} q$; moreover, if q initial, then $w' = \epsilon$.

Proof by induction on i. Basis $i = 1$ is easy.

$i > 1$. Let (p_1, γ, q') be the i-th transition added to $\mathcal{A}$ (p_1 initial state!).
Let j be the number of times that (p_1, γ, q') is used in $p \xrightarrow[i]{w} q$.
By induction on j. Basis $j = 0$ is easy.

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1) $\quad p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q \quad$ (by 'zooming into' $p \xrightarrow[i]{w} q$)

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by 'zooming into' $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by ‘zooming into’ $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)
(4)	$\langle p, u \rangle \Rightarrow^* \langle p_1, \varepsilon \rangle$	(by induction hypothesis on i)

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by 'zooming into' $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)
(4)	$\langle p, u \rangle \Rightarrow^* \langle p_1, \varepsilon \rangle$	(by induction hypothesis on i)
(5)	$\langle p_2, w_2 v \rangle \Rightarrow^* \langle p', w' \rangle$	
(6)	$p' \xrightarrow[0]{w'} q$	(by induction hypothesis on j)

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by ‘zooming into’ $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)
(4)	$\langle p, u \rangle \Rightarrow^* \langle p_1, \varepsilon \rangle$	(by induction hypothesis on i)
(5)	$\langle p_2, w_2 v \rangle \Rightarrow^* \langle p', w' \rangle$	
(6)	$p' \xrightarrow[0]{w'} q$	(by induction hypothesis on j)

$$\langle p, w \rangle \underset{(1)}{=} \langle p, u\gamma v \rangle$$

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by 'zooming into' $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)
(4)	$\langle p, u \rangle \Rightarrow^* \langle p_1, \varepsilon \rangle$	(by induction hypothesis on i)
(5)	$\langle p_2, w_2 v \rangle \Rightarrow^* \langle p', w' \rangle$	
(6)	$p' \xrightarrow[0]{w'} q$	(by induction hypothesis on j)

$$\langle p, w \rangle \underset{(1)}{=} \langle p, u\gamma v \rangle \underset{(4)}{\Longrightarrow^*} \langle p_1, \gamma v \rangle$$

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by 'zooming into' $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)
(4)	$\langle p, u \rangle \Rightarrow^* \langle p_1, \varepsilon \rangle$	(by induction hypothesis on i)
(5)	$\langle p_2, w_2 v \rangle \Rightarrow^* \langle p', w' \rangle$	
(6)	$p' \xrightarrow[0]{w'} q$	(by induction hypothesis on j)

$$\langle p, w \rangle \underset{(1)}{=} \langle p, u\gamma v \rangle \underset{(4)}{\Longrightarrow^*} \langle p_1, \gamma v \rangle \underset{(2)}{\Longrightarrow} \langle p_2, w_2 v \rangle$$

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by ‘zooming into’ $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)
(4)	$\langle p, u \rangle \Rightarrow^* \langle p_1, \varepsilon \rangle$	(by induction hypothesis on i)
(5)	$\langle p_2, w_2 v \rangle \Rightarrow^* \langle p', w' \rangle$	
(6)	$p' \xrightarrow[0]{w'} q$	(by induction hypothesis on j)

$$\langle p, w \rangle \underset{(1)}{=} \langle p, u\gamma v \rangle \underset{(4)}{\Longrightarrow^*} \langle p_1, \gamma v \rangle \underset{(2)}{\Longrightarrow} \langle p_2, w_2 v \rangle \underset{(5)}{\Longrightarrow^*} \langle p', w' \rangle$$

The proof (3/4)

Step. $j > 0$. So (p_1, γ, q') occurs in $p \xrightarrow[i]{w} q$. We have:

(1)	$p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{\gamma} q' \xrightarrow[i]{v} q$	(by 'zooming into' $p \xrightarrow[i]{w} q$)
(2)	$\langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle$	
(3)	$p_2 \xrightarrow[i-1]{w_2} q' \xrightarrow[i]{v} q$	(by the saturation rule)
(4)	$\langle p, u \rangle \Rightarrow^* \langle p_1, \varepsilon \rangle$	(by induction hypothesis on i)
(5)	$\langle p_2, w_2 v \rangle \Rightarrow^* \langle p', w' \rangle$	
(6)	$p' \xrightarrow[0]{w'} q$	(by induction hypothesis on j)

$$\langle p, w \rangle \underset{(1)}{=} \langle p, u\gamma v \rangle \underset{(4)}{\Longrightarrow^*} \langle p_1, \gamma v \rangle \underset{(2)}{\Longrightarrow} \langle p_2, w_2 v \rangle \underset{(5)}{\Longrightarrow^*} \langle p', w' \rangle$$

Finally, if q initial then $w' = \epsilon$ because of (6) and precondition.

Forward search and complexity

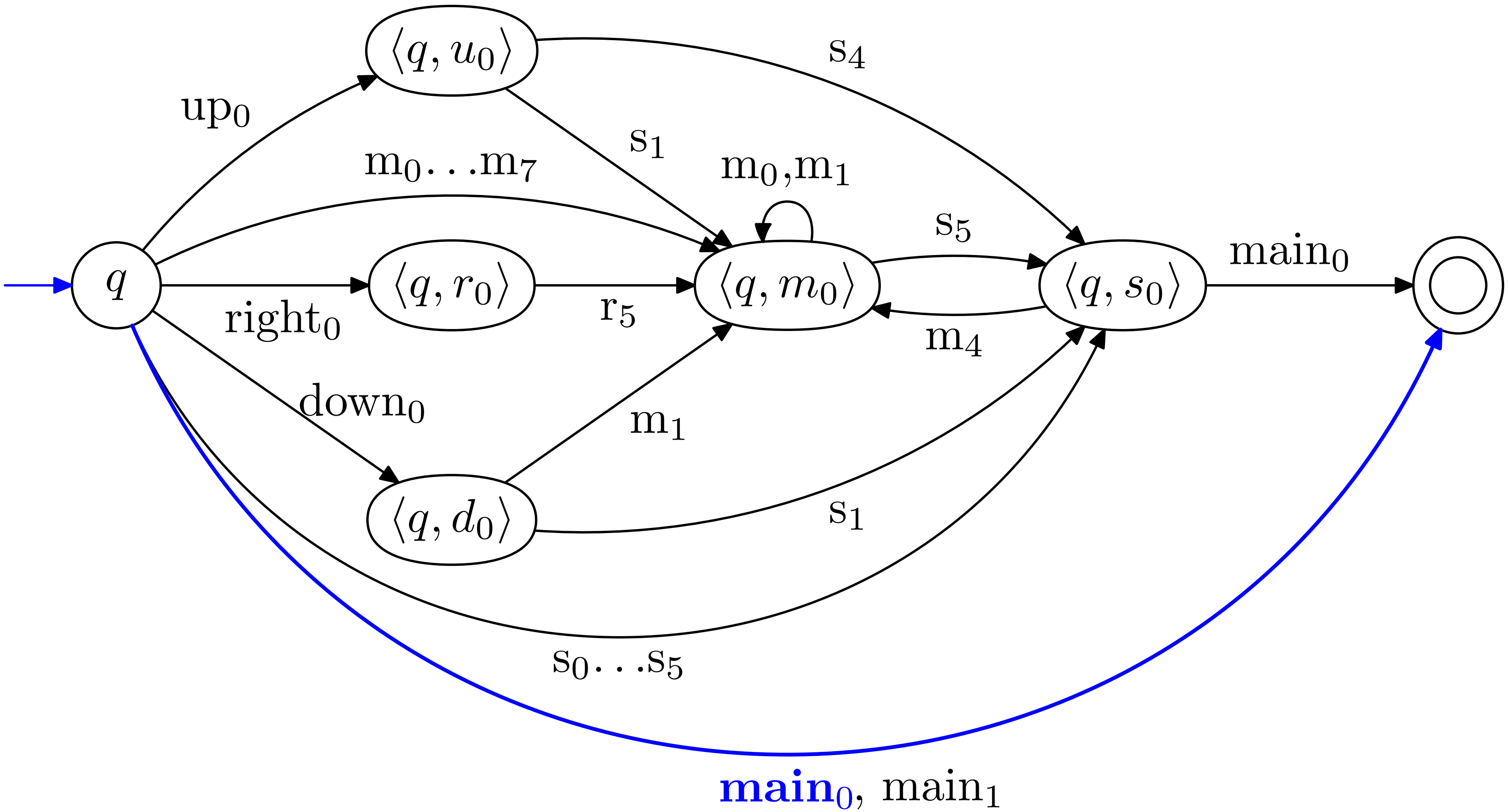
Symbolic forward search with regular sets can be accelerated in a similar way

Recall input: Pushdown automaton (P, Γ, Δ), NFA $\mathcal{A} = (Q, \Gamma, \rightarrow_0, P, F)$.

Complexity of backward search: $O(|Q|^2 \cdot |\Delta|)$ time, $O(|Q| \cdot |\Delta| + |\rightarrow_0|)$ space.

Complexity of forward search: $O(|P| \cdot |\Delta| \cdot (|Q \setminus P| + |\Delta|) + |P| \cdot |\rightarrow_0|)$ time and space.

Reachable configurations of the plotter program



Repeated reachability for pushdown systems

Let $I = \langle p_0, \gamma_0 \rangle$ and $D = \langle p, \Gamma^* \rangle$.

D can be repeatedly reached from I iff

$$\langle p_0, \gamma_0 \rangle \longrightarrow^* \langle p', \gamma w \rangle$$

and

$$\langle p', \gamma \rangle \longrightarrow^* \langle p, v \rangle \longrightarrow^* \langle p', \gamma u \rangle$$

for some p', γ, w, v, u.

Repeated reachability can be reduced to computing several pre^*.

To know more

Pushdown automata usually called pushdown processes in our context.

They are equivalent to recursive state machines.

The class of one-state PDAs is interesting, usually studied under the name Basic Process Algebra(BPA) or context-free processes

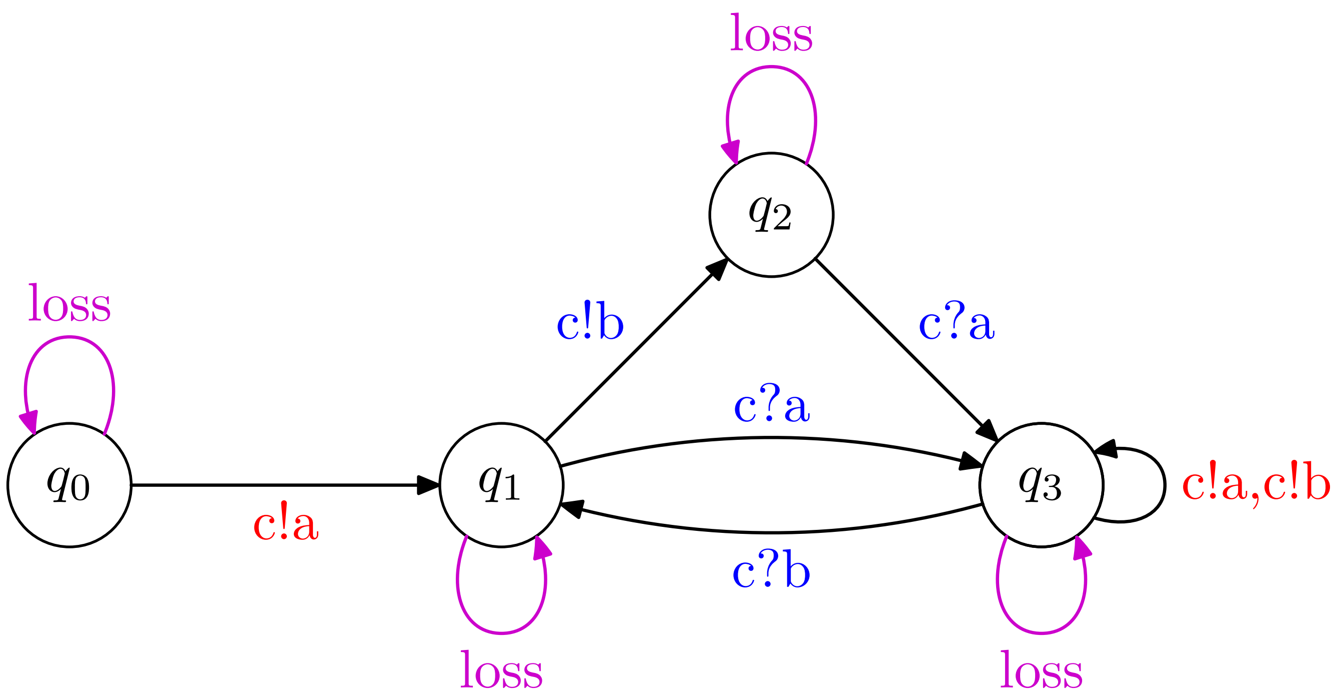
Some people: Alur, Baeten, Bouajjani, Caucal, E., Etessami, Schwoon, Steffen, Stirling, Yannakakis, Walukiewicz . . .

Tools: Moped, available online at
http://www.informatik.uni-stuttgart.de/fmi/szs/tools/moped/

Technology transfer: the Static Driver Verifier (Microsoft)
see http://www.microsoft.com/whdc/devtools/tools/sdv.mspx

(Lossy) Channel Systems

(Lossy) Channel systems



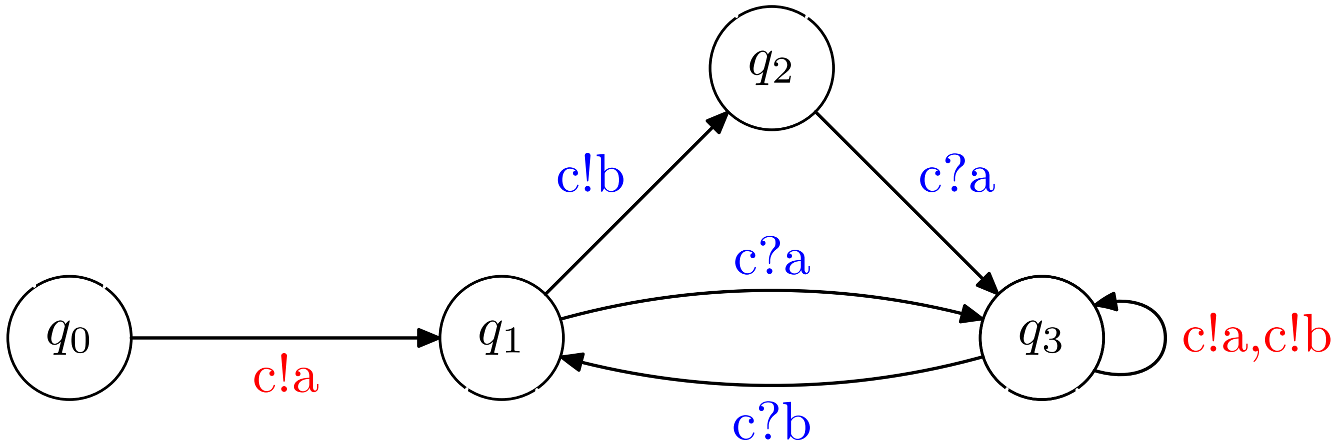
Automata extended with channels (unbounded queues)

Send transitions: no guard, action sends message to the channel.

Receive transitions: guard checks if the channel is nonempty, action removes the first message.

Loss transitions: self-loops, no guard, action removes an arbitrary message.

(Lossy) Channel systems



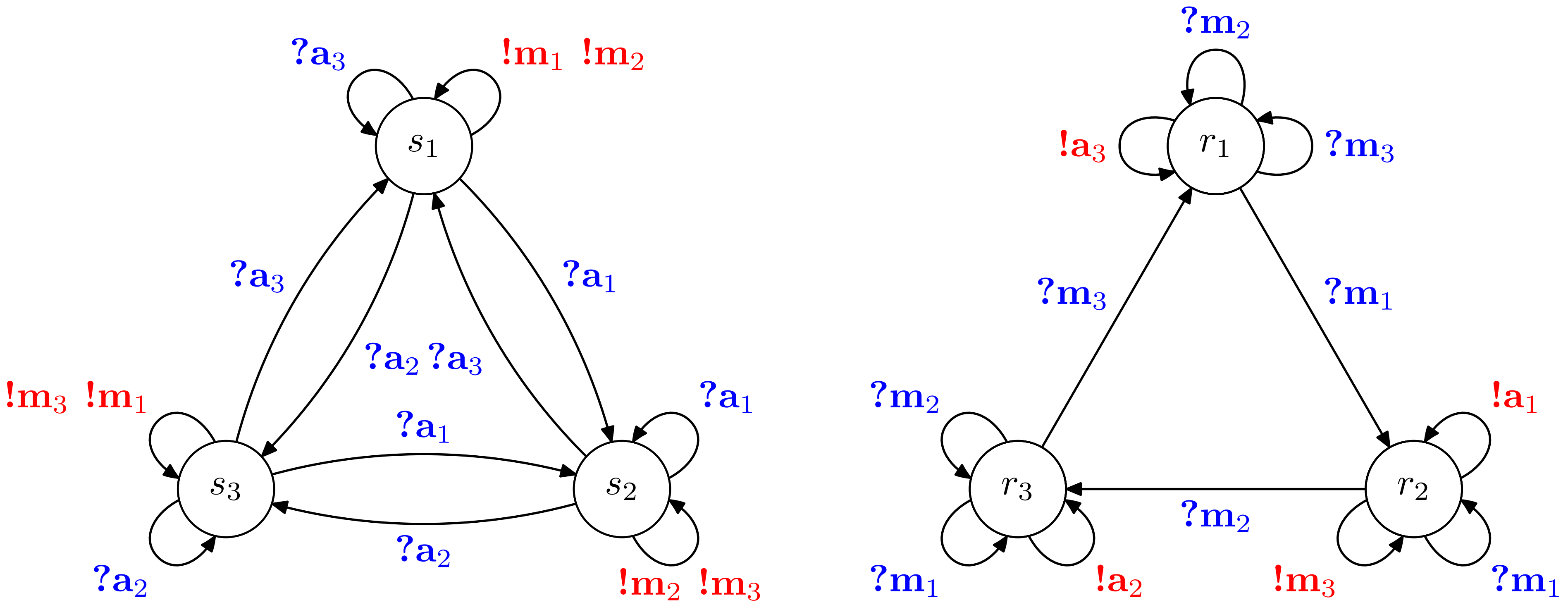
Automata extended with channels (unbounded queues)

Send transitions: no guard, action sends message to the channel.

Receive transitions: guard checks if the channel is nonempty, action removes the first message.

Loss transitions: self-loops, no guard, action removes an arbitrary message.

Case study: A sliding window protocol



Symbolic reachability for (lossy) channel systems

Perfect channels: Turing powerful model, even with only one channel.

Lossy channels:

- Backward search: decidable for D upward-closed set
- Forward search: Choose $\mathcal{C}$ as the set of simple regular expressions (SREs).

$$\begin{aligned} \text{Atomic expression:} \quad & (a+\epsilon) \mid (a_1 + \ldots + a_m)^* \\ \text{Product:} \quad & e_1 e_2 \ldots e_n \\ \text{SRE:} \quad & p_1 + \ldots + p_n \end{aligned}$$

SREs satisfy conditions (1)-(5) (exercise), but not (6).
The fixpoint is an SRE, but it cannot be effectively computed (!), and so no ‘perfect’ acceleration can exist.

Acceleration through loops

Compute a symbolic reachability graph with elements of $\mathcal{C}$ as nodes:

- Add I as first node
- For each node C and each transition t, add an edge $C \xrightarrow{t} post[t](C)$

Acceleration through loops

Compute a symbolic reachability graph with elements of $\mathcal{C}$ as nodes:

- Add I as first node
- For each node C and each transition t, add an edge $C \xrightarrow{t} post[t](C)$

Replace $C \xrightarrow{\sigma} post[\sigma](C)$ by $C \xrightarrow{\sigma} X$, where X satisfies

- $post[\sigma](C) \subseteq X$, and
- X contains only reachable configurations.

Acceleration through loops

Compute a symbolic reachability graph with elements of $\mathcal{C}$ as nodes:

- Add I as first node
- For each node C and each transition t, add an edge $C \xrightarrow{t} post[t](C)$

Replace $C \xrightarrow{\sigma} post[\sigma](C)$ by $C \xrightarrow{\sigma} X$, where X satisfies

- $post[\sigma](C) \subseteq X$, and
- X contains only reachable configurations.

A loop is a sequence of transitions leading from a control state to itself.

Acceleration: given a loop $C \xrightarrow{\sigma} post[\sigma](C)$, replace $post[\sigma](C)$ by

$$X = post[\sigma^*](C) = C \cup post[\sigma](C) \cup post[\sigma^2](C) \cup \ldots$$

Acceleration through loops

Compute a symbolic reachability graph with elements of $\mathcal{C}$ as nodes:

- Add I as first node
- For each node C and each transition t, add an edge $C \xrightarrow{t} post[t](C)$

Replace $C \xrightarrow{\sigma} post[\sigma](C)$ by $C \xrightarrow{\sigma} X$, where X satisfies

- $post[\sigma](C) \subseteq X$, and
- X contains only reachable configurations.

A loop is a sequence of transitions leading from a control state to itself.

Acceleration: given a loop $C \xrightarrow{\sigma} post[\sigma](C)$, replace $post[\sigma](C)$ by

$$X = post[\sigma^*](C) = C \cup post[\sigma](C) \cup post[\sigma^2](C) \cup \ldots$$

Question: find a suitable class of loops such that $post[\sigma^*](C)$ belongs to $\mathcal{C}$.

An acceleration for lossy channel systems

Theorem [Abdulla, Bouajjani, Jonsson, CAV'98]: For any loop σ of a lossy channel system and any SRE r, the set $post[\sigma^*](r)$ is an SRE that can be computed in quadratic time in the size of r.

An acceleration for lossy channel systems

Theorem [Abdulla, Bouajjani, Jonsson, CAV'98]: For any loop σ of a lossy channel system and any SRE r, the set $post[\sigma^*](r)$ is an SRE that can be computed in quadratic time in the size of r.

Use in verification:

An acceleration for lossy channel systems

Theorem [Abdulla, Bouajjani, Jonsson, CAV'98]: For any loop σ of a lossy channel system and any SRE r, the set $post[\sigma^*](r)$ is an SRE that can be computed in quadratic time in the size of r.

Use in verification:

Preselect a set of loops (e.g., those corresponding to simple cycles).

An acceleration for lossy channel systems

Theorem [Abdulla, Bouajjani, Jonsson, CAV'98]: For any loop σ of a lossy channel system and any SRE r, the set $post[\sigma^*](r)$ is an SRE that can be computed in quadratic time in the size of r.

Use in verification:

Preselect a set of loops (e.g., those corresponding to simple cycles).

Given a set of configurations, compute first the effect of executing each of the loops infinitely often, and then compute for each transition the effect of computing it.

An acceleration for lossy channel systems

Theorem [Abdulla, Bouajjani, Jonsson, CAV'98]: For any loop σ of a lossy channel system and any SRE r, the set $post[\sigma^*](r)$ is an SRE that can be computed in quadratic time in the size of r.

Use in verification:

- Preselect a set of loops (e.g., those corresponding to simple cycles).
- Given a set of configurations, compute first the effect of executing each of the loops infinitely often, and then compute for each transition the effect of computing it.
- Pray for termination.

Channel contents of the sliding window protocol

States	Mess. channel	Ack. channel
s_1, r_1	$(m_2 + m_3)^*(m_1 + m_3)^*(m_1 + m_2)^*$	a_3^*
s_1, r_2	$(m_1 + m_3)^*(m_1 + m_2)^*$	$a_3^* a_1^*$
s_1, r_3	$(m_1 + m_2)^*$	$a_3^* a_1^* a_2^*$
s_2, r_1	$(m_2 + m_3)^*$	$a_1^* a_2^* a_3^*$
s_2, r_2	$(m_1 + m_3)^*(m_1 + m_2)^*(m_2 + m_3)^*$	a_1^*
s_2, r_3	$(m_1 + m_2)^*(m_2 + m_3)^*$	$a_1^* a_2^*$
s_3, r_1	$(m_2 + m_3)^*(m_1 + m_3)^*$	$a_1^* a_2^*$
s_3, r_2	$(m_1 + m_3)^*$	$a_2^* a_3^* a_1^*$
s_3, r_3	$(m_1 + m_2)^*(m_2 + m_3)^*(m_1 + m_3)^*$	a_2^*

The learning approach

Problem of accelerations: your prayers may not be heard.

The learning approach

Problem of accelerations: your prayers may not be heard.

No results characterizing the cases for which they will be.
(The ways of God are inscrutable).

The learning approach

Problem of accelerations: your prayers may not be heard.

No results characterizing the cases for which they will be.
(The ways of God are inscrutable).

Recent alternative [Vardhan, Sen, Viswanathan, Agha, FSTTCS '04]:
apply learning algorithms for regular languages.

Angluin's learning setting [I&C '87]

Two agents, the Teacher and the Learner.

The Teacher knows a regular language $L \subseteq \Sigma^*$.

The Learner knows Σ and wants to learn L.

Angluin’s learning setting [I&C ’87]

Two agents, the Teacher and the Learner.

The Teacher knows a regular language $L \subseteq \Sigma^*$.

The Learner knows Σ and wants to learn L.

The Learner is only allowed to ask the Teacher two types of questions:

> Membership queries: The Learner produces $w \in \Sigma^*$, and asks if $w \in L$.
> The Teacher answers yes/no.

Angluin's learning setting [I&C '87]

Two agents, the Teacher and the Learner.

The Teacher knows a regular language $L \subseteq \Sigma^*$.

The Learner knows Σ and wants to learn L.

The Learner is only allowed to ask the Teacher two types of questions:

> Membership queries: The Learner produces $w \in \Sigma^*$, and asks if $w \in L$.
> The Teacher answers yes/no.
>
> Equivalence queries: The Learner produces a regular language $H \subseteq \Sigma^*$ (a hypothesis), and asks if $L = H$.
> The Teacher answers either yes or no + counterexample (a word in the symmetric difference of L and H).

Angluin's learning setting [I&C '87]

Two agents, the Teacher and the Learner.

The Teacher knows a regular language $L \subseteq \Sigma^*$.

The Learner knows Σ and wants to learn L.

The Learner is only allowed to ask the Teacher two types of questions:

> Membership queries: The Learner produces $w \in \Sigma^*$, and asks if $w \in L$.
> The Teacher answers yes/no.
>
> Equivalence queries: The Learner produces a regular language $H \subseteq \Sigma^*$ (a hypothesis), and asks if $L = H$.
> The Teacher answers either yes or no $+$ counterexample (a word in the symmetric difference of L and H).

Question: give an algorithm (a strategy) for the Learner.

Structure of Angluin's algorithm

The Learner repeatedly asks membership queries until it has enough information to state a hypothesis.

Structure of Angluin's algorithm

The Learner repeatedly asks membership queries until it has enough information to state a hypothesis.

The hypothesis $H_1, H_2, H_3, \ldots$ are presented as minimal DFAs.

Structure of Angluin's algorithm

The Learner repeatedly asks membership queries until it has enough information to state a hypothesis.

The hypothesis $H_1, H_2, H_3, \ldots$ are presented as minimal DFAs.

The hypothesis satisfy $n_1 < n_2 < n_3 < \ldots$, where n_i denotes the number of states for H_i.

Structure of Angluin's algorithm

The Learner repeatedly asks membership queries until it has enough information to state a hypothesis.

The hypothesis $H_1, H_2, H_3, \ldots$ are presented as minimal DFAs.

The hypothesis satisfy $n_1 < n_2 < n_3 < \ldots$, where n_i denotes the number of states for H_i.

For every hypothesis H, either $H = L$ or the minimal DFA for H has fewer states than the minimal DFA for L.

Structure of Angluin's algorithm

The Learner repeatedly asks membership queries until it has enough information to state a hypothesis.

The hypothesis $H_1, H_2, H_3, \ldots$ are presented as minimal DFAs.

The hypothesis satisfy $n_1 < n_2 < n_3 < \ldots$, where n_i denotes the number of states for H_i.

For every hypothesis H, either $H = L$ or the minimal DFA for H has fewer states than the minimal DFA for L.

If $H \neq L$, then the Learner uses the counterexample returned by the Teacher to generate a new round of membership queries.

Structure of Angluin's algorithm

The Learner repeatedly asks membership queries until it has enough information to state a hypothesis.

The hypothesis $H_1, H_2, H_3, \ldots$ are presented as minimal DFAs.

The hypothesis satisfy $n_1 < n_2 < n_3 < \ldots$, where n_i denotes the number of states for H_i.

For every hypothesis H, either $H = L$ or the minimal DFA for H has fewer states than the minimal DFA for L.

If $H \neq L$, then the Learner uses the counterexample returned by the Teacher to generate a new round of membership queries.

Completeness: the Learner eventually produces L as hypothesis.

Complexity: polynomial in the size of the minimal DFA for L.

Learning for (lossy) channel systems

First attempt: Learn the language of reachable configurations, under the assumption that it is regular.

.

Learning for (lossy) channel systems

First attempt: Learn the language of reachable configurations, under the assumption that it is regular.

Does not work: answering a membership query is equivalent to solving the reachability problem, and answering equivalence queries is equivalent to the problem we wish to solve!

.

Learning for (lossy) channel systems

First attempt: Learn the language of reachable configurations, under the assumption that it is regular.

Does not work: answering a membership query is equivalent to solving the reachability problem, and answering equivalence queries is equivalent to the problem we wish to solve!

Second attempt:

.

Learning for (lossy) channel systems

First attempt: Learn the language of reachable configurations, under the assumption that it is regular.

Does not work: answering a membership query is equivalent to solving the reachability problem, and answering equivalence queries is equivalent to the problem we wish to solve!

Second attempt:

> Define an execution as a pair (σ, c) where c is a configuration and σ is a witness, i.e., a sequence of transitions that can be executed from some initial configuration and whose execution leads to c.

.

Learning for (lossy) channel systems

First attempt: Learn the language of reachable configurations, under the assumption that it is regular.

Does not work: answering a membership query is equivalent to solving the reachability problem, and answering equivalence queries is equivalent to the problem we wish to solve!

Second attempt:

> Define an execution as a pair (σ, c) where c is a configuration and σ is a witness, i.e., a sequence of transitions that can be executed from some initial configuration and whose execution leads to c.
>
> Learn the language *Exec* of all executions, under the assumption that it is regular.

.

Learning for (lossy) channel systems

First attempt: Learn the language of reachable configurations, under the assumption that it is regular.

Does not work: answering a membership query is equivalent to solving the reachability problem, and answering equivalence queries is equivalent to the problem we wish to solve!

Second attempt:

Define an execution as a pair (σ, c) where c is a configuration and σ is a witness, i.e., a sequence of transitions that can be executed from some initial configuration and whose execution leads to c.

Learn the language *Exec* of all executions, under the assumption that it is regular.

Membership queries: easy, simulate σ and check it leads to c.

.

Learning for (lossy) channel systems

First attempt: Learn the language of reachable configurations, under the assumption that it is regular.

Does not work: answering a membership query is equivalent to solving the reachability problem, and answering equivalence queries is equivalent to the problem we wish to solve!

Second attempt:

> Define an execution as a pair (σ, c) where c is a configuration and σ is a witness, i.e., a sequence of transitions that can be executed from some initial configuration and whose execution leads to c.
>
> Learn the language *Exec* of all executions, under the assumption that it is regular.
>
> Membership queries: easy, simulate σ and check it leads to c.
>
> . . . but equivalence queries still hopeless.

Third attempt

Define a marked transition sequence (MTS) as a pair (σ, c), where σ is a sequence of transition names and c is a configuration.
Notice that executions are MTS.

Third attempt

Define a marked transition sequence (MTS) as a pair (σ, c), where σ is a sequence of transition names and c is a configuration.
Notice that executions are MTS.

Don't learn *Exec*, just decide whether $\mathit{Exec} \cap D = \emptyset$ for a given regular set D of dangerous MTSs.

Third attempt

Define a marked transition sequence (MTS) as a pair(σ, c), where σ is a sequence of transition names and c is a configuration.
Notice that executions are MTS.

Don't learn *Exec*, just decide whether $\mathit{Exec} \cap D = \emptyset$ for a given regular set D of dangerous MTSs.

Adapt Angluin's algorithm to learn either

(DE) a dangerous execution, or

(SS) a safe superset of *Exec*, i.e., one containing no dangerous executions.

Third attempt

Define a marked transition sequence (MTS) as a pair(σ, c), where σ is a sequence of transition names and c is a configuration.
Notice that executions are MTS.

Don't learn Exec, just decide whether $\mathit{Exec} \cap D = \emptyset$ for a given regular set D of dangerous MTSs.

Adapt Angluin's algorithm to learn either

(DE) a dangerous execution, or

(SS) a safe superset of Exec, i.e., one containing no dangerous executions.

Membership queries: as in the previous attempt, but if a dangerous execution is found, the Learner has learned DE, and the algorithm stops.

Third attempt

Define a marked transition sequence (MTS) as a pair(σ, c), where σ is a sequence of transition names and c is a configuration.
Notice that executions are MTS.

Don't learn *Exec*, just decide whether $\mathit{Exec} \cap D = \emptyset$ for a given regular set D of dangerous MTSs.

Adapt Angluin's algorithm to learn either

(DE) a dangerous execution, or

(SS) a safe superset of *Exec*, i.e., one containing no dangerous executions.

Membership queries: as in the previous attempt, but if a dangerous execution is found, the Learner has learned DE, and the algorithm stops.

Replace equivalence queries by containment queries.

Containment queries

Containment queries: the Learner produces a regular hypothesis H, and asks the Teacher whether $H \supseteq Exec$ and, if so, whether $H \cap D = \emptyset$.

If the Teacher answers

Containment queries

Containment queries: the Learner produces a regular hypothesis H, and asks the Teacher whether $H \supseteq Exec$ and, if so, whether $H \cap D = \emptyset$.
If the Teacher answers

1. $H \supseteq Exec$ and $H \cap D = \emptyset$, the Learner has learned a SS, stop.

Containment queries

Containment queries: the Learner produces a regular hypothesis H, and asks the Teacher whether $H \supseteq Exec$ and, if so, whether $H \cap D = \emptyset$.
If the Teacher answers

1. $H \supseteq Exec$ and $H \cap D = \emptyset$, the Learner has learned a SS, stop.
2. $H \supseteq Exec$ and $H \cap D \neq \emptyset$, then the Teacher returns $(\sigma, c) \in H \cap D$.
   The Learner checks whether $(\sigma, c) \in Exec$:
   2.1. if $(\sigma, c) \in Exec$, then the Learner has learned a DE, stop;
   2.2. if $(\sigma, c) \notin Exec$, then $(\sigma, c) \in H \oplus Exec$, and so the Learner has got a counterexample.

Containment queries

Containment queries: the Learner produces a regular hypothesis H, and asks the Teacher whether $H \supseteq \mathit{Exec}$ and, if so, whether $H \cap D = \emptyset$.
If the Teacher answers

1. $H \supseteq \mathit{Exec}$ and $H \cap D = \emptyset$, the Learner has learned a SS, stop.
2. $H \supseteq \mathit{Exec}$ and $H \cap D \neq \emptyset$, then the Teacher returns $(\sigma, c) \in H \cap D$.
   The Learner checks whether $(\sigma, c) \in \mathit{Exec}$:
   2.1. if $(\sigma, c) \in \mathit{Exec}$, then the Learner has learned a DE, stop;
   2.2. if $(\sigma, c) \notin \mathit{Exec}$, then $(\sigma, c) \in H \oplus \mathit{Exec}$, and so the Learner has got a counterexample.
3. $H \not\supseteq \mathit{Exec}$, then the Teacher returns some element in $H \oplus \mathit{Exec}$ as counterexample.

Containment queries

Containment queries: the Learner produces a regular hypothesis H, and asks the Teacher whether $H \supseteq Exec$ and, if so, whether $H \cap D = \emptyset$.
If the Teacher answers

1. $H \supseteq Exec$ and $H \cap D = \emptyset$, the Learner has learned a SS, stop.
2. $H \supseteq Exec$ and $H \cap D \neq \emptyset$, then the Teacher returns $(\sigma, c) \in H \cap D$.
   The Learner checks whether $(\sigma, c) \in Exec$:

   2.1. if $(\sigma, c) \in Exec$, then the Learner has learned a DE, stop;

   2.2. if $(\sigma, c) \notin Exec$, then $(\sigma, c) \in H \oplus Exec$, and so the Learner has got a counterexample.

3. $H \not\supseteq Exec$, then the Teacher returns some element in $H \oplus Exec$ as counterexample.

. . . but checking $H \subseteq Exec$ is also hopeless!

Pre-fixpoint queries (1/3)

We only check a sufficient condition for $H \supseteq Exec$.

Pre-fixpoint queries (1/3)

We only check a sufficient condition for $H \supseteq \mathit{Exec}$.

The clever idea:

If $c \xrightarrow{t} c'$, then say $(\sigma, c) \to (\sigma t, c')$. Given a set M of MTSs, let

$$\mathit{post}(M) = \{m \mid \exists m' \in M \wedge m' \to m\}$$

Exec is the least fixed point of the equation $X = \mathcal{F}(X)$ where

$$\mathcal{F}(X) =_{def} \{(\epsilon, c) \mid c \in I\} \cup \mathit{post}(X)$$

By standard fixed point theory: if $\mathcal{F}(H) \subseteq H$, then $H \supseteq \mathit{Exec}$.

Pre-fixpoint queries (1/3)

We only check a sufficient condition for $H \supseteq \mathit{Exec}$.

The clever idea:

If $c \xrightarrow{t} c'$, then say $(\sigma, c) \rightarrow (\sigma t, c')$. Given a set M of MTSs, let

$$\mathit{post}(M) = \{m \mid \exists m' \in M \wedge m' \rightarrow m\}$$

Exec is the least fixed point of the equation $X = \mathcal{F}(X)$ where

$$\mathcal{F}(X) =_{def} \{(\epsilon, c) \mid c \in I\} \cup \mathit{post}(X)$$

By standard fixed point theory: if $\mathcal{F}(H) \subseteq H$, then $H \supseteq \mathit{Exec}$.

We replace the query $H \supseteq \mathit{Exec}$ by the query $\mathcal{F}(H) \subseteq H$.

Pre-fixpoint queries (2/3)

Pre-fixpoint queries: the Learner produces a regular hypothesis H, asks the Teacher whether $\mathcal{F}(H) \subseteq H$ and, if so, whether $H \cap D = \emptyset$. If the Teacher answers

Pre-fixpoint queries (2/3)

Pre-fixpoint queries: the Learner produces a regular hypothesis H, asks the Teacher whether $\mathcal{F}(H) \subseteq H$ and, if so, whether $H \cap D = \emptyset$. If the Teacher answers

1. $\mathcal{F}(H) \subseteq H$, then $H \supseteq \mathit{Exec}$, and we can proceed as before.

Pre-fixpoint queries (2/3)

Pre-fixpoint queries: the Learner produces a regular hypothesis H, asks the Teacher whether $\mathcal{F}(H) \subseteq H$ and, if so, whether $H \cap D = \emptyset$. If the Teacher answers

1. $\mathcal{F}(H) \subseteq H$, then $H \supseteq Exec$, and we can proceed as before.
2. $\mathcal{F}(H) \setminus H \neq \emptyset$, then the Teacher chooses $m \in \mathcal{F}(H) \setminus H$.
   So we have $m \in \{(\epsilon, c) \mid c \in I\} \cup post(H)$ and $m \notin H$.

Pre-fixpoint queries (2/3)

Pre-fixpoint queries: the Learner produces a regular hypothesis H, asks the Teacher whether $\mathcal{F}(H) \subseteq H$ and, if so, whether $H \cap D = \emptyset$. If the Teacher answers

1. $\mathcal{F}(H) \subseteq H$, then $H \supseteq \mathit{Exec}$, and we can proceed as before.
2. $\mathcal{F}(H) \setminus H \neq \emptyset$, then the Teacher chooses $m \in \mathcal{F}(H) \setminus H$.
   So we have $m \in \{(\epsilon, c) \mid c \in I\} \cup \mathit{post}(H)$ and $m \notin H$.

   2.1 If $m \in \{(\epsilon, c) \mid c \in I\}$, then $m \in \mathit{Exec} \setminus H$.
   The Teacher returns m as counterexample.

Pre-fixpoint queries (2/3)

Pre-fixpoint queries: the Learner produces a regular hypothesis H, asks the Teacher whether $\mathcal{F}(H) \subseteq H$ and, if so, whether $H \cap D = \emptyset$. If the Teacher answers

1. $\mathcal{F}(H) \subseteq H$, then $H \supseteq Exec$, and we can proceed as before.
2. $\mathcal{F}(H) \setminus H \neq \emptyset$, then the Teacher chooses $m \in \mathcal{F}(H) \setminus H$.
   So we have $m \in \{(\epsilon, c) \mid c \in I\} \cup post(H)$ and $m \notin H$.

   2.1 If $m \in \{(\epsilon, c) \mid c \in I\}$, then $m \in Exec \setminus H$.
   The Teacher returns m as counterexample.

   2.2 If $m \in post(H)$, the Teacher computes $m' \in H$ with $m' \to m$.

Pre-fixpoint queries (2/3)

Pre-fixpoint queries: the Learner produces a regular hypothesis H, asks the Teacher whether $\mathcal{F}(H) \subseteq H$ and, if so, whether $H \cap D = \emptyset$. If the Teacher answers

1. $\mathcal{F}(H) \subseteq H$, then $H \supseteq \mathit{Exec}$, and we can proceed as before.
2. $\mathcal{F}(H) \setminus H \neq \emptyset$, then the Teacher chooses $m \in \mathcal{F}(H) \setminus H$.
   So we have $m \in \{(\epsilon, c) \mid c \in I\} \cup \mathit{post}(H)$ and $m \notin H$.

   2.1 If $m \in \{(\epsilon, c) \mid c \in I\}$, then $m \in \mathit{Exec} \setminus H$.
   The Teacher returns m as counterexample.

   2.2 If $m \in \mathit{post}(H)$, the Teacher computes $m' \in H$ with $m' \to m$.

   2.2.1 If $m' \notin \mathit{Exec}$, then $m' \in H \setminus \mathit{Exec}$.
   The Teacher returns m' as counterexample.

Pre-fixpoint queries (2/3)

Pre-fixpoint queries: the Learner produces a regular hypothesis H, asks the Teacher whether $\mathcal{F}(H) \subseteq H$ and, if so, whether $H \cap D = \emptyset$. If the Teacher answers

1. $\mathcal{F}(H) \subseteq H$, then $H \supseteq \mathit{Exec}$, and we can proceed as before.
2. $\mathcal{F}(H) \setminus H \neq \emptyset$, then the Teacher chooses $m \in \mathcal{F}(H) \setminus H$.
   So we have $m \in \{(\epsilon, c) \mid c \in I\} \cup \mathit{post}(H)$ and $m \notin H$.

   2.1 If $m \in \{(\epsilon, c) \mid c \in I\}$, then $m \in \mathit{Exec} \setminus H$.
   The Teacher returns m as counterexample.

   2.2 If $m \in \mathit{post}(H)$, the Teacher computes $m' \in H$ with $m' \rightarrow m$.

   2.2.1 If $m' \notin \mathit{Exec}$, then $m' \in H \setminus \mathit{Exec}$.
   The Teacher returns m' as counterexample.

   2.2.2 If $m' \in \mathit{Exec}$, then $m \in \mathit{Exec}$ ($m' \rightarrow m$ holds) and so $m \in \mathit{Exec} \setminus H$.
   The Teacher returns m as counterexample.

Fixed point queries (2/3)

Remaining problems:

- decide $\mathcal{F}(H) \subseteq H$, and if not
- compute $m \in \mathcal{F}(H) \setminus H$.

Fixed point queries (2/3)

Remaining problems:

- decide $\mathcal{F}(H) \subseteq H$, and if not
- compute $m \in \mathcal{F}(H) \setminus H$.

Theorem (exercise): If M is a regular set of MTSs of a (lossy) channel system, then so is $post(M)$. Moreover, $post(M)$ can be effectively computed.

Fixed point queries (2/3)

Remaining problems:

- decide $\mathcal{F}(H) \subseteq H$, and if not
- compute $m \in \mathcal{F}(H) \setminus H$.

Theorem (exercise): If M is a regular set of MTSs of a (lossy) channel system, then so is $post(M)$. Moreover, $post(M)$ can be effectively computed.

Corollary: If I is a regular set of configurations and H is a regular hypothesis of a (lossy) channel system, then $\mathcal{F}(H)$ is also regular and can be effectively computed.

Fixed point queries (2/3)

Remaining problems:

- decide $\mathcal{F}(H) \subseteq H$, and if not
- compute $m \in \mathcal{F}(H) \setminus H$.

Theorem (exercise): If M is a regular set of MTSs of a (lossy) channel system, then so is $post(M)$. Moreover, $post(M)$ can be effectively computed.

Corollary: If I is a regular set of configurations and H is a regular hypothesis of a (lossy) channel system, then $\mathcal{F}(H)$ is also regular and can be effectively computed.

Algorithms for the remaining problems follow easily from the Corollary.

Some observations

The learning algorithm is complete in the following sense: if *Exec* is regular, then the algorithm terminates.

Some observations

The learning algorithm is complete in the following sense: if *Exec* is regular, then the algorithm terminates.

We learn either a dangerous execution or an invariant proving that there are no dangerous executions.

Some observations

The learning algorithm is complete in the following sense: if *Exec* is regular, then the algorithm terminates.

We learn either a dangerous execution or an invariant proving that there are no dangerous executions.

In practice, the assumption '*Exec* is regular' is stronger than the assumption '$post^*(I)$ is regular'. For instance, $post^*(I)$ is always regular for a pushdown system (assuming I regular), while *Exec* is context-free.

Some observations

The learning algorithm is complete in the following sense: if *Exec* is regular, then the algorithm terminates.

We learn either a dangerous execution or an invariant proving that there are no dangerous executions.

In practice, the assumption '*Exec* is regular' is stronger than the assumption '$post^*(I)$ is regular'. For instance, $post^*(I)$ is always regular for a pushdown system (assuming I regular), while *Exec* is context-free.

The assumption '*Exec* is regular' may depend on the encoding use to represent a pair (σ, c) as a word.