Kleene Algebra with Tests: A Tutorial

Part II

Dexter Kozen
Cornell University

RAMiCS, September 17–18, 2012

Today

- automata on guarded strings (AGS)
- schematic KAT (SKAT)
- strictly deterministic automata and flowchart programs
- Kleene coalgebra (KC) and Kleene coalgebra with tests (KCT)
- automata theory and program schematology
- the Brzozowski derivative
- minimization as finality
- automatic extraction of equivalence proofs and relation to proof-carrying code

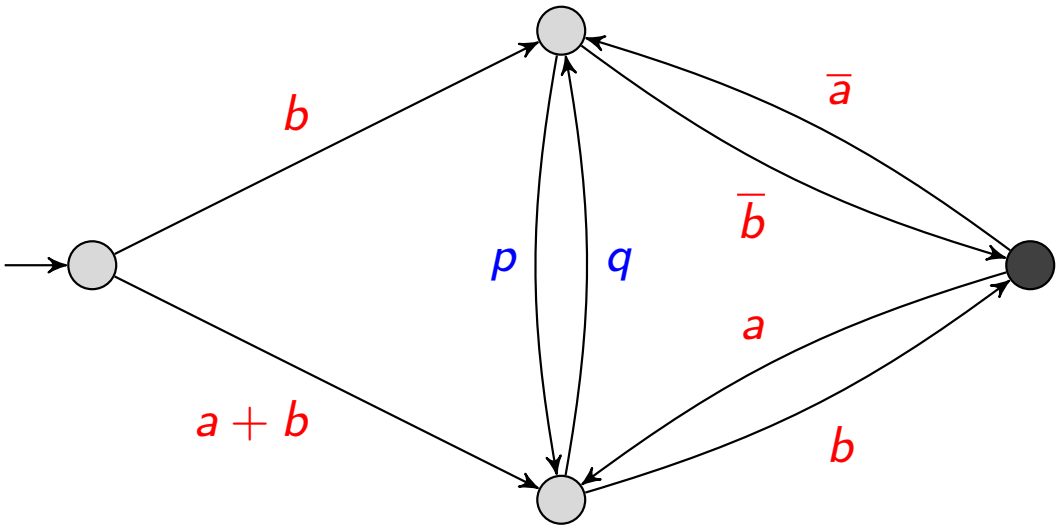
Automata on Guarded Strings (AGS)

- A generalization of classical automata theory to include Booleans
- An ε-transition is really a 1-transition (i.e., an ordinary automaton with ε-transitions is an AGS over the two-element Boolean algebra)
- Classical constructions of ordinary finite-state automata generalize readily
  - determinization
  - state minimization
  - Kleene's theorem

Automata on Guarded Strings (AGS)

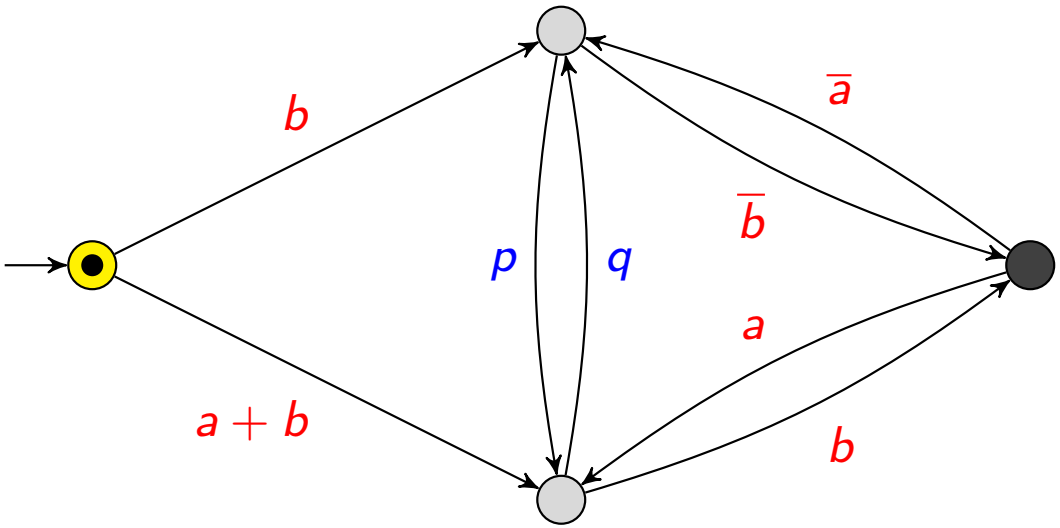
- Labeled transition system with states Q,
  start states $S \subseteq Q$, accept states $F \subseteq Q$
- atomic action symbols Σ, atomic test symbols T,
  $B =$ the free Boolean algebra generated by T,
  $\mathrm{At} = \{\text{atoms of } B\}$
- action transitions $s \xrightarrow{p} t$, $p \in \Sigma$
- test transitions $s \xrightarrow{b} t$, $b \in B$
- inputs are guarded strings $\alpha_0 p_0 \alpha_1 p_1 \cdots \alpha_{n-1} p_{n-1} \alpha_n$
- traces $s_0 q_0 s_1 q_1 \cdots s_{n-1} q_{n-1} s_n$ where $s_i \xrightarrow{q_i} s_{i+1}$, $q_i \in \Sigma \cup B$
- x accepted if $\exists$ trace $s_0 q_0 s_1 q_1 \cdots s_{n-1} q_{n-1} s_n$ such that $s_0 \in S$,
  $s_n \in F$, $x \in G(q_0, \ldots, q_{n-1})$

Automata on Guarded Strings (AGS)



$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

Automata on Guarded Strings (AGS)

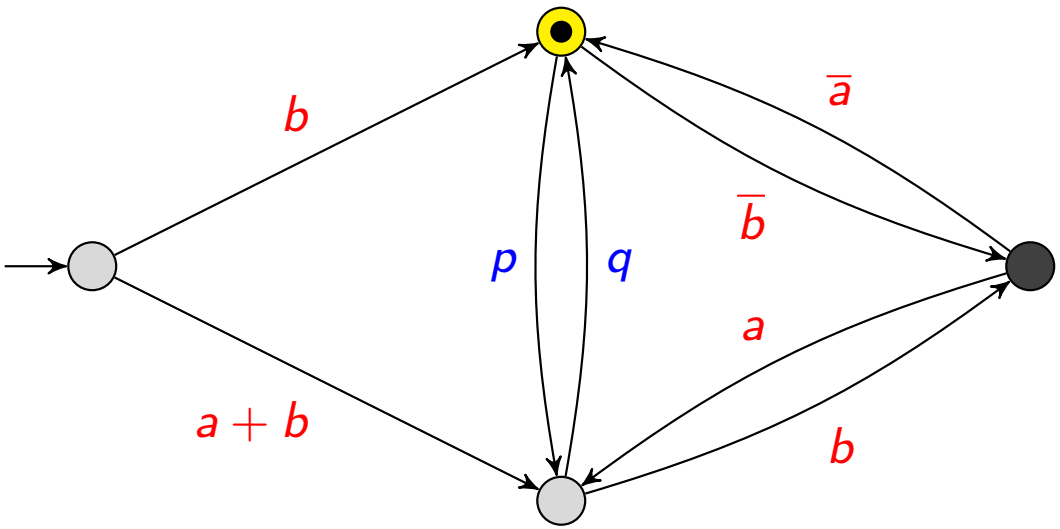


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace:

Automata on Guarded Strings (AGS)

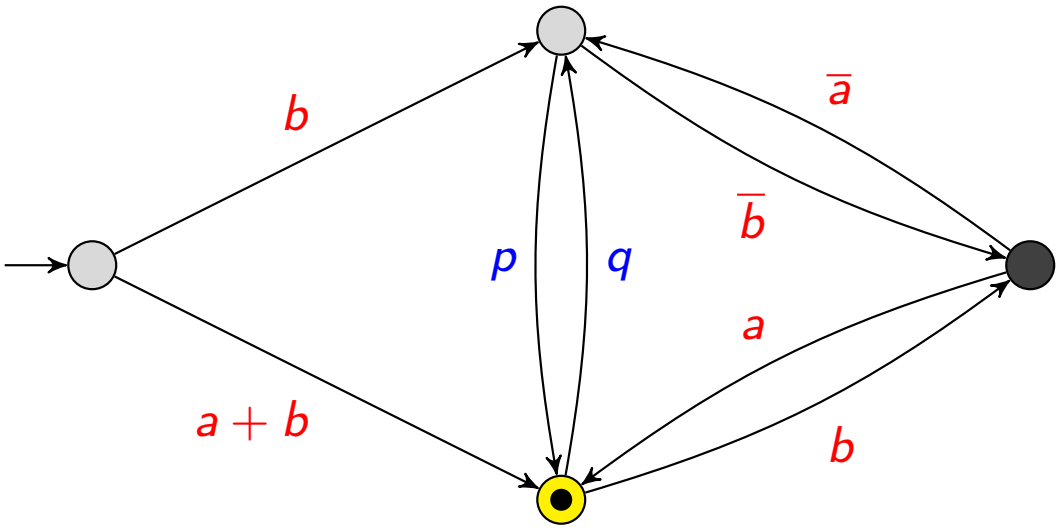


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace: b

Automata on Guarded Strings (AGS)

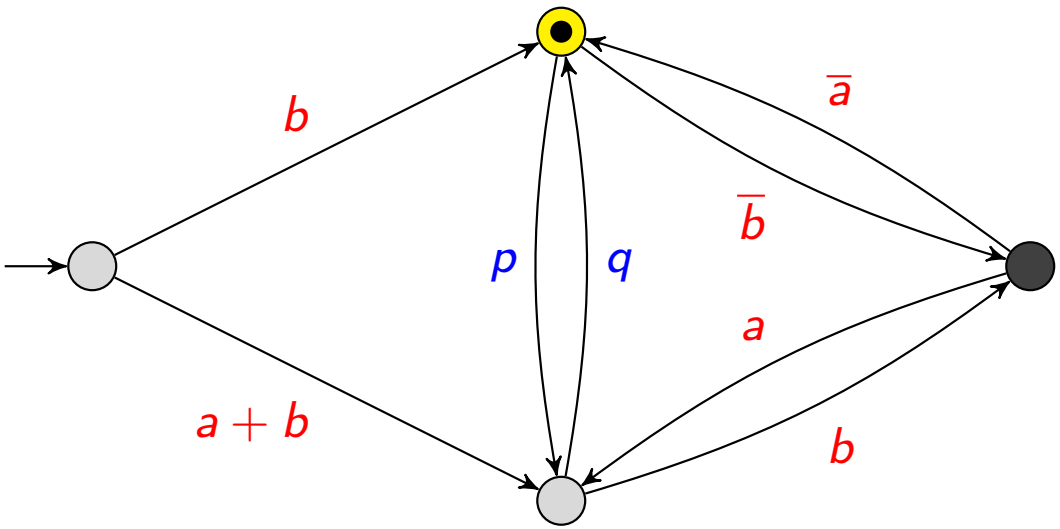


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace: bp

Automata on Guarded Strings (AGS)

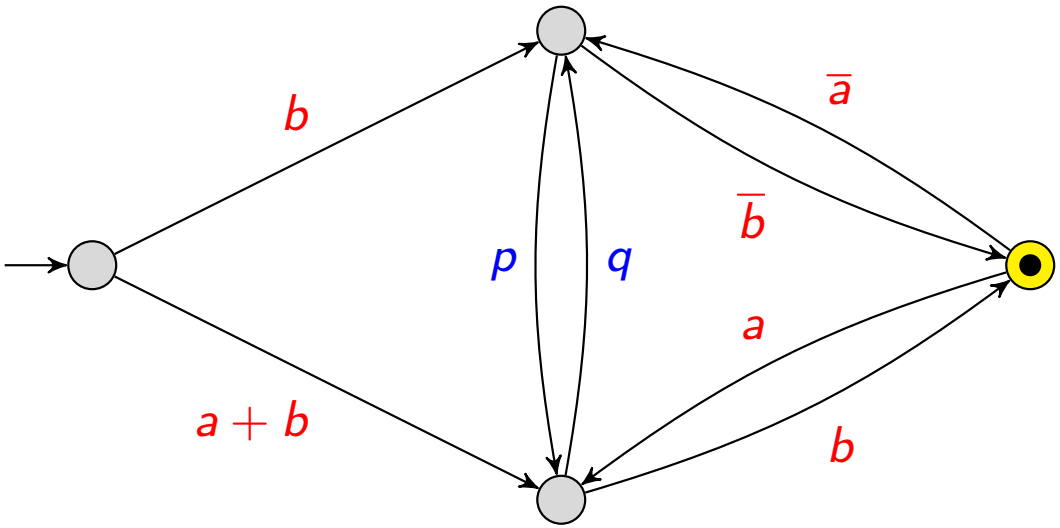


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace: bpq

Automata on Guarded Strings (AGS)

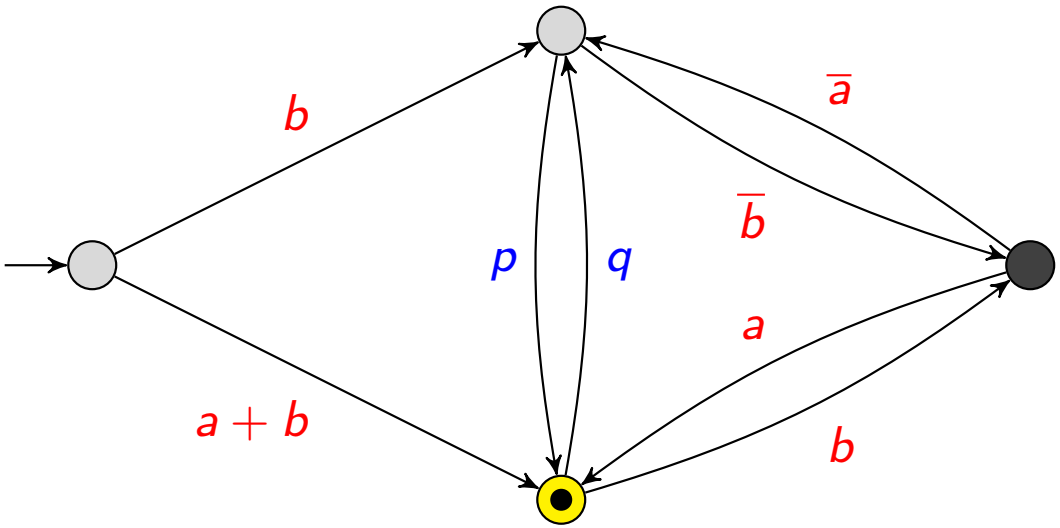


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace: $bpq\overline{b}$

Automata on Guarded Strings (AGS)

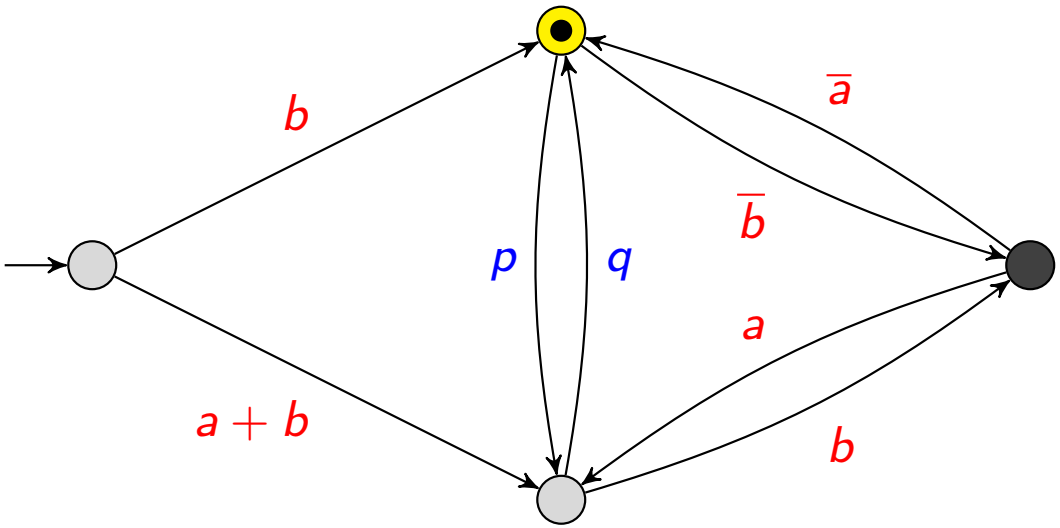


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace: $bpq\overline{b}a$

Automata on Guarded Strings (AGS)

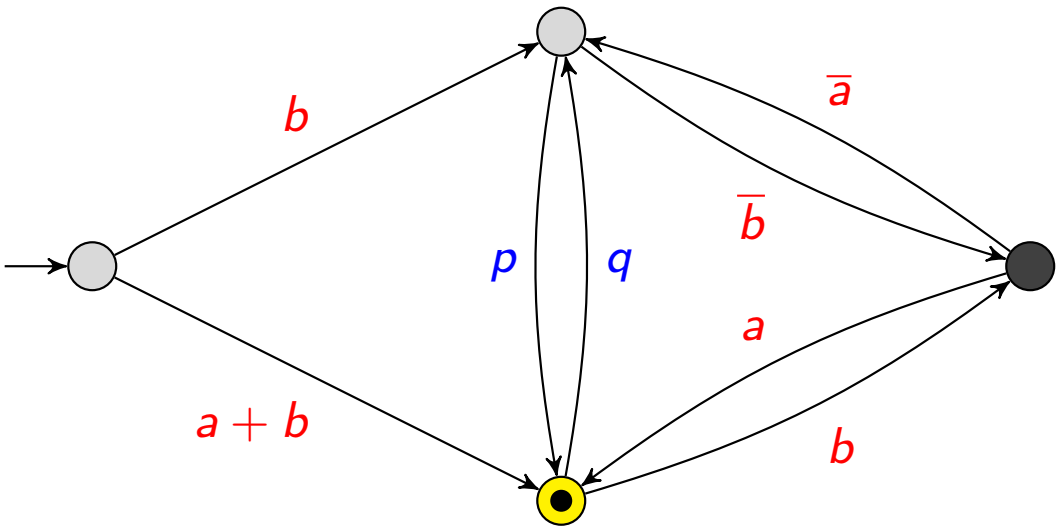


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace: $bpq\overline{b}aq$

Automata on Guarded Strings (AGS)

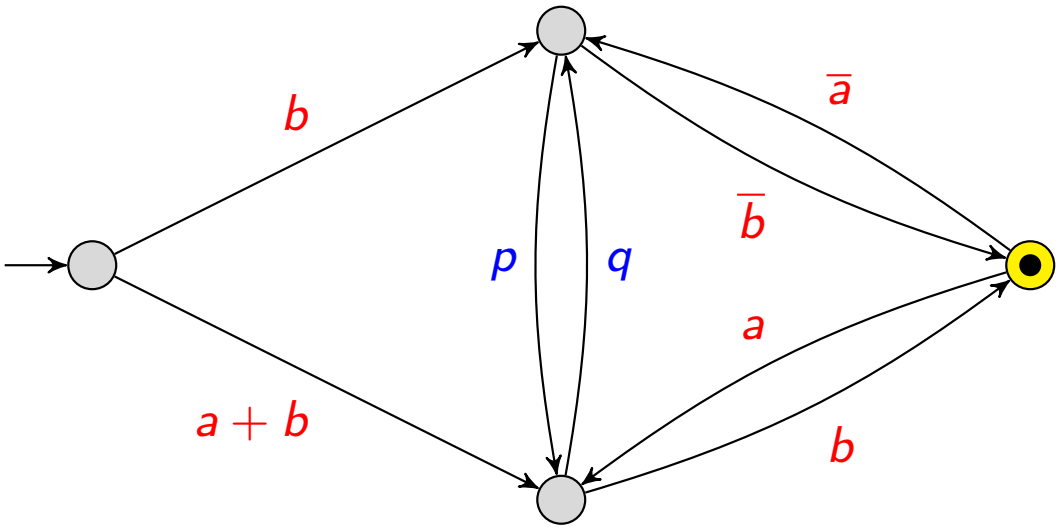


$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲

trace: $bpq\overline{b}aqp$

Automata on Guarded Strings (AGS)



$ab\ p\ \overline{a}b\ q\ a\overline{b}\ q\ \overline{a}\overline{b}\ p\ ab$

▲ Accept!

trace: $bpq\overline{b}aqpb$

Deterministic Automata

1. exactly one start state
2. each state is an action state or a test state, not both
3. action states: exactly one transition for each element of Σ
4. test states: exiting tests are propositionally pairwise exclusive and exhaustive
5. every cycle contains at least one action state
6. all final states are action states

Some Facts

- Kleene's theorem (nondeterministic automaton constructed is linear in the size of the KAT expression)
- *PSPACE*-completeness
- can determinize with a subset construction
- can minimize via a variant of Myhill–Nerode
- minimal OBDDs are a special case of this construction

Schematic KAT (SKAT)

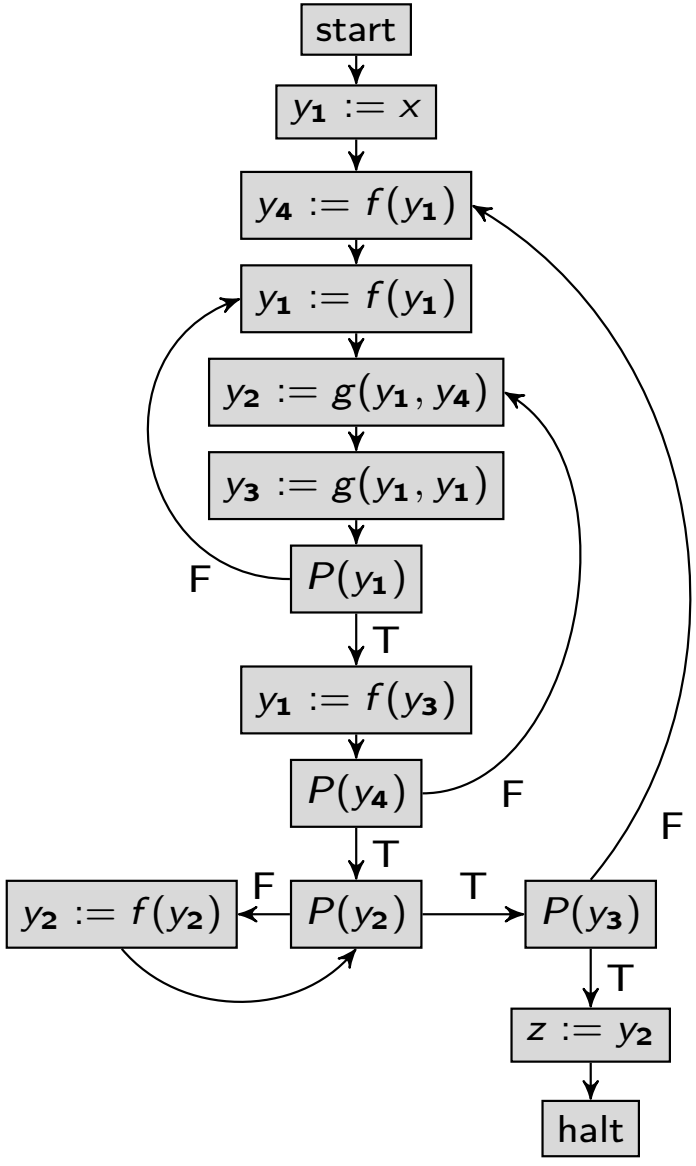
$$
\begin{array}{rcll}
x := s \,;\, y := t & = & y := t[x/s] \,;\, x := s & (y \notin \mathrm{FV}(s)) \\
x := s \,;\, y := t & = & x := s \,;\, y := t[x/s] & (x \notin \mathrm{FV}(s)) \\
x := s \,;\, x := t & = & x := t[x/s] & \\
\varphi[x/t] \,;\, x := t & = & x := t \,;\, \varphi &
\end{array}
$$

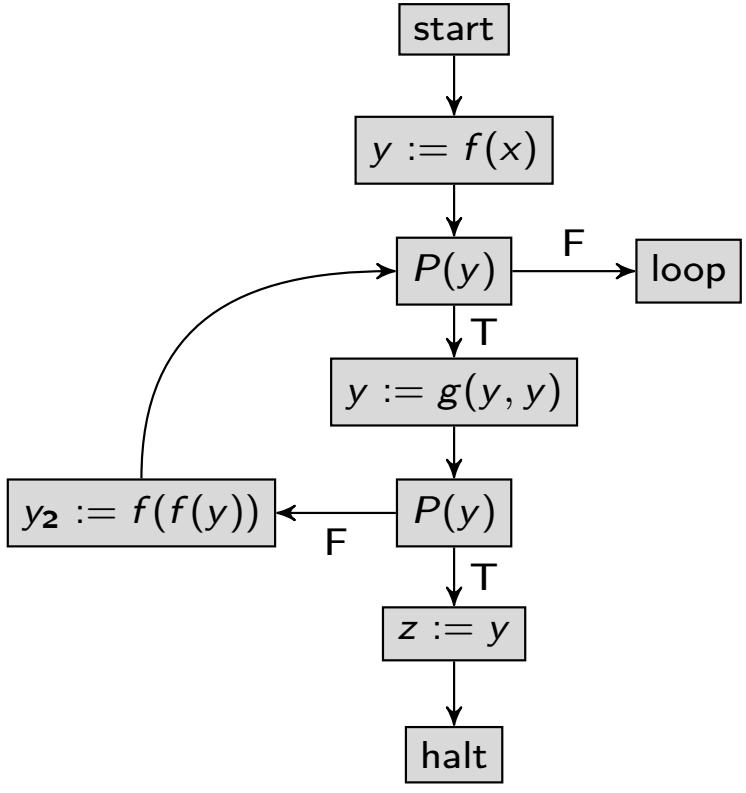
In particular,

$$
\begin{array}{rcll}
x := s \,;\, y := t & = & y := t \,;\, x := s & (x \notin \mathrm{FV}(t), y \notin \mathrm{FV}(s)) \\
\varphi \,;\, x := t & = & x := t \,;\, \varphi & (x \notin \mathrm{FV}(\varphi))
\end{array}
$$

Scheme Equivalence

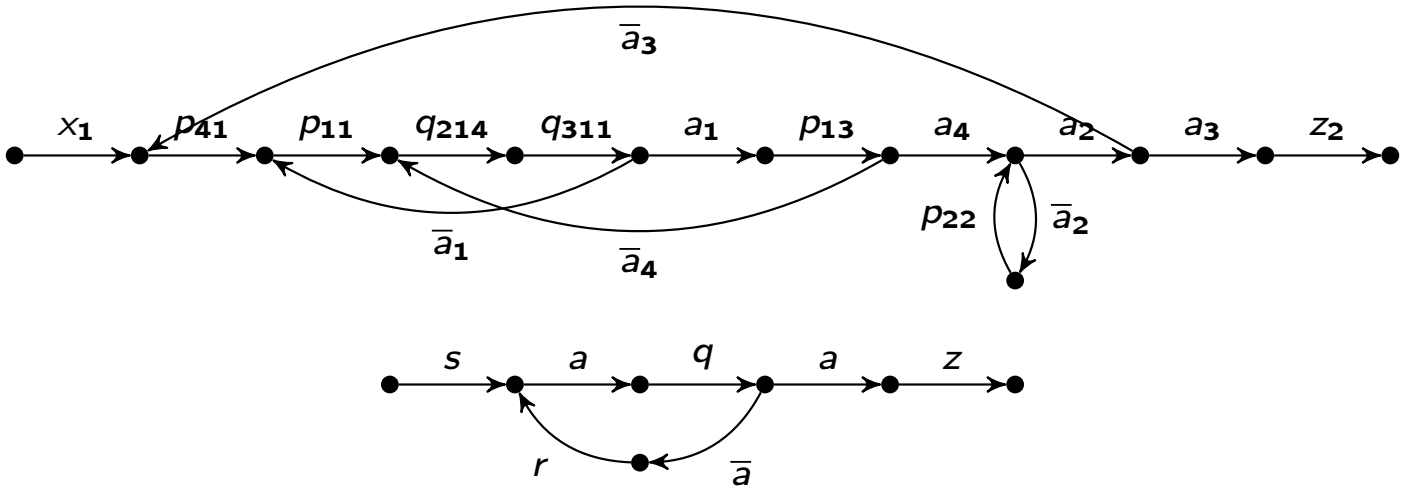
Example of Paterson from [Manna 74]





Scheme Equivalence

Example of Paterson from [Manna 74]



$$x_1 p_{41} p_{11} q_{214} q_{311} (\bar{a}_1 p_{11} q_{214} q_{311})^* a_1 p_{13}$$
$$((\bar{a}_4 + a_4 (\bar{a}_2 p_{22})^* a_2 \bar{a}_3 p_{41} p_{11}) q_{214} q_{311} (\bar{a}_1 p_{11} q_{214} q_{311})^* a_1 p_{13})^*$$
$$a_4 (\bar{a}_2 p_{22})^* a_2 a_3 z_2 z$$
$$= saq(\bar{a}raq)^* az$$

Strictly Deterministic Automata

Flowchart programs correspond to a limited class of AGS called strictly deterministic.

Intuitively:

- An input is an infinite sequence of atoms provided by an external agent
- The automaton responds to each atom in sequence deterministically according to its transition function—either
  - emits an action symbol and moves to a new state,
  - halts, or
  - fails

Strictly Deterministic Automata

Formally, a strictly deterministic automaton over Σ, T is a structure

$$M \quad = \quad (Q, \delta, \text{start})$$

where Q is a set of states, start $\in Q$ is a start state, and

$$\delta : (Q \times \text{At}) \ \rightarrow \ (\Sigma \times Q) + \{\text{halt}, \text{fail}\},$$

(Note: halt, fail are not states)

Strictly Deterministic Automata

Given a state s and an infinite sequence of atoms $\sigma \in \mathrm{At}^\omega$, there is at most one finite or infinite guarded string $\mathrm{gs}(s, \sigma)$ obtained by running the automaton starting in state s.

Formally, define a partial map

$$\mathrm{gs} : (Q \times \mathrm{At}^\omega) \;\to\; (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At} \;+\; (\mathrm{At} \cdot \Sigma)^\omega$$

coinductively:

$$\mathrm{gs}(s, \alpha\,\sigma) \;\stackrel{\mathrm{def}}{=}\; \begin{cases} \alpha \cdot p \cdot \mathrm{gs}(t, \sigma), & \text{if } \delta(s, \alpha) = (p, t) \\ \alpha, & \text{if } \delta(s, \alpha) = \mathsf{halt} \\ \text{undefined}, & \text{if } \delta(s, \alpha) = \mathsf{fail} \end{cases}$$

The set of (finite) guarded strings represented by M is

$$\mathrm{gs}(M) \;\stackrel{\mathrm{def}}{=}\; \{\mathrm{gs}(\mathrm{start}, \sigma) \mid \sigma \in \mathrm{At}^\omega\} \cap (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}.$$

Bisimulation Between Strictly Deterministic Automata

A bisimulation between M and N is a binary relation $\equiv$ between Q_M and Q_N such that

- $\text{start}_M \equiv \text{start}_N$, and
- if $s \equiv t$ then
  - $\delta_M(s, \alpha) = \text{halt} \;\Leftrightarrow\; \delta_N(t, \alpha) = \text{halt}$;
  - $\delta_M(s, \alpha) = \text{fail} \;\Leftrightarrow\; \delta_N(t, \alpha) = \text{fail}$;
  - $\delta_M(s, \alpha) = (p, s') \;\wedge\; \delta_N(t, \alpha) = (q, t') \;\Rightarrow\; p = q \;\wedge\; s' \equiv t'$.

Bisimulation Between Strictly Deterministic Automata

Theorem

If M and N are bisimilar, then $\mathrm{gs}(M) = \mathrm{gs}(N)$.

Moreover, the converse is true if

- *M never fails,*
- *every reachable state in M can reach halt.*

Proof.

($\Leftarrow$) Define $s \equiv t \overset{\text{def}}{\iff} \forall \sigma \in \mathrm{At}^\omega \ \mathrm{gs}_M(s, \sigma) = \mathrm{gs}_N(t, \sigma)$.
Property 2 is easy to check. For property 1, want $\mathrm{gs}_M(\mathrm{start}_M, \sigma) = \mathrm{gs}_N(\mathrm{start}_N, \sigma)$ for all σ. Since $\mathrm{gs}(M) = \mathrm{gs}(N)$, if $\mathrm{gs}_M(\mathrm{start}_M, \sigma)$ is finite, then so is $\mathrm{gs}_N(\mathrm{start}_N, \sigma)$ and they are equal. Thus

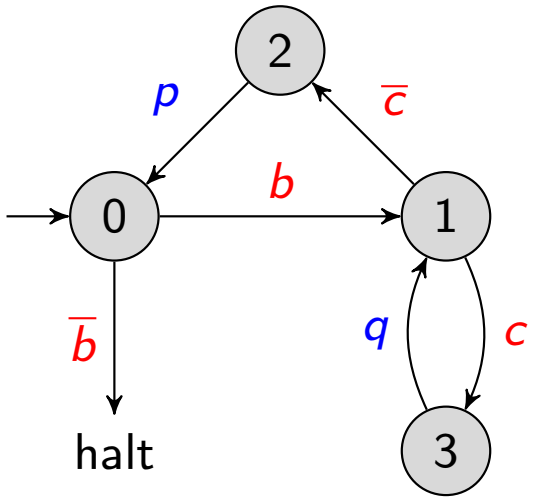
$$\begin{aligned} \mathrm{gs}_M(\mathrm{start}_M, -) : \mathrm{At}^\omega &\to (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At} + (\mathrm{At} \cdot \Sigma)^\omega \\ \mathrm{gs}_N(\mathrm{start}_N, -) : \mathrm{At}^\omega &\to (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At} + (\mathrm{At} \cdot \Sigma)^\omega \end{aligned}$$

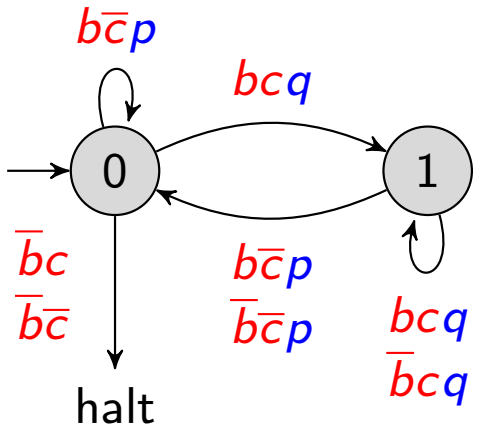
agree on the set $\{\sigma \mid \mathrm{gs}_M(\mathrm{start}_M, \sigma) \text{ is finite}\}$. This set is dense in At^ω. Moreover, the two functions are continuous, and continuous functions that agree on a dense set must agree everywhere. □

Strictly Deterministic Automata

Every while program is equivalent to a strictly deterministic automaton:

```
while b do {
    while c do q;
    p;
}
```





The converse is false (Ashcroft & Manna 1972)

The Böhm–Jacopini Theorem

Theorem (Böhm–Jacopini 1966)

Every deterministic flowchart program is equivalent to a while program.

- Formulated and proved in a first-order setting
- Their construction introduced extra auxiliary variables
- Are the auxiliary variables necessary?

A Negative Result

Theorem (Ashcroft & Manna 1972)

There is a deterministic flowchart program that cannot be converted to a while program without auxiliary variables.

- Formulated and proved in a first-order setting
- Counterexample has 13 states

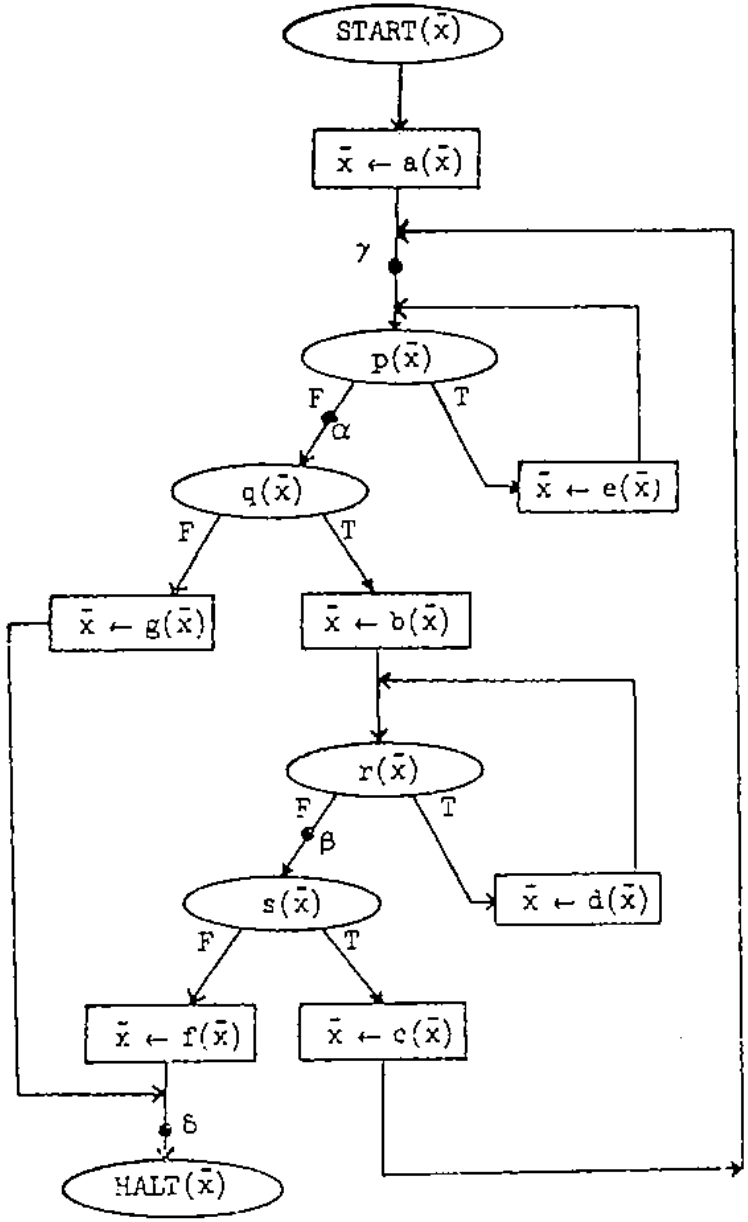
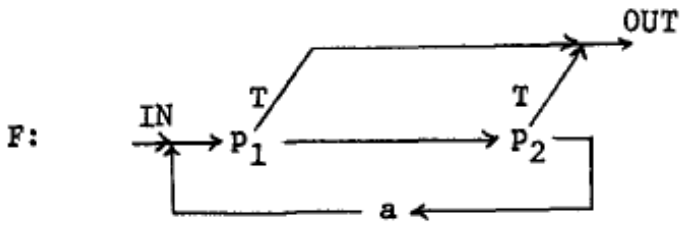


Fig. 1. The flowchart program P_1.

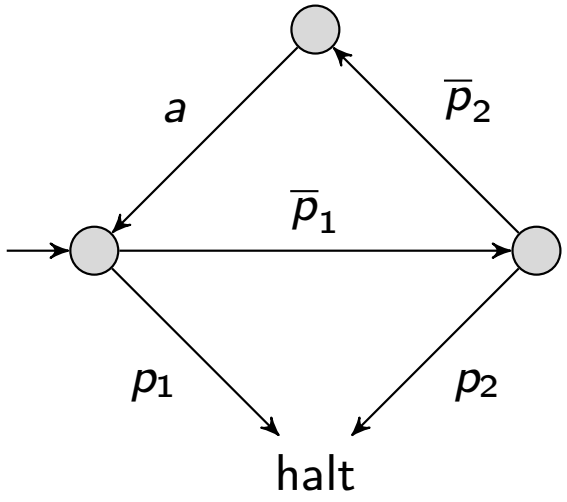
—from (Ashcroft & Manna 1972)

Kosaraju's Counterexample

Theorem 2: Flow chart F given below cannot be weakly equivalent to any D-flow chart which is built up from only p_1, p_2 and a:

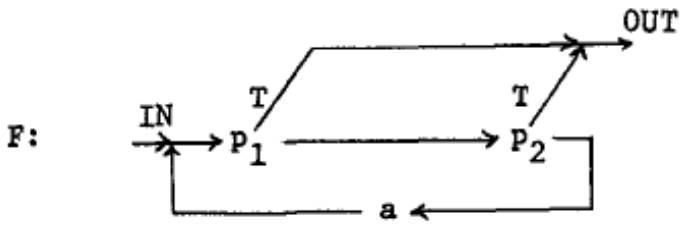


—from (Kosaraju 1973)

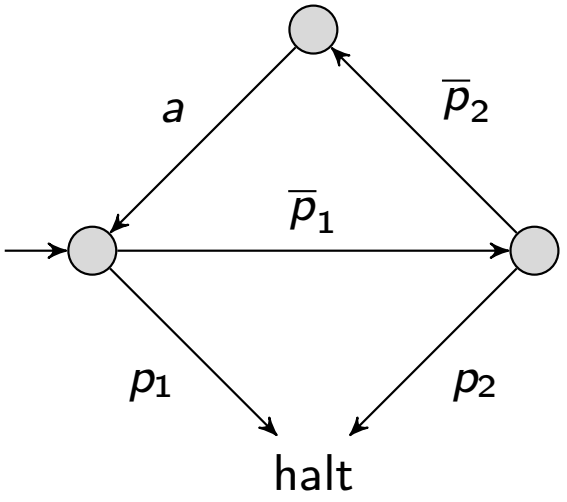


Kosaraju's Counterexample

Theorem 2: Flow chart F given below cannot be weakly equivalent to any D-flow chart which is built up from only p_1, p_2 and a:



—from (Kosaraju 1973)



. . . is not a counterexample: while $\overline{p}_1\overline{p}_2$ do a

The Loop Hierarchy

Theorem (Kosaraju 1973)

Every deterministic flowchart is equivalent to a loop program with multilevel breaks. Moreover, there is a strict hierarchy depending on the level of breaks allowed.

```
loop {
    ⋮
    loop {
        ⋮
        break 1;
        ⋮
    } ⟵ break 1 comes here
    ⋮
}
```

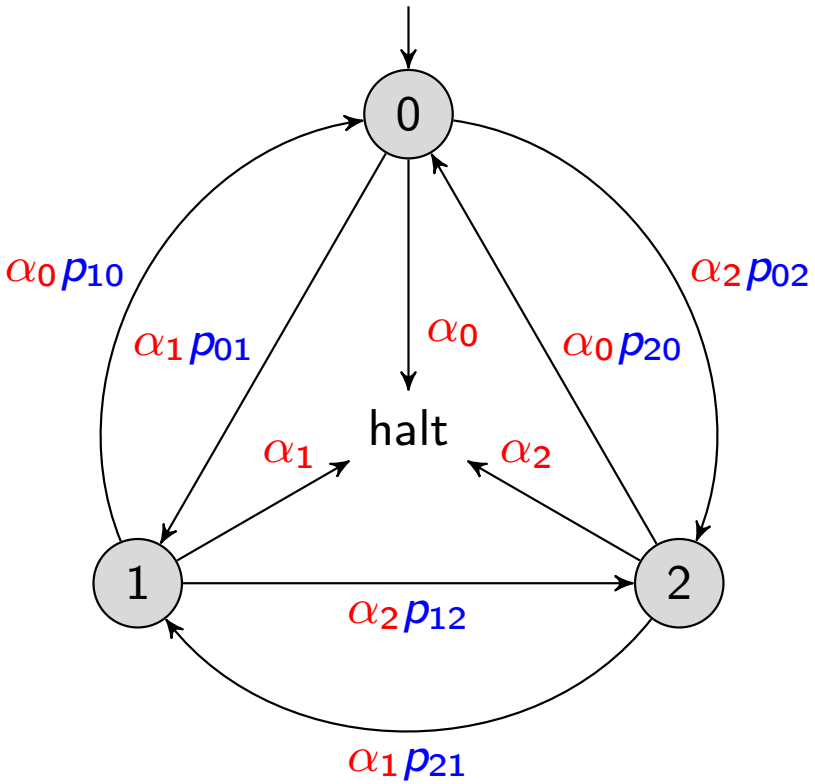
```
loop {
    ⋮
    loop {
        ⋮
        break 2;
        ⋮
    }
    ⋮
} ⟵ break 2 comes here
```

Theorem 5: For every flow chart having n basic predicate units, there exists a weakly equivalent RE_n-flow chart (using the same basic elements and RPT, END, and EXIT i, i = 1,...,n).

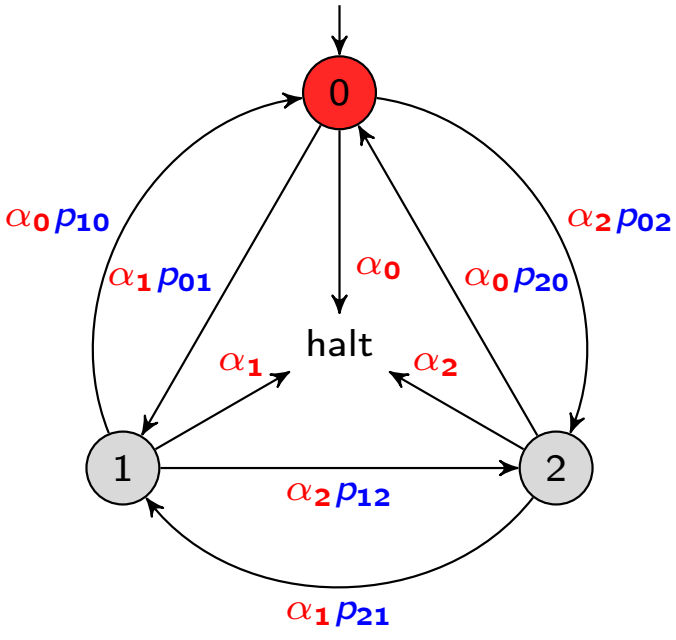
Proof: The proof is closely related to Böhm and Jacopini construction [2]. One should not have any problem in replacing the control variables used in Böhm and Jacopini's construction by RPT, END, EXIT constructs applicable to this class of flow charts. We leave the details to the reader. QED

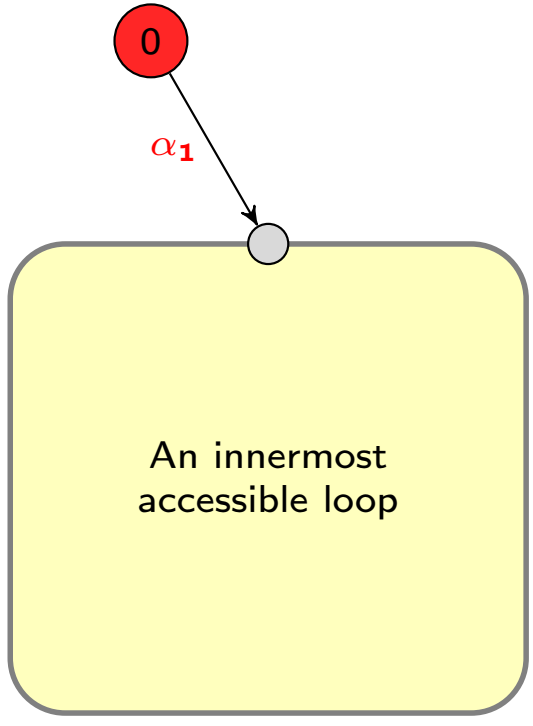
—from (Kosaraju 1973)

A 3-State SDA not Equivalent to Any While Program

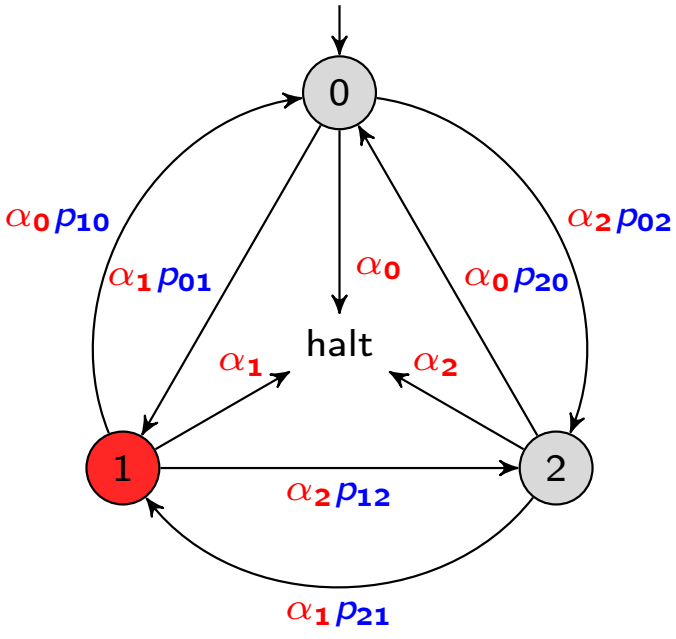


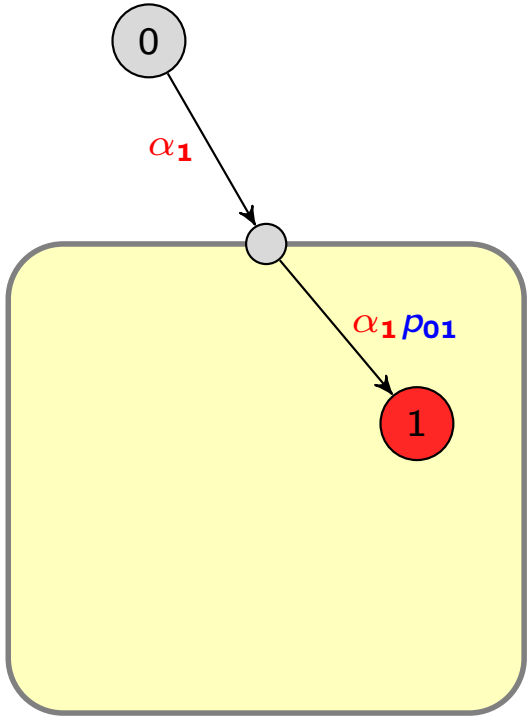
A 3-State SDA not Equivalent to Any While Program



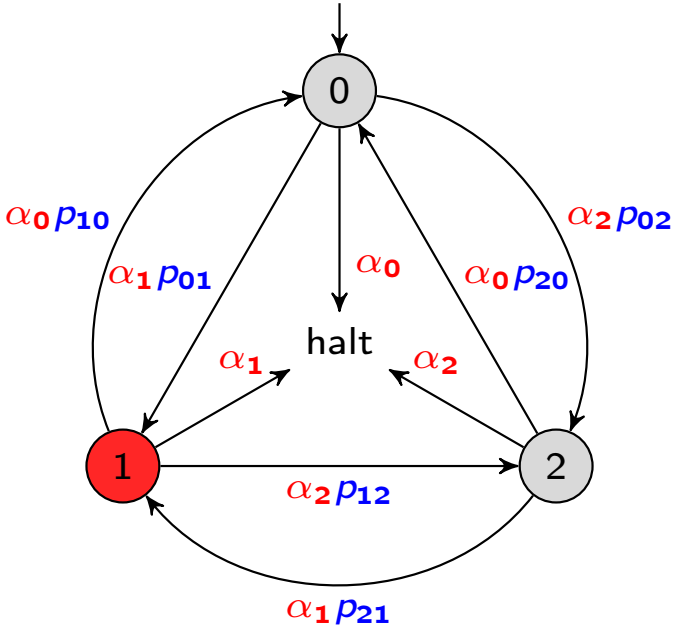


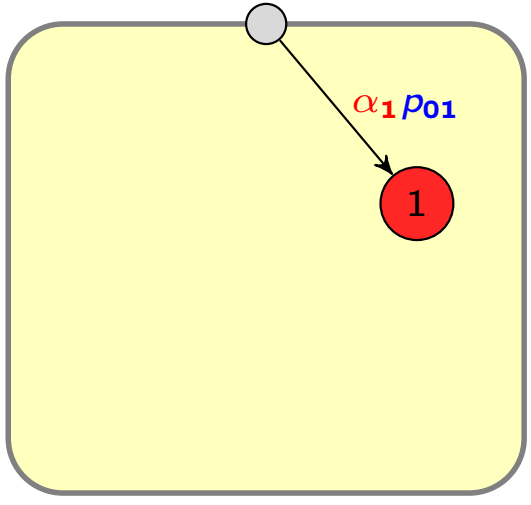
A 3-State SDA not Equivalent to Any While Program



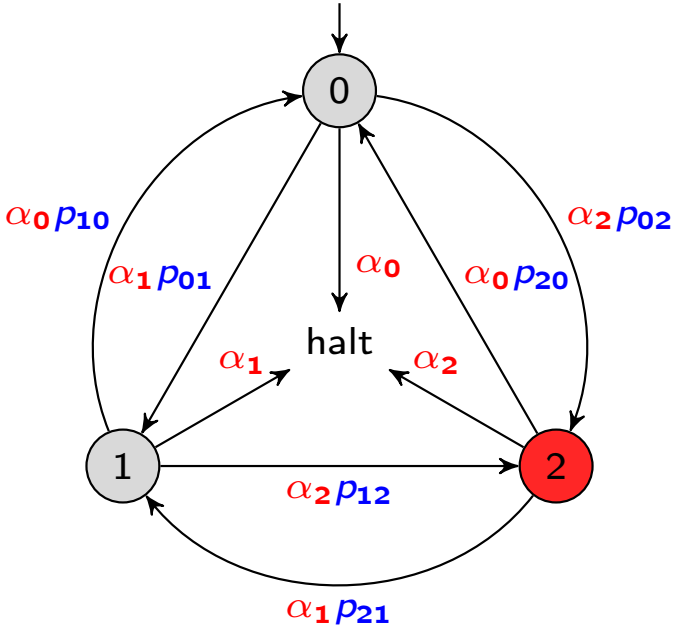


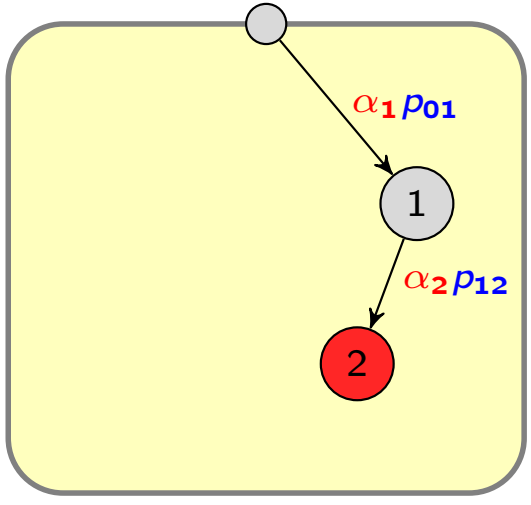
A 3-State SDA not Equivalent to Any While Program



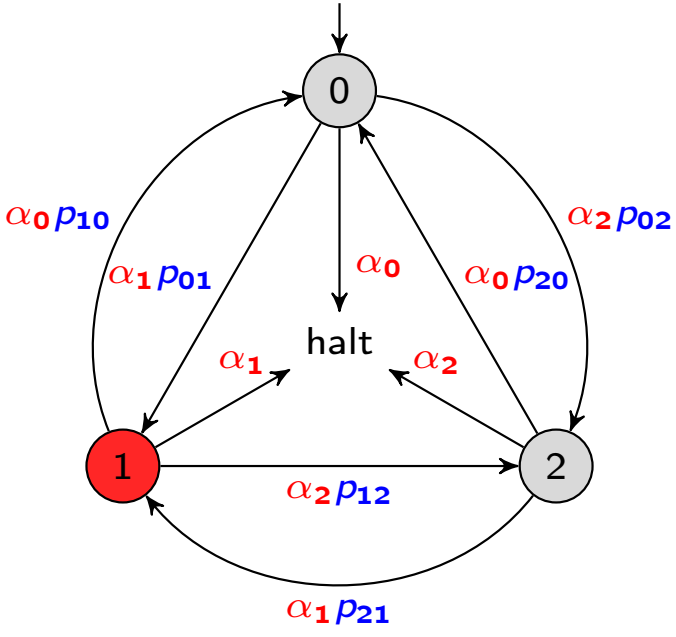


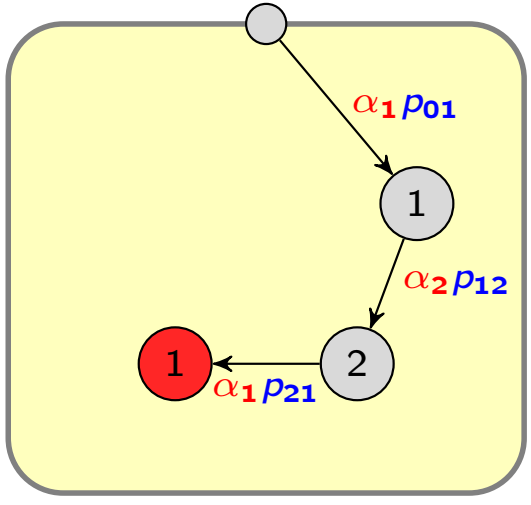
A 3-State SDA not Equivalent to Any While Program



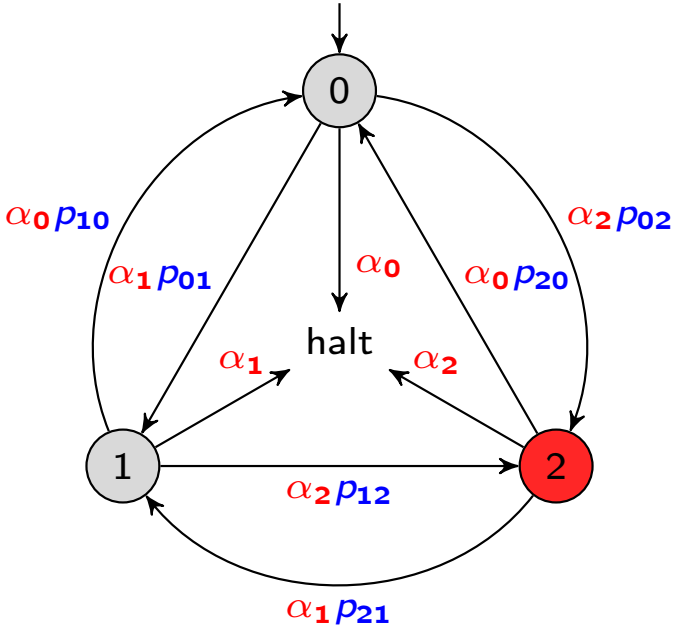


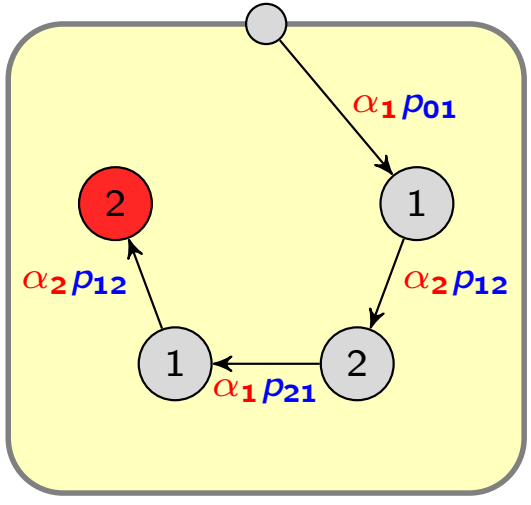
A 3-State SDA not Equivalent to Any While Program



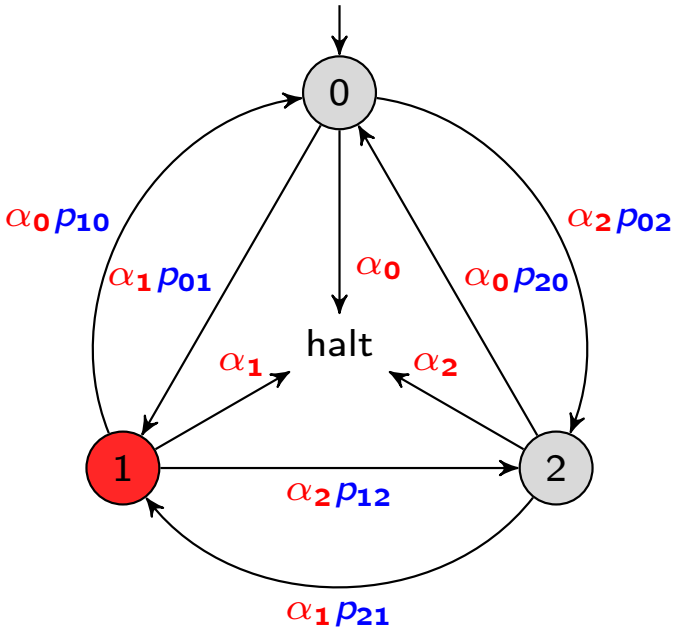


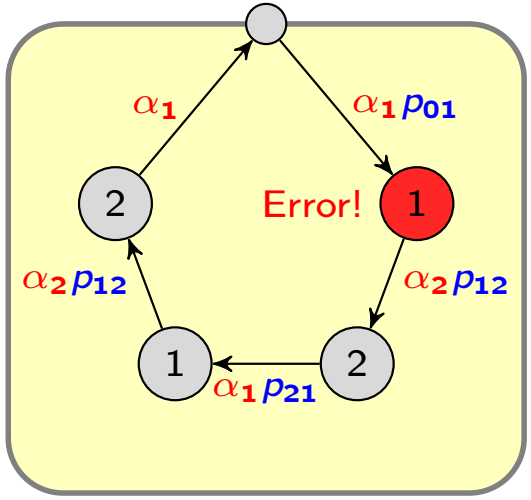
A 3-State SDA not Equivalent to Any While Program



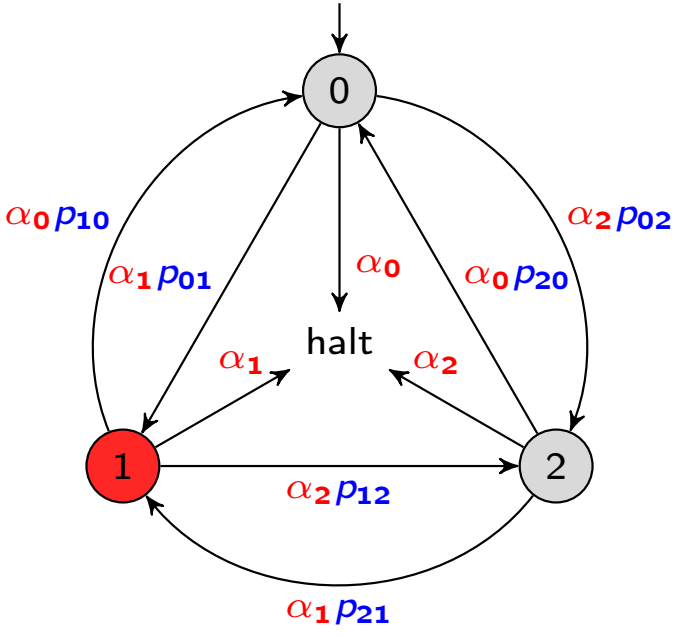


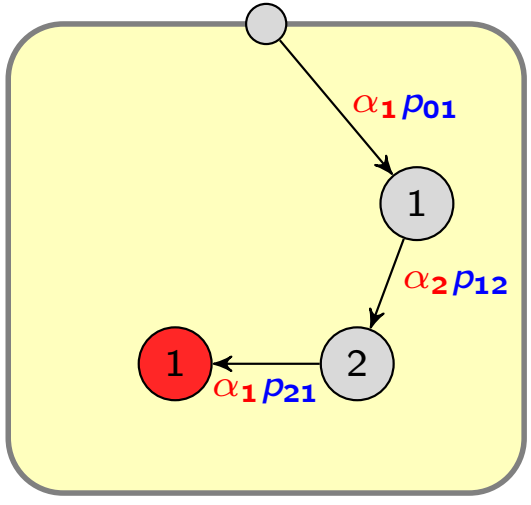
A 3-State SDA not Equivalent to Any While Program



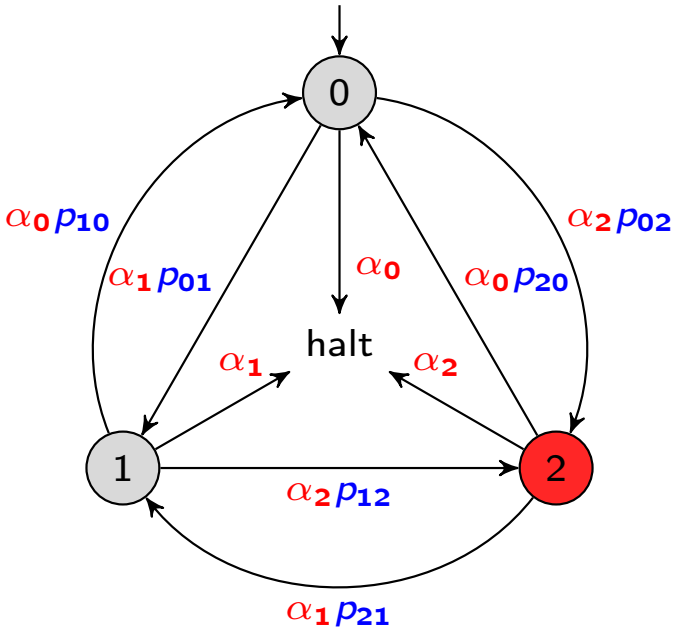


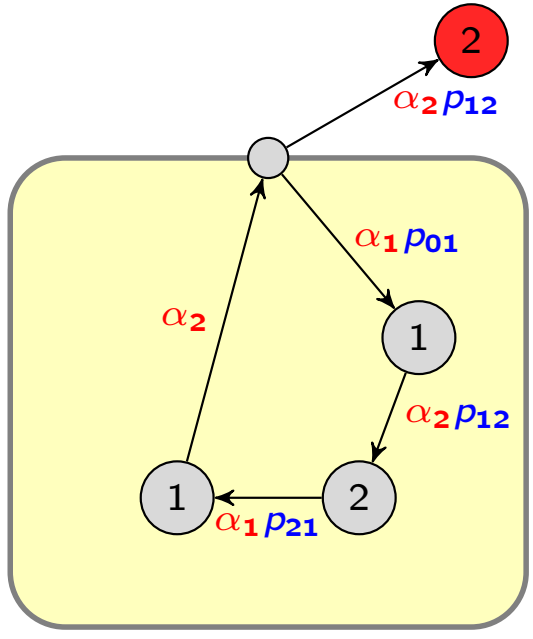
A 3-State SDA not Equivalent to Any While Program



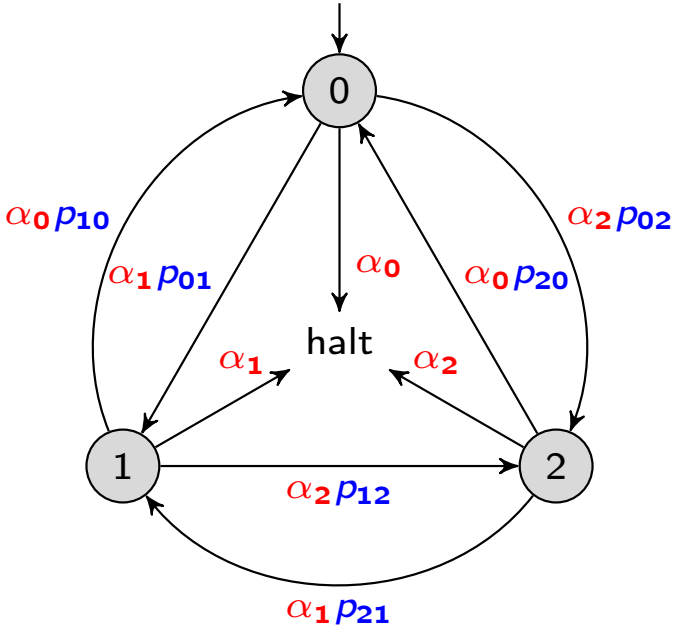


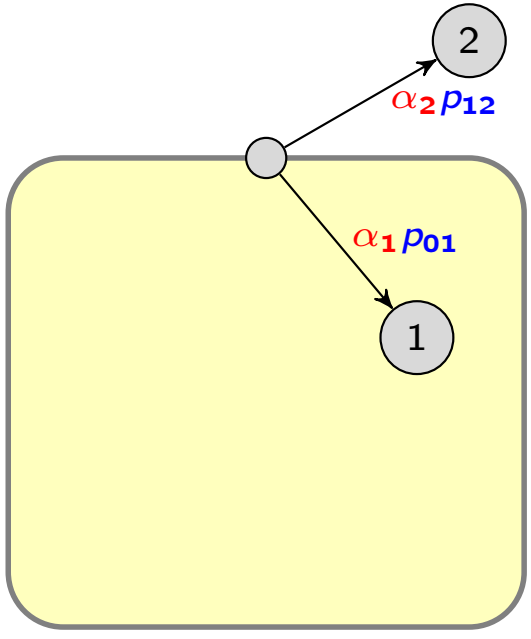
A 3-State SDA not Equivalent to Any While Program



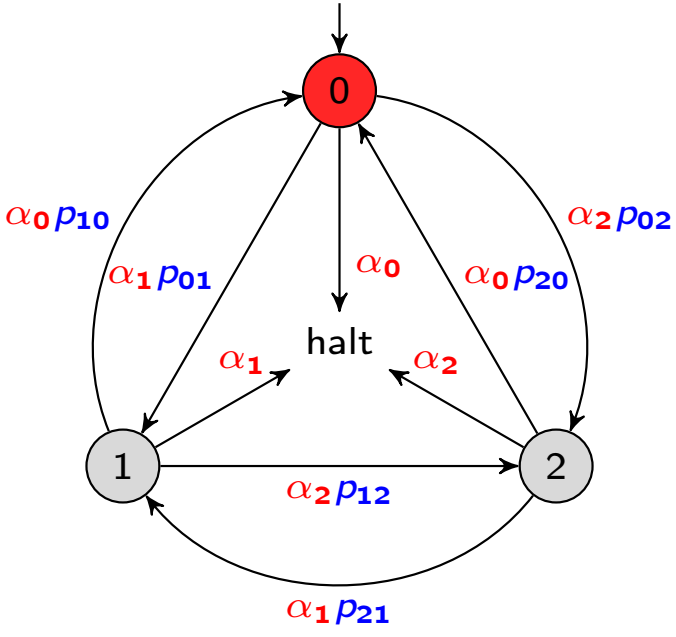


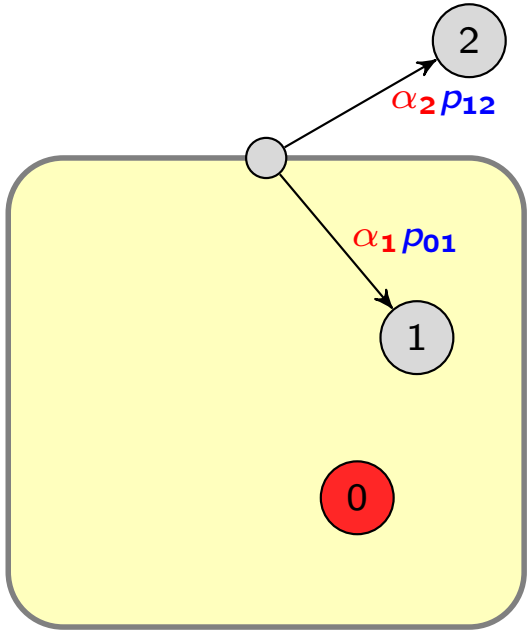
A 3-State SDA not Equivalent to Any While Program



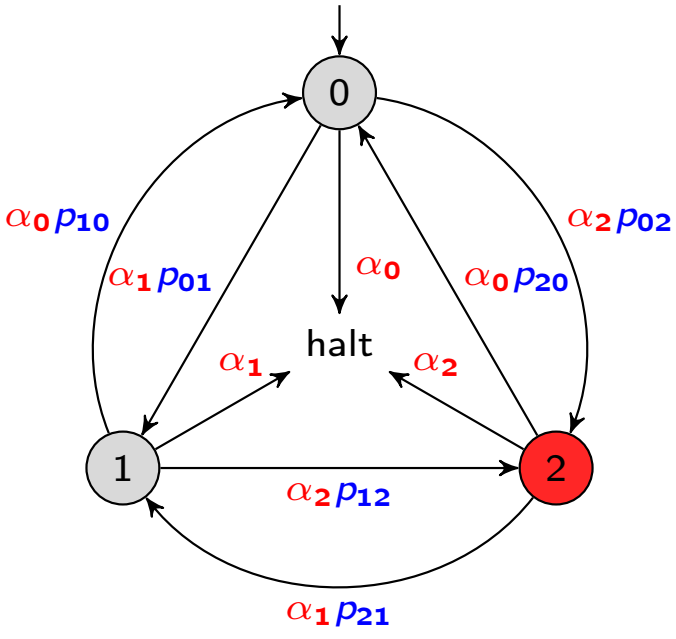


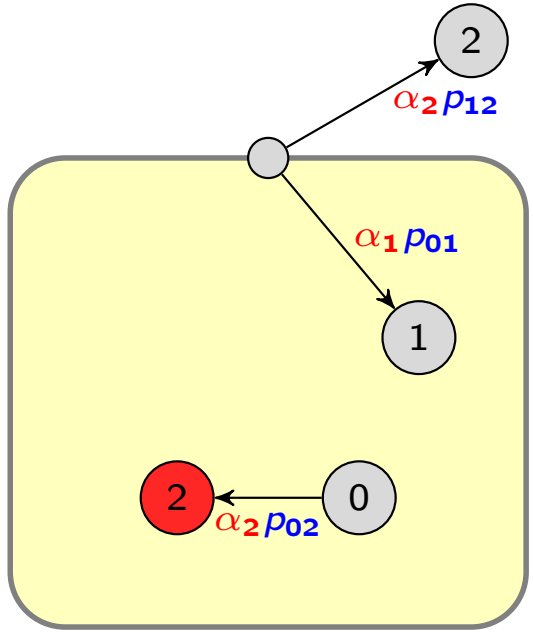
A 3-State SDA not Equivalent to Any While Program



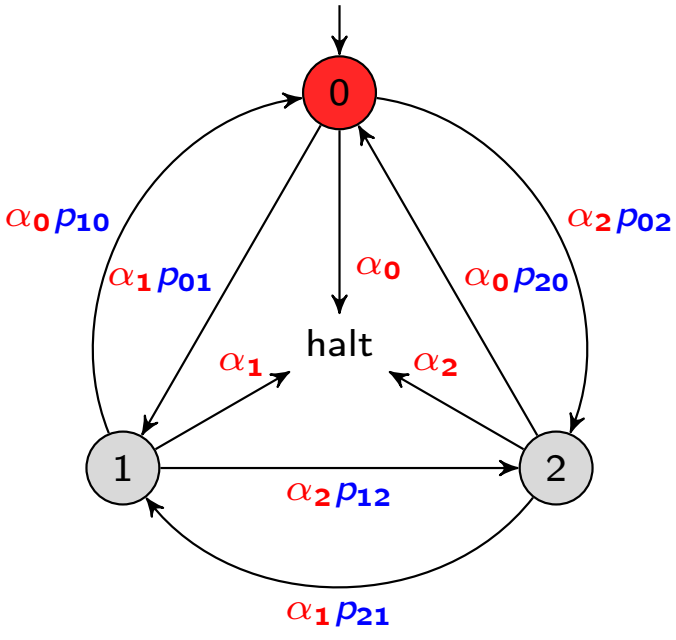


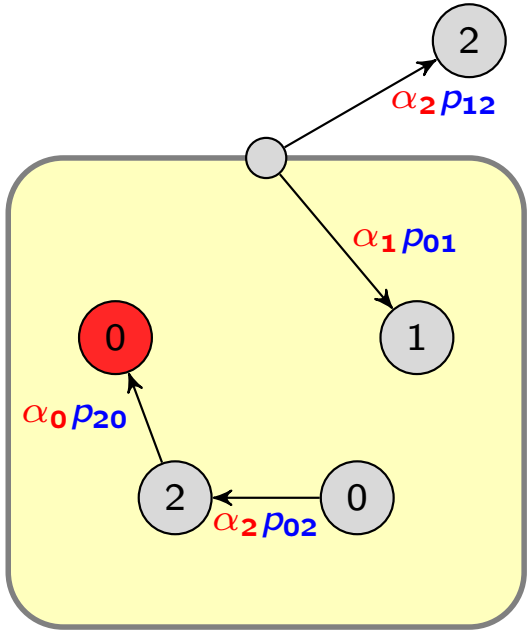
A 3-State SDA not Equivalent to Any While Program



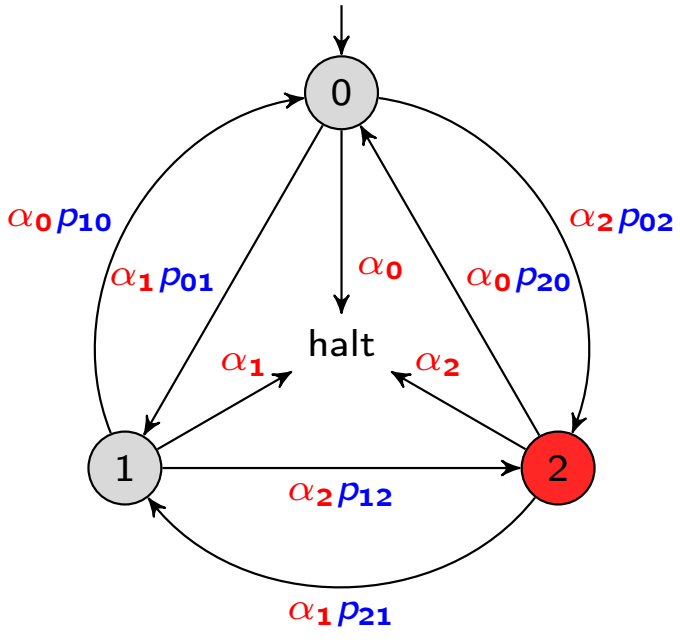


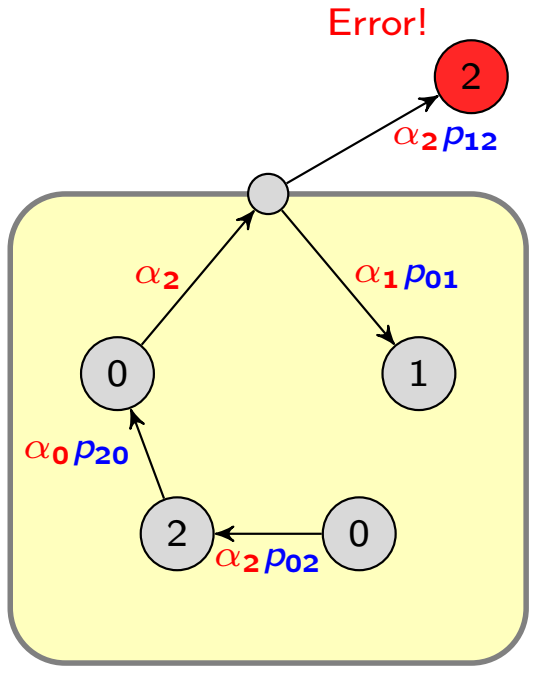
A 3-State SDA not Equivalent to Any While Program



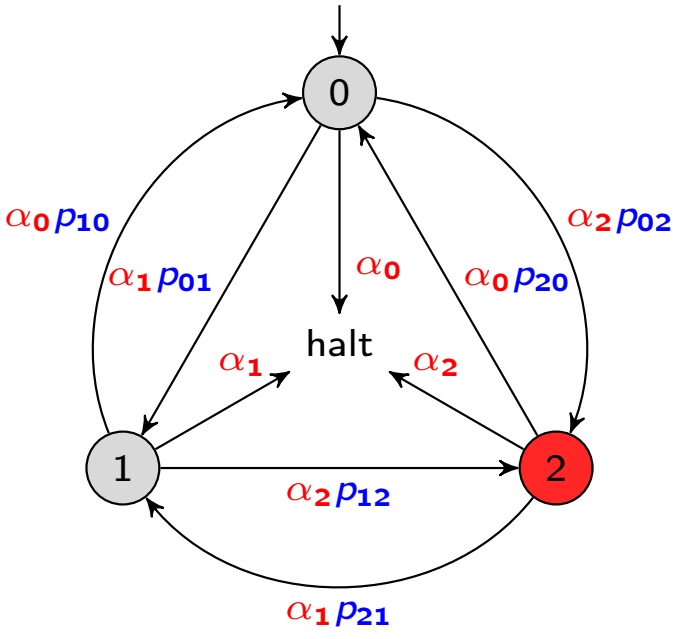


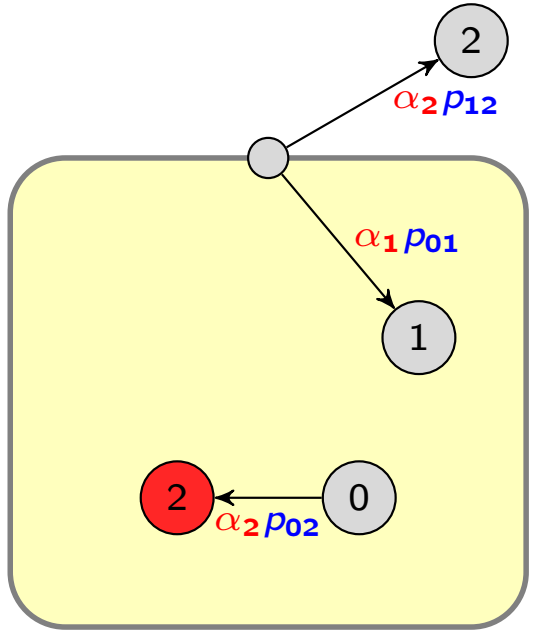
A 3-State SDA not Equivalent to Any While Program



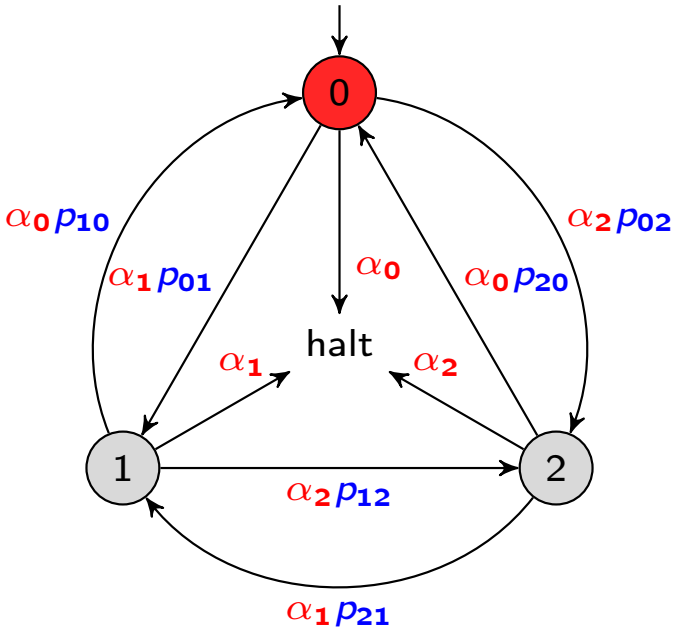


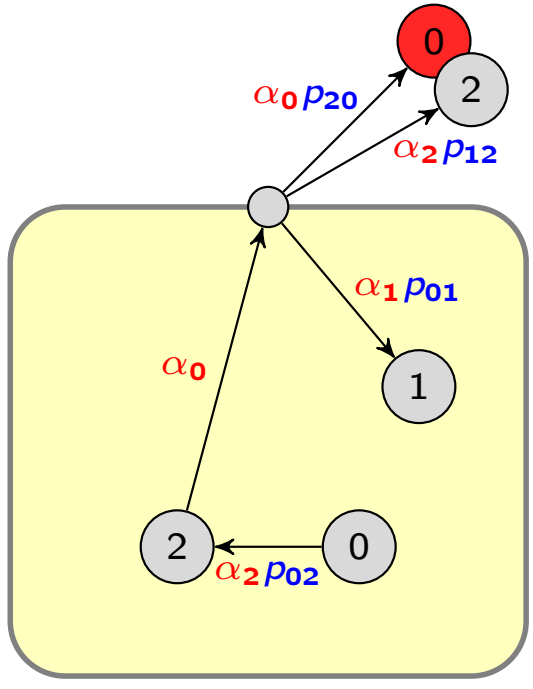
A 3-State SDA not Equivalent to Any While Program



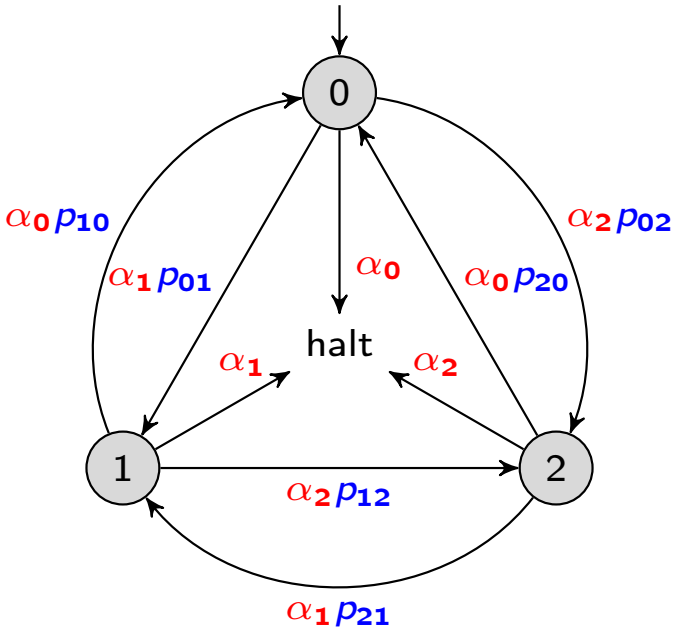


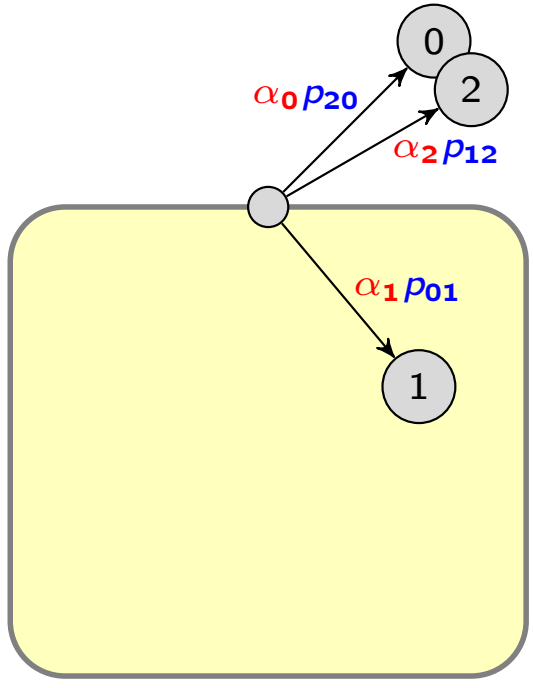
A 3-State SDA not Equivalent to Any While Program



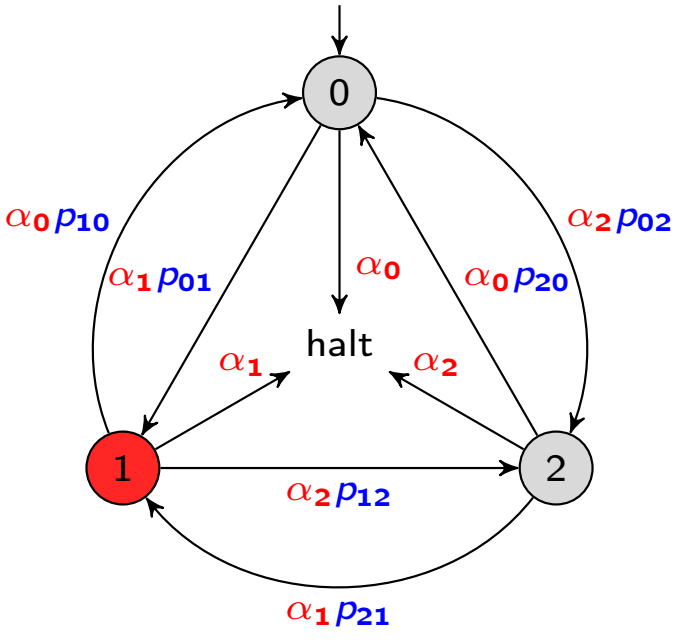


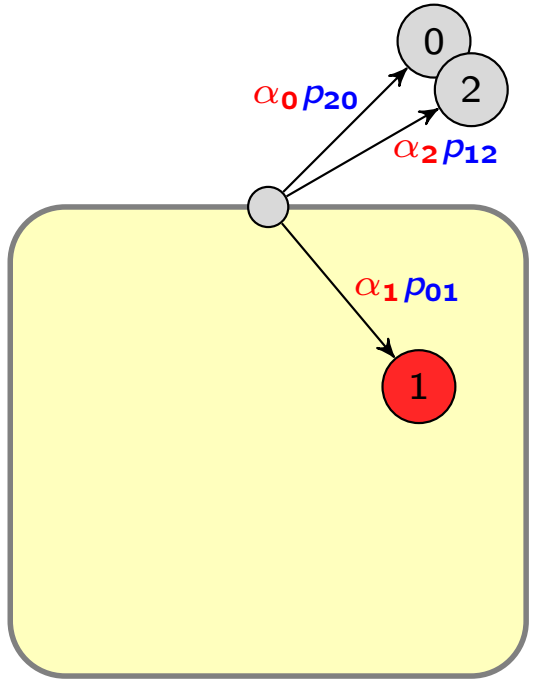
A 3-State SDA not Equivalent to Any While Program



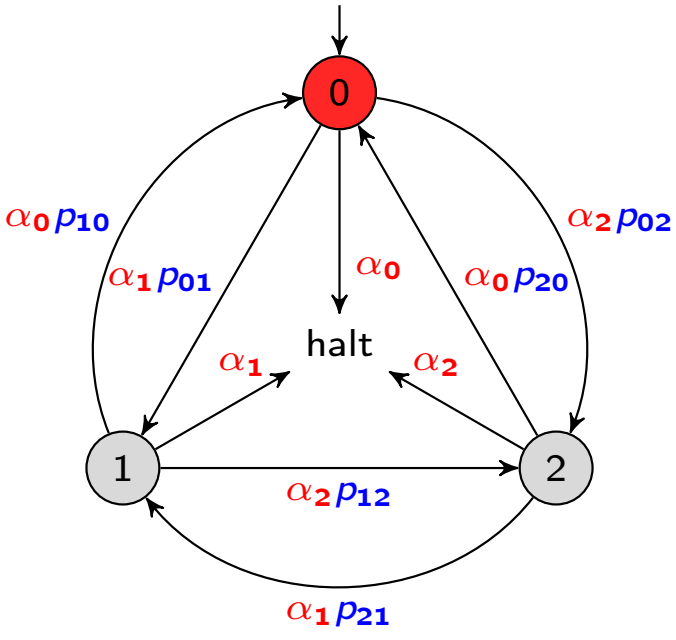


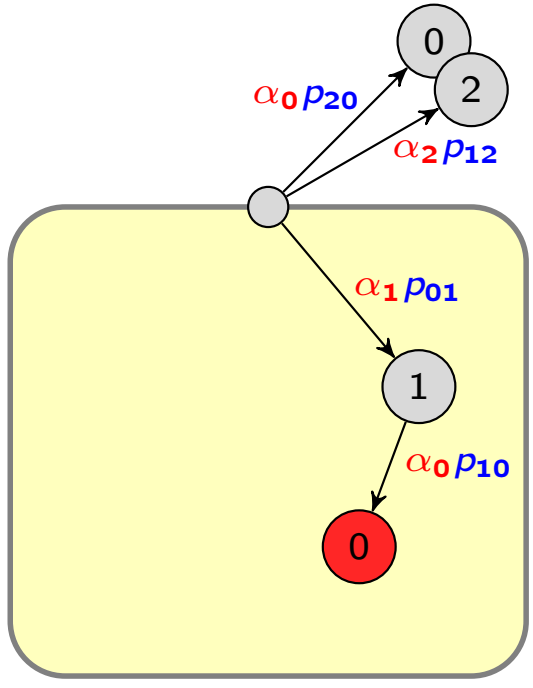
A 3-State SDA not Equivalent to Any While Program



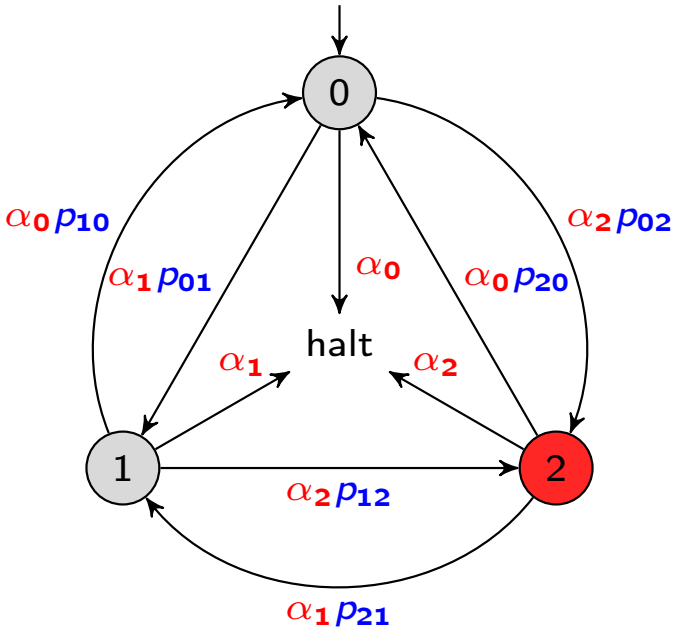


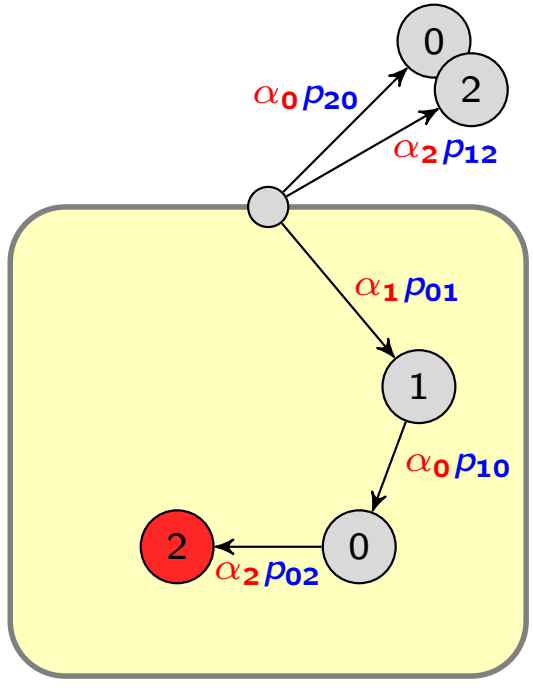
A 3-State SDA not Equivalent to Any While Program



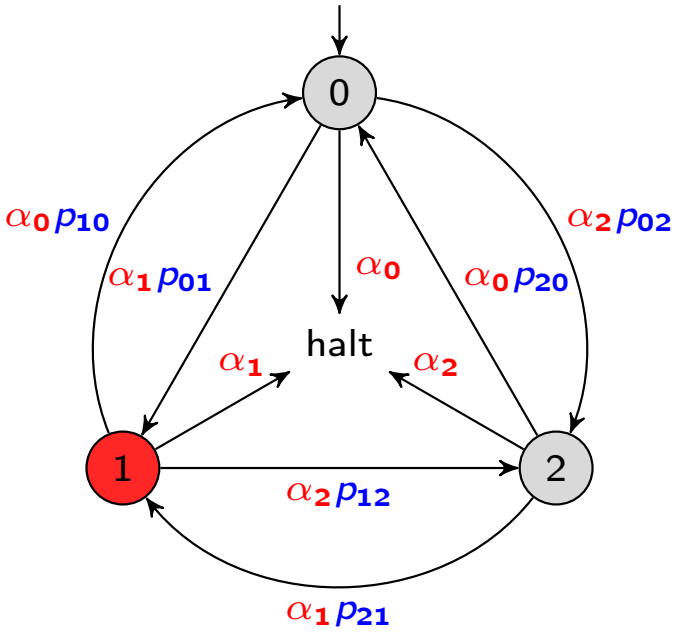


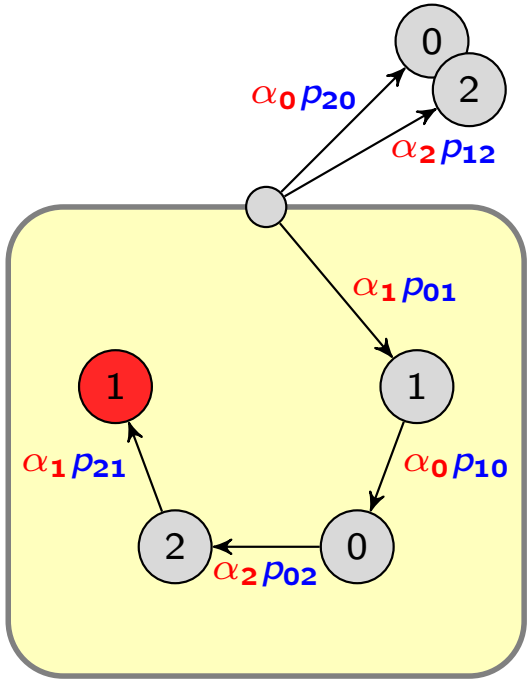
A 3-State SDA not Equivalent to Any While Program



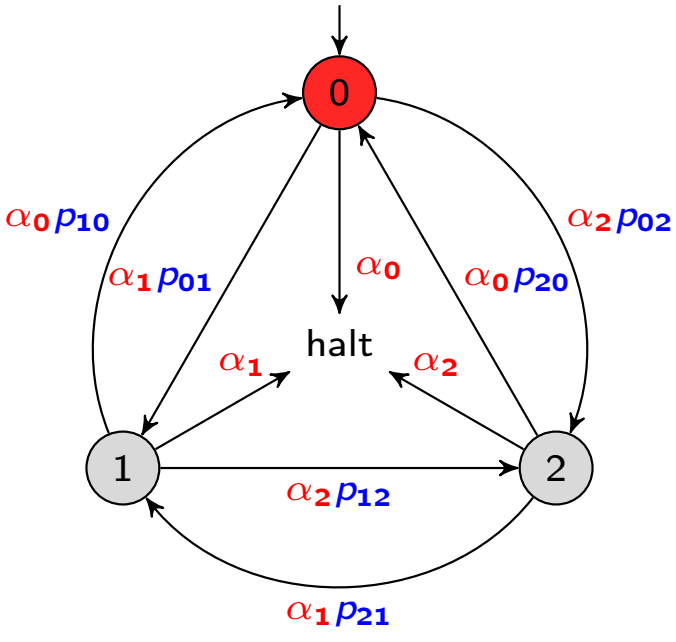


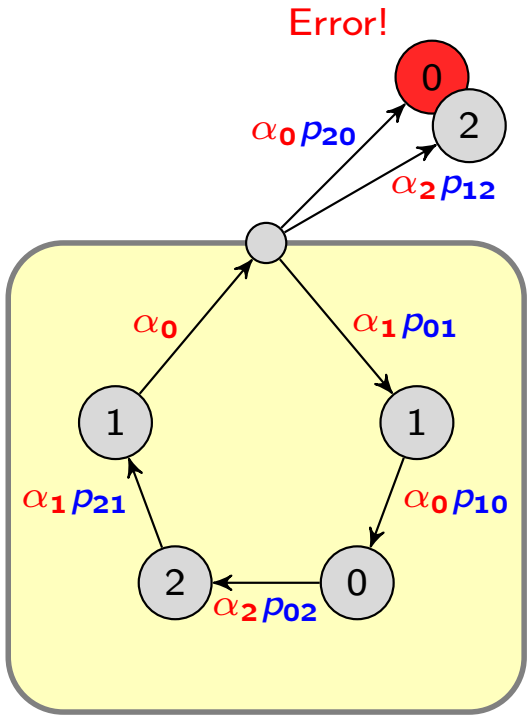
A 3-State SDA not Equivalent to Any While Program



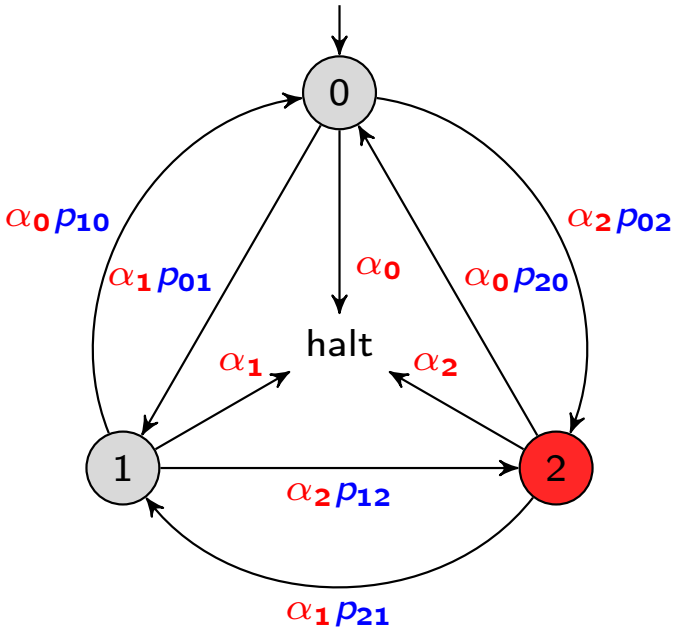


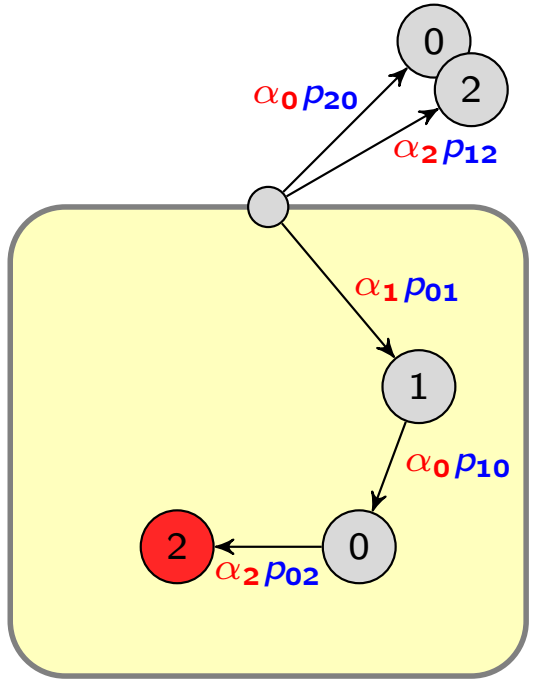
A 3-State SDA not Equivalent to Any While Program



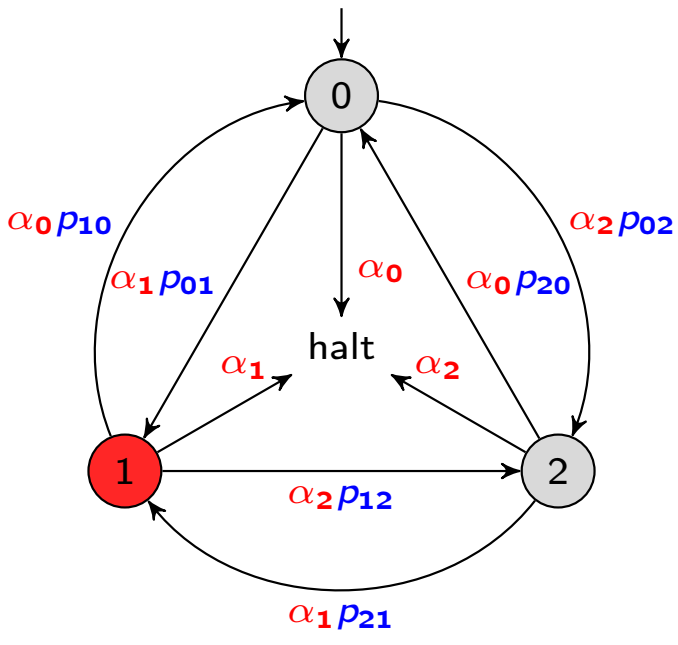


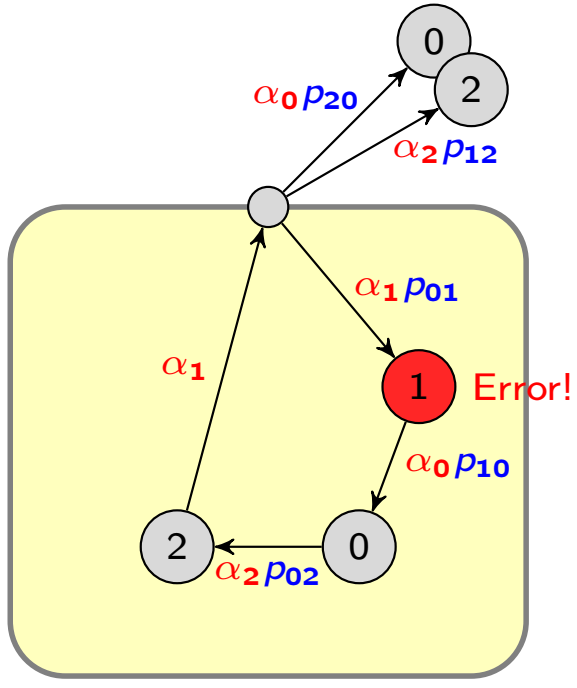
A 3-State SDA not Equivalent to Any While Program



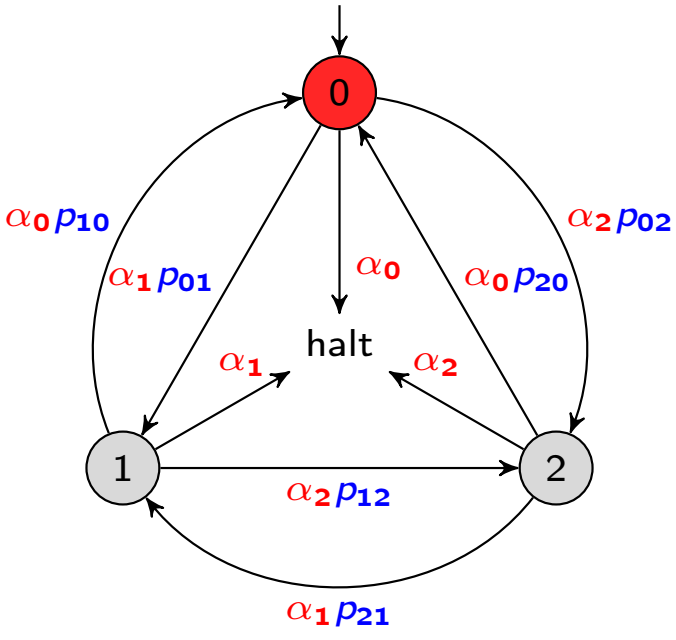


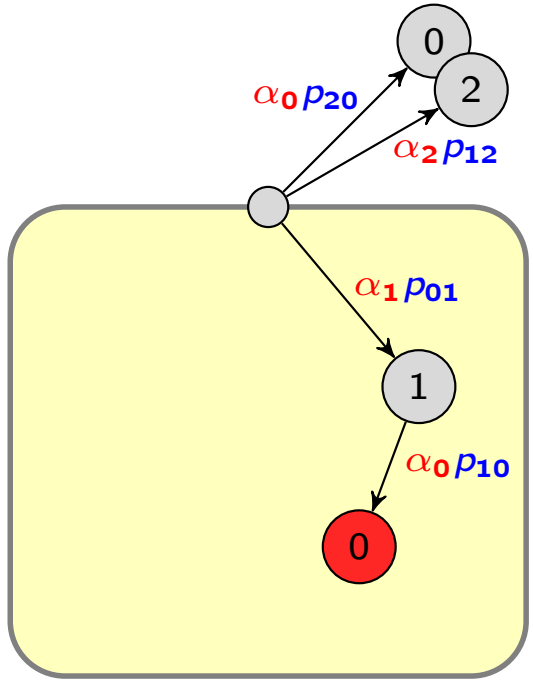
A 3-State SDA not Equivalent to Any While Program



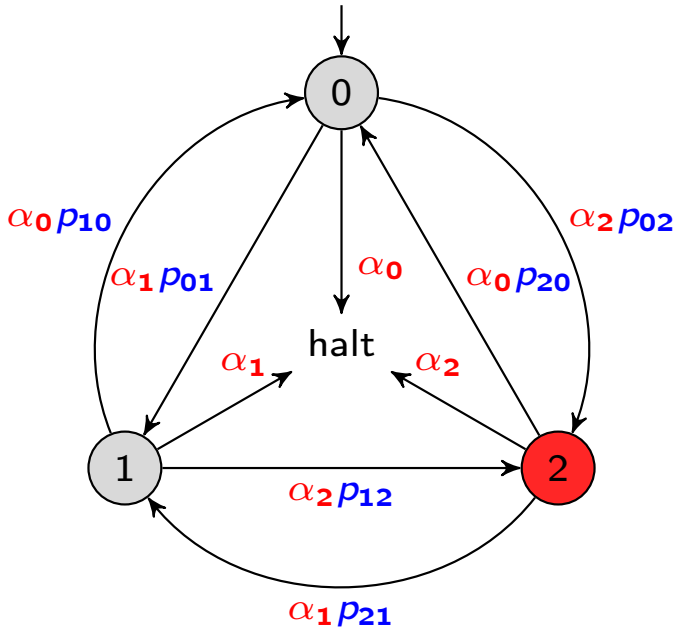


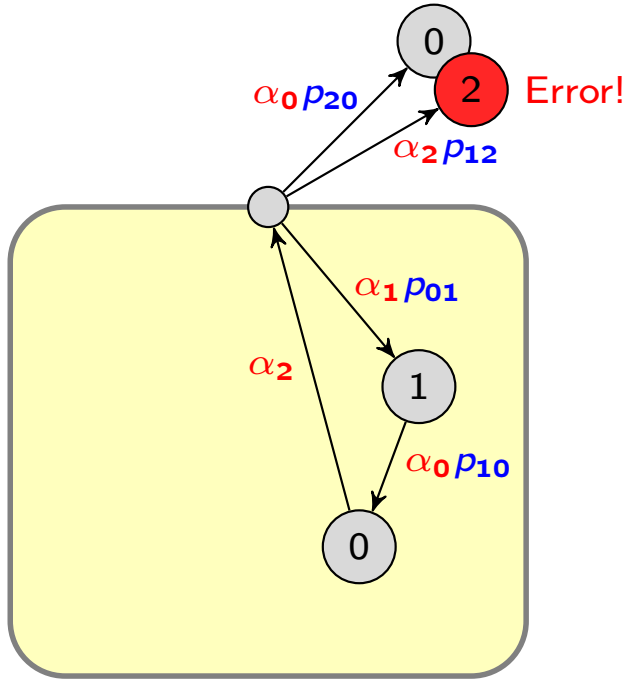
A 3-State SDA not Equivalent to Any While Program





A 3-State SDA not Equivalent to Any While Program

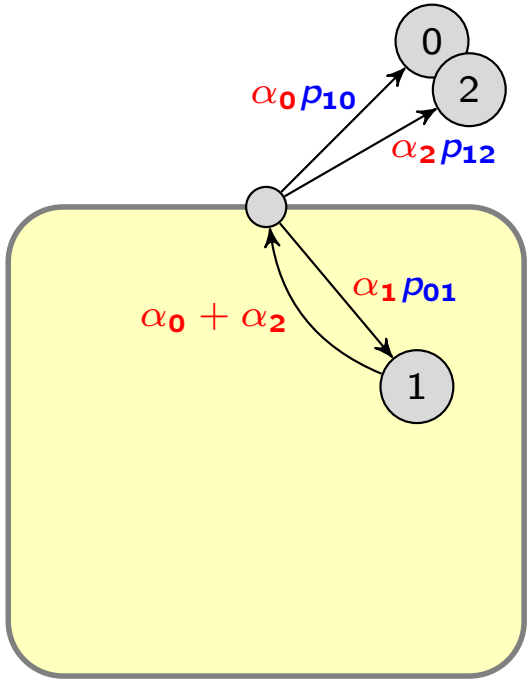




A 3-State SDA not Equivalent to Any While Program

If there is no state bisimilar to 0 or 2 inside the loop, the loop is equivalent to

```
while α1 {
   p01;
   if α1 then halt;
}
```



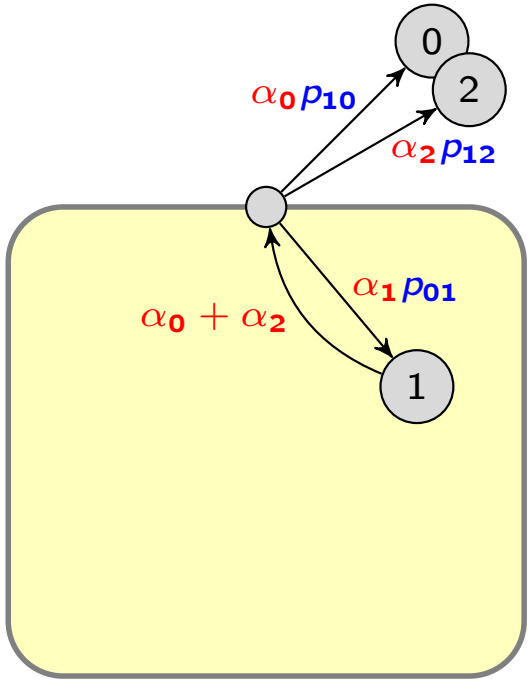
A 3-State SDA not Equivalent to Any While Program

If there is no state bisimilar to 0 or 2 inside the loop, the loop is equivalent to

```
while α1 {
    p01;
    if α1 then halt;
}
```

. . . which is equivalent to

```
if α1 then {
    p01;
    if α1 then halt;
}
```

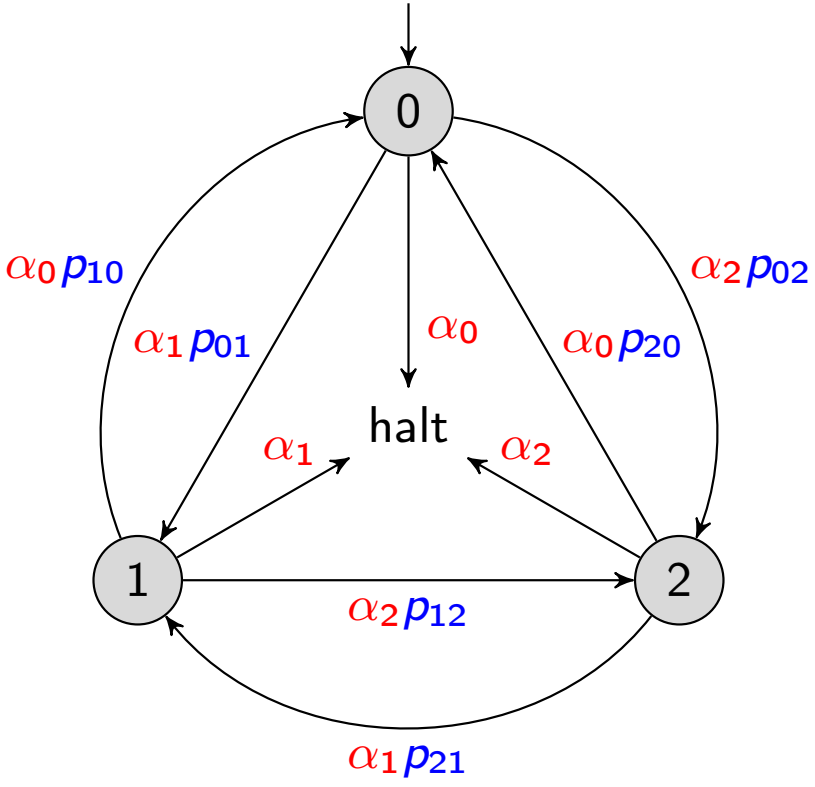


Loops Programs are Sufficient

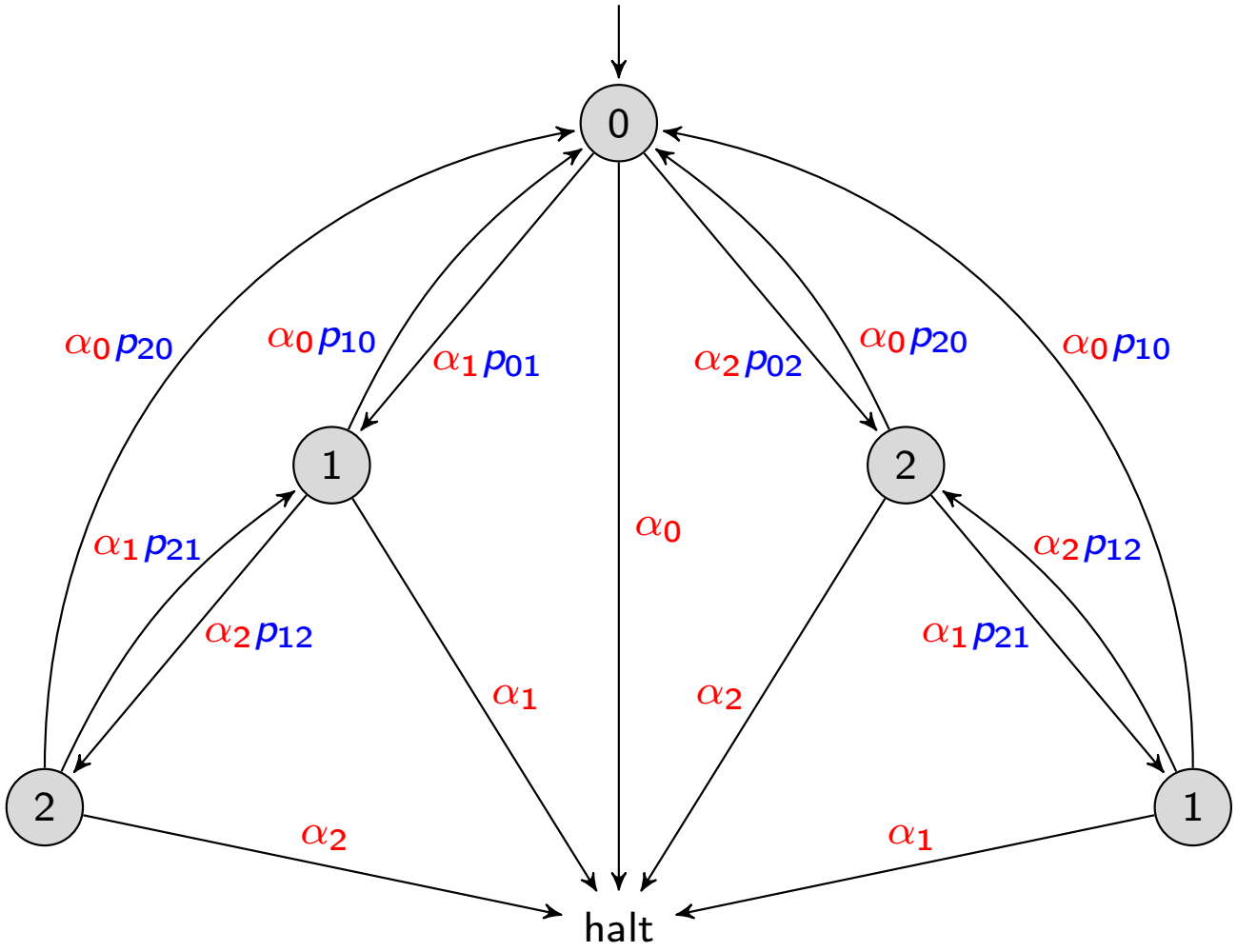
Theorem (Kosaraju 73)

Every program is equivalent to a program with loops and multilevel breaks but without gotos.

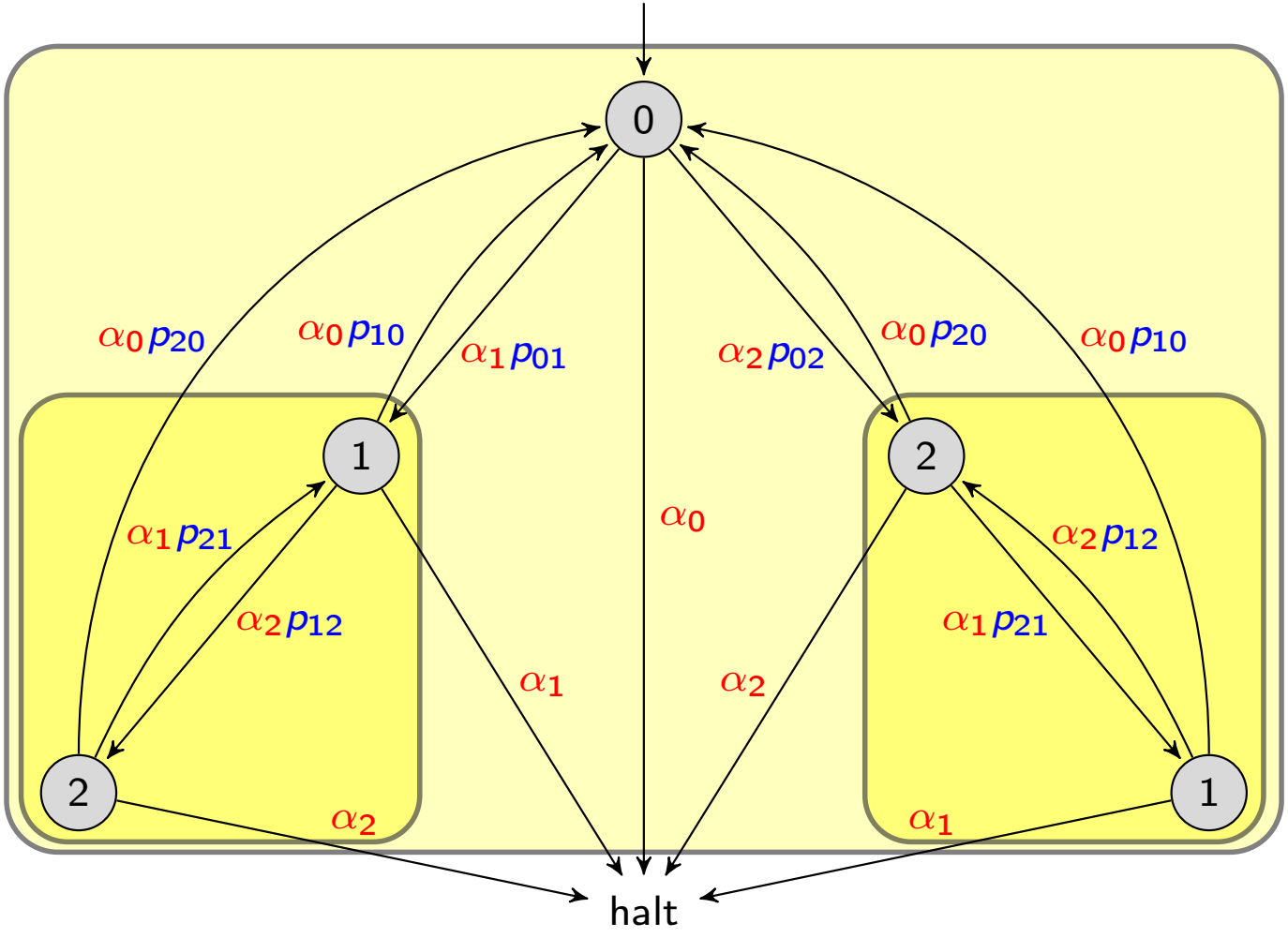
Example


0
1
2
halt
$\alpha_0 p_{10}$
$\alpha_1 p_{01}$
α_0
$\alpha_0 p_{20}$
$\alpha_2 p_{02}$
α_1
α_2
$\alpha_2 p_{12}$
$\alpha_1 p_{21}$

Example


0
1
2
2
1
halt
$\alpha_0 p_{20}$
$\alpha_0 p_{10}$
$\alpha_1 p_{01}$
$\alpha_2 p_{02}$
$\alpha_0 p_{20}$
$\alpha_0 p_{10}$
$\alpha_1 p_{21}$
$\alpha_2 p_{12}$
α_0
$\alpha_2 p_{12}$
$\alpha_1 p_{21}$
α_1
α_2
α_2
α_1

Example


0
1
2
2
1
halt
$\alpha_0 p_{20}$
$\alpha_0 p_{10}$
$\alpha_1 p_{01}$
$\alpha_2 p_{02}$
$\alpha_0 p_{20}$
$\alpha_0 p_{10}$
$\alpha_1 p_{21}$
$\alpha_2 p_{12}$
α_0
$\alpha_2 p_{12}$
$\alpha_1 p_{21}$
α_1
α_2
α_2
α_1

Example

```
loop {
    if α0 then break 1;
    if α1 then {
        p01;
        loop {
            if α1 then break 2;
            if α0 then {
                p10;
                break 1;
            } else {
                p12;
                if α2 then break 2;
                if α0 then {
                    p20;
                    break 1;
                }
                else p21;
            }
        }
    } else {
        p02;
        loop {
            if α2 then break 2;
            if α0 then {
                p20;
                break 1;
            } else {
                p21;
                if α1 then break 2;
                if α0 then {
                    p10;
                    break 1;
                }
                else p12;
            }
        }
    }
}
```

Equational Treatment of Nonlocal Control Flow

- A rigorous equational treatment of control constructs involving nonlocal transfer of control using KAT and AGS
  - goto ℓ
  - loop/break n
  - continue, exception handlers, try-catch-finally
- Compositional semantics
- Complete equational axiomatization

The Coalgebraic Theory

- Kleene coalgebra (KC) and Kleene coalgebra with tests (KCT)
- the Brzozowski derivative
- minimization as finality
- automatic extraction of equivalence proofs and relation to proof-carrying code

Automata as Coalgebras

algebra	coalgebra
constructors	destructors
initial algebras	final coalgebras
least fixpoints	greatest fixpoints

Example: Σ^ω = infinite streams over Σ with operations

- $head(a_0 a_1 a_2 a_3 \cdots) = a_0$
- $tail(a_0 a_1 a_2 a_3 \cdots) = a_1 a_2 a_3 \cdots$

A coalgebra of this signature is $(S, obs, cont)$, where $obs : S \to \Sigma$ and $cont : S \to S$.

Every state $s \in S$ generates a unique stream
$h(s) = obs(s), obs(cont(s)), obs(cont^2(s)), \ldots$.

The streams $(\Sigma^\omega, head, tail)$ form the final coalgebra, and the map h is the unique homomorphism $(S, obs, cont) \to (\Sigma^\omega, head, tail)$.

Automata as Coalgebras

Rutten 1998, Bonsangue 2008, Silva 2010

- Automata $\approx$ coalgebras
- Myhill-Nerode state minimization $\approx$ quotient by maximal bisimulation
- minimal automaton $\approx$ final coalgebra
- Brzozowski derivative $\approx$ a coalgebra of expressions

Kleene Coalgebra (KC)

A deterministic automaton without a start state

A Kleene coalgebra (KC) over Σ is a structure (S, δ, F), where

- S is a set of states
- $\delta_p : S \to S$, $p \in \Sigma$ is the transition function
- $F : S \to 2$ are the accept states.

Define $L : S \to \Sigma^* \to 2$ (i.e., $L : S \to 2^{\Sigma^*}$) coinductively:

- $L(s)(\varepsilon) = F(s)$
- $L(s)(px) = L(\delta_p(s))(x)$

Then

- $L(s)(x) = 1$ iff x is accepted starting from state s
- L is the unique KC-homomorphism to the final coalgebra
- its kernel is the unique maximal autobisimulation

Brzozowski Derivative (Brzozowski 1964)

A certain Kleene coalgebra $(2^{\Sigma^*}, D, E)$:

- $D_p : 2^{\Sigma^*} \to 2^{\Sigma^*}$ defined by $D_p(A) = \{x \mid px \in A\}$
- $E : 2^{\Sigma^*} \to 2$ defined by $E(A) = \begin{cases} 1 & \text{if } \varepsilon \in A \\ 0 & \text{if } \varepsilon \notin A \end{cases}$

This is the final KC over Σ

Brzozowski Derivative, Syntactic Form

Another Kleene coalgebra $(\mathrm{RExp}_\Sigma, D, E)$:

$$D_p : \mathrm{RExp}_\Sigma \to \mathrm{RExp}_\Sigma \qquad E : \mathrm{RExp}_\Sigma \to 2$$

$$\begin{aligned}
D_p(e + e') &= D_p(e) + D_p(e') & E(e + e') &= E(e) + E(e') \\
D_p(ee') &= D_p(e) \cdot e' + E(e) \cdot D_p(e') & E(ee') &= E(e) \cdot E(e') \\
D_p(e^*) &= D_p(e) \cdot e^* & E(e^*) &= 1 \\
D_p(p) &= 1 & E(p) &= 0, \ p \in \Sigma \\
D_p(q) &= 0, \ q \neq p & E(1) &= 1 \\
D_p(1) &= D_p(0) = 0 & E(0) &= 0
\end{aligned}$$

- $L(e)$ = the regular subset of Σ^* represented by e
- Kernel of L is KA equivalence = maximal autobisimulation

Coinductive Equivalence Proofs

To prove $e = e'$, suffices to establish a bisimulation $\equiv$ between $\{D_x(e) \mid x \in \Sigma^*\}$ and $\{D_x(e') \mid x \in \Sigma^*\}$ with $e \equiv e'$

Bisimulation:

- $e \equiv e' \Rightarrow D_p(e) \equiv D_p(e')$, $p \in \Sigma$
- $e \equiv e' \Rightarrow E(e) = E(e')$

The set $\{D_x(e) \mid x \in \Sigma^*\}$ is finite modulo the axioms of idempotent semirings

Can generate the maximal $\equiv$ automatically (if one exists)

Extension to KAT (Chen & Pucella 2003)

- Automatic proof generation for proof-carrying code (Necula & Lee 1997)
- There is a *PSPACE* decision procedure for KAT, but it only gives a one-bit answer
- Equational proofs require "cleverness", whereas coalgebraic proofs can be produced purely mechanically (Chen & Pucella 2003)
- Not true, actually (Worthington 2008)
  - can produce equational proofs in *PSPACE*
  - exponential length in the worst case (but probably unavoidable, since *PSPACE*-complete)

Kleene Coalgebra with Tests (KCT)

Deterministic AGS without a start state

A KCT over Σ, T is a structure (S, δ, ε), where

$$\delta : \mathrm{At} \cdot \Sigma \to S \to S \qquad \varepsilon : \mathrm{At} \to S \to 2$$
$$\delta_{\alpha p} : S \to S \qquad \varepsilon_\alpha : S \to 2$$

Define $L : S \to \mathcal{G} \to 2$ coinductively:

$$L(s)(\alpha) = \varepsilon_\alpha(s) \qquad L(s)(\alpha p x) = L(\delta_{\alpha p}(s))(x)$$

Then

- $L(s)(x) = 1$ iff x is accepted starting from state s
- L is the unique KCT-homomorphism to the final coalgebra

The Brzozowski Derivative

A KCT $(2^{\mathcal{G}}, D, E)$ where

$D : \mathrm{At} \cdot \Sigma \to 2^{\mathcal{G}} \to 2^{\mathcal{G}}$ $\qquad$ $E : \mathrm{At} \to 2^{\mathcal{G}} \to 2$

$D_{\alpha p} : 2^{\mathcal{G}} \to 2^{\mathcal{G}}$ $\qquad$ $E_\alpha : 2^{\mathcal{G}} \to 2$

$D_{\alpha p}(A) = \{x \mid \alpha p x \in A\}$ $\qquad$ $E_\alpha(A) = 1$ if $\alpha \in A$, 0 if $\alpha \notin A$

$L : 2^{\mathcal{G}} \to \mathcal{G} \to 2$

$L(A)(\alpha) = E_\alpha(A)$ $\qquad$ $L(A)(\alpha p x) = L(D_{\alpha p}(A))(x)$

$L(A)(x) = 1$ iff $x \in A$

Syntactic Form

Another KCT $(\mathrm{RExp}_{\Sigma,T},\ D,\ E)$

$D_{\alpha p} : \mathrm{RExp}_{\Sigma,T} \to \mathrm{RExp}_{\Sigma,T}$

$D_{\alpha p}(e + e') = D_{\alpha p}(e) + D_{\alpha p}(e')$

$D_{\alpha p}(ee') = D_{\alpha p}(e) \cdot e' + E_\alpha(e) \cdot D_{\alpha p}(e')$

$D_{\alpha p}(e^*) = D_{\alpha p}(e) \cdot e^*$

$D_{\alpha p}(p) = 1$

$D_{\alpha p}(q) = 0, q \neq p$

$D_{\alpha p}(1) = D_{\alpha p}(0) = 0$

$E_\alpha : \mathrm{RExp}_{\Sigma,T} \to 2$

$E_\alpha(e + e') = E_\alpha(e) + E_\alpha(e')$

$E_\alpha(ee') = E_\alpha(e) \cdot E_\alpha(e')$

$E_\alpha(e^*) = 1$

$E_\alpha(p) = 0,\ p \in \Sigma$

$$E_\alpha(b) = \begin{cases} 1 & \text{if } \alpha \leq b,\ b \in B \\ 0 & \text{if not} \end{cases}$$

$L(e)$ = the regular set of guarded strings represented by e

Complexity

Lemma

Any KAT *expression* e *has at most* $2^{|e|}$ *derivatives modulo the semiring axioms.*

Theorem

Can generate coinductive equivalence proofs (bisimulations) and inequivalence proofs (witnesses to the nonexistence of a bisimulation) automatically in PSPACE (matches (Worthington 2008) for equational proofs in KAT*)*

Proof.

Construct two nondeterministic finite automata by Kleene's theorem for KAT expressions, determinize by the subset construction, nondeterministically guess a counterexample to bisimiliarity in *PSPACE*. Make deterministic by Savitch's theorem. □

What I Haven't Talked About...

- alternative and weaker axiomatizations (left-handed, non-strict, non-idempotent...)
- total correctness, concurrency
- domain KA, Kleene modules
- ω-KA
- applications (compiler optimization, static analysis, pointer analysis, ...)
- complexity issues
- implementations

Thanks!