Kleene Algebra and Kleene Algebra with Tests

Part III

December 5, 2015

Today – The Coalgebraic Theory

- 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 with Tests

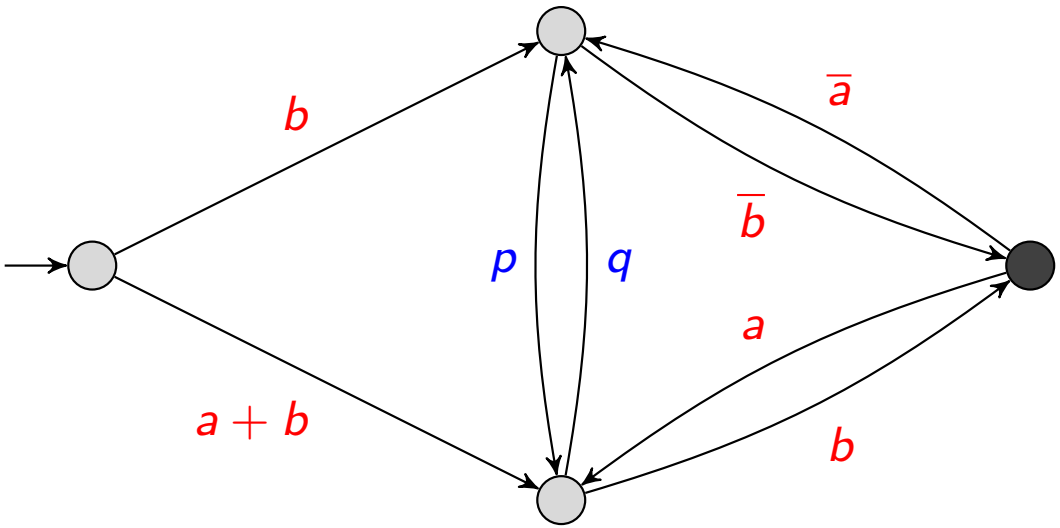
aka Automata on Guarded Strings

- 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 automaton with tests over the two-element Boolean algebra)
- Classical constructions of ordinary finite-state automata generalize readily
  - determinization
  - state minimization
  - Kleene's theorem

Automata with Tests

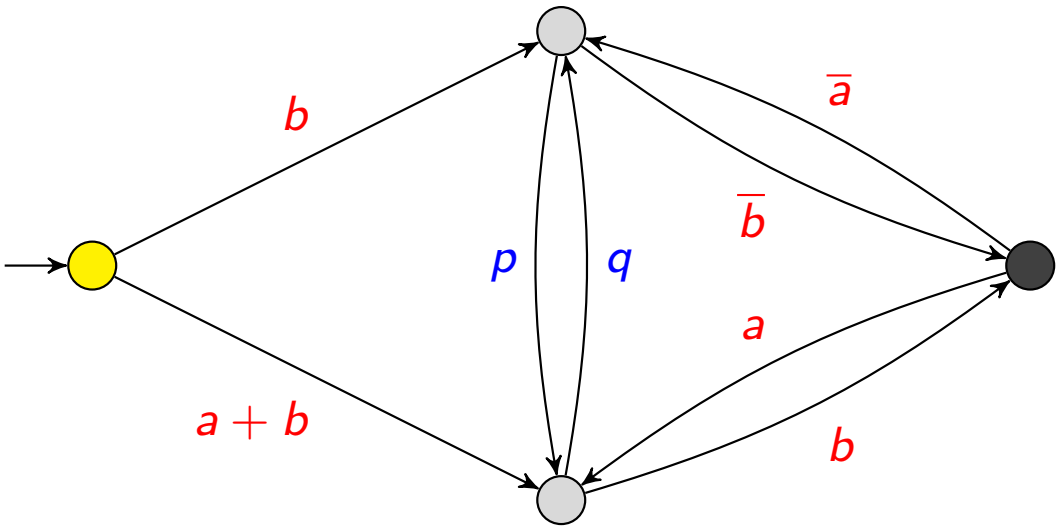
- atomic action symbols Σ, atomic test symbols T,
  B = the free Boolean algebra generated by T,
  At $= \{$atoms of $B\}$
- action transitions $s \xrightarrow{p} t$, $p \in \Sigma$
- test transitions $s \xrightarrow{b} t$, $b \in B$
- inputs are guarded strings $\alpha_0 q_1 \alpha_1 q_2 \cdots \alpha_{n-1} q_n \alpha_n$
- traces $s_0 q_1 s_1 q_2 \cdots s_{n-1} q_n s_n$ where $s_i \xrightarrow{q_{i+1}} s_{i+1}$
- x accepted if $\exists$ trace $s_0 q_1 s_1 q_2 \cdots s_{n-1} q_n s_n$ such that $s_0 \in S$, $s_n \in F$,
  $x \in G(q_1, \ldots, q_n)$

Automata with Tests



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

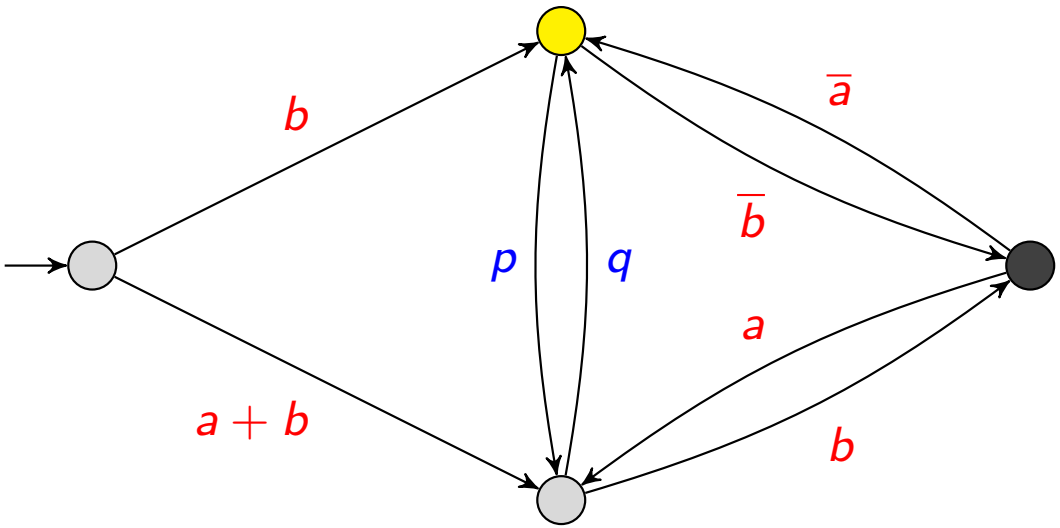
Automata with Tests



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

▲

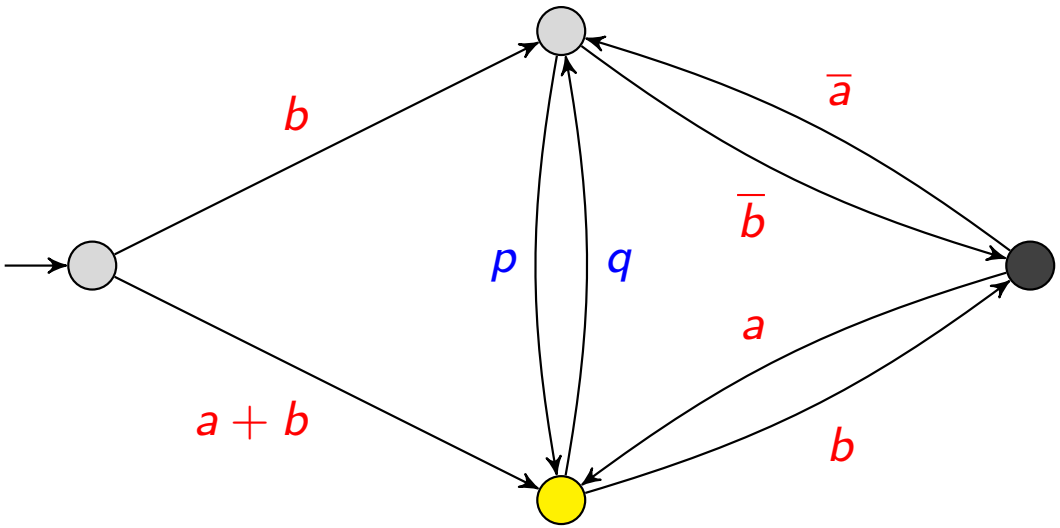
Automata with Tests



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

▲

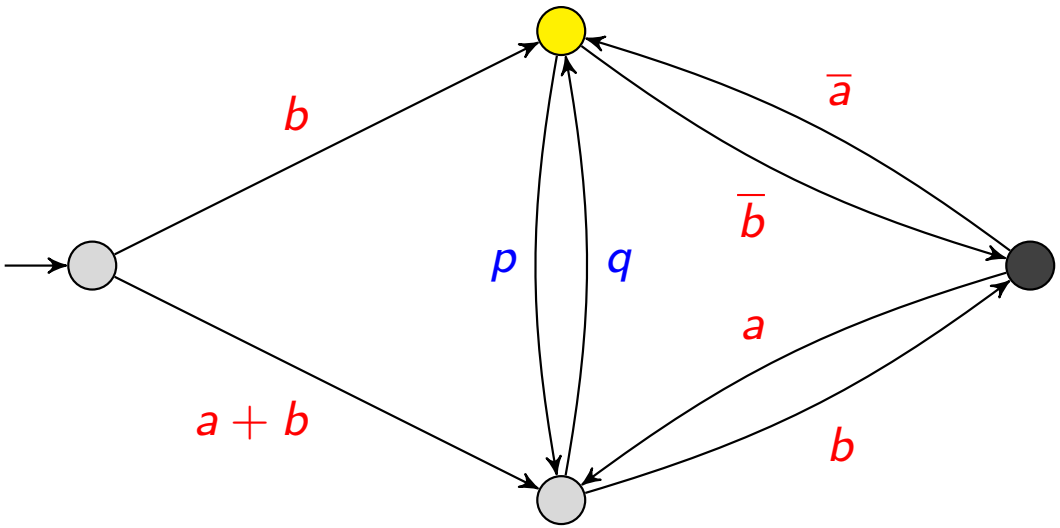
Automata with Tests



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

▲

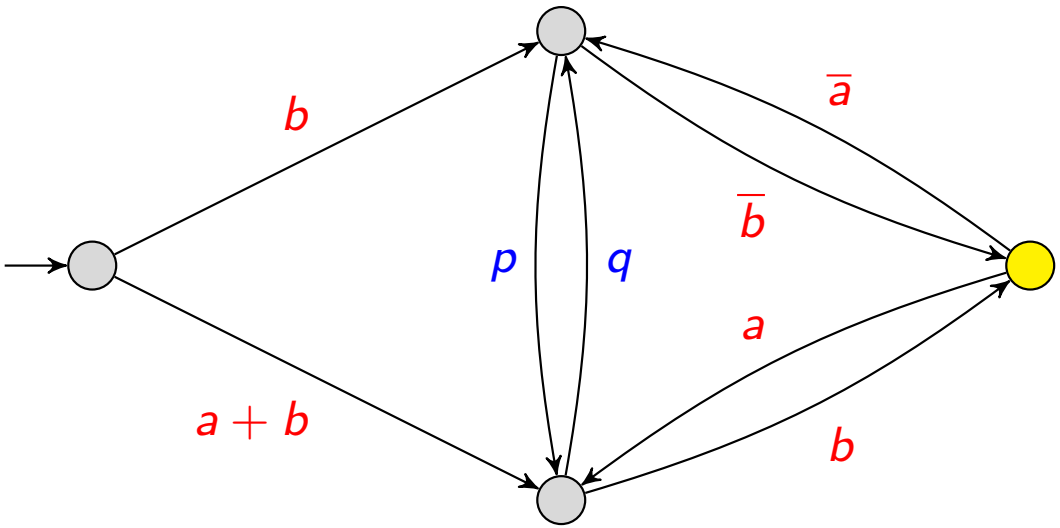
Automata with Tests



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

▲

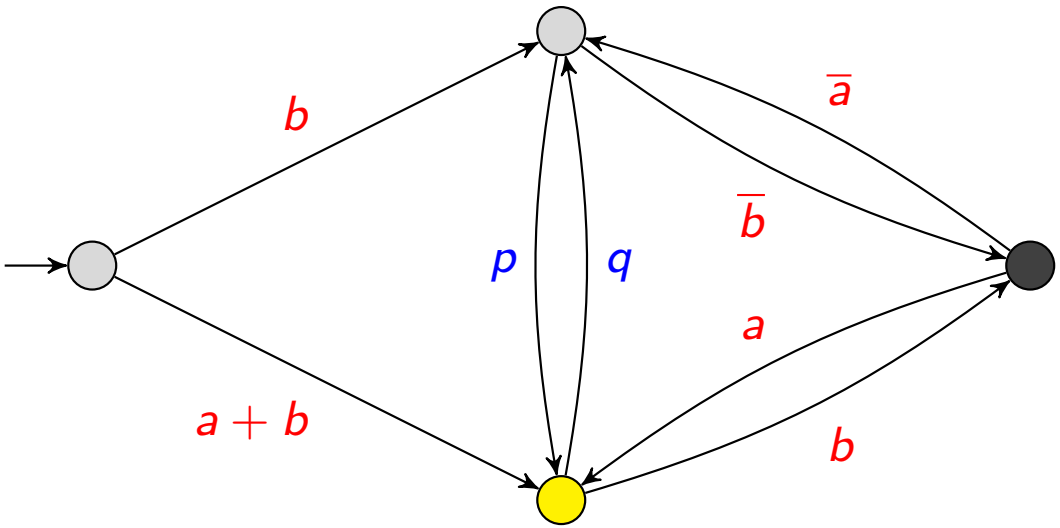
Automata with Tests



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

▲

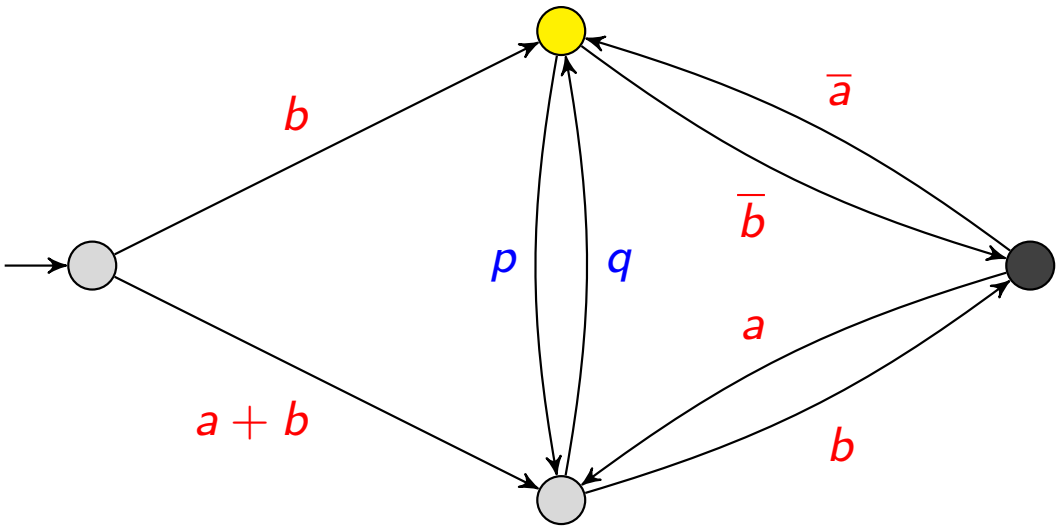
Automata with Tests



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

▲

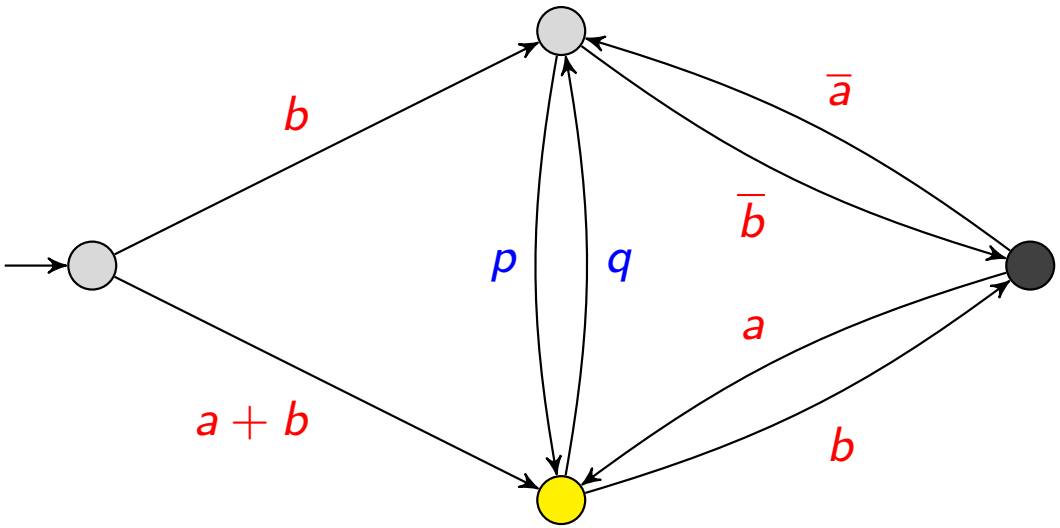
Automata with Tests



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

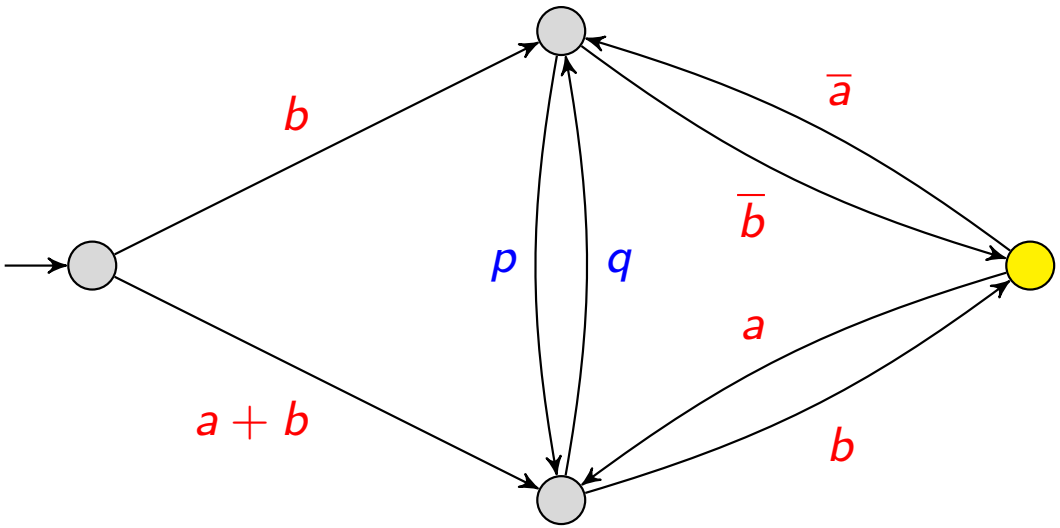
▲

Automata with Tests



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

Automata with Tests



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

▲ Accept!

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 deterministic automata are not unique; but they are unique if only atomic tests and their negations are allowed and tests are ordered
- minimal OBDDs are a special case of this construction

Modeling Flowchart Programs

Flowchart programs correspond to a limited class of automata with tests 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

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

(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:

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

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

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

Bisimulation Between Strictly Deterministic Automata

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

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

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

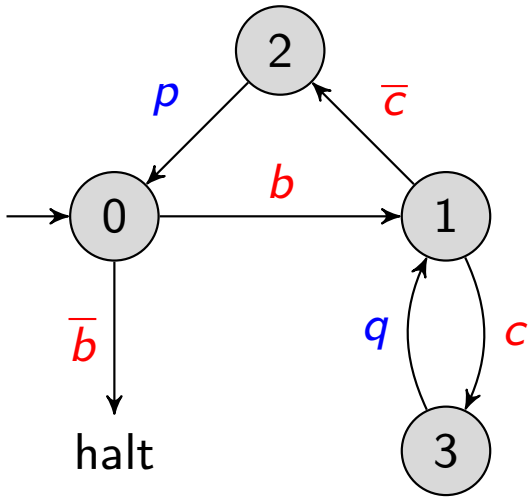
$$\begin{aligned} \mathsf{gs}_M(\mathsf{start}_M, -) : \mathsf{At}^\omega &\to (\mathsf{At} \cdot \Sigma)^* \cdot \mathsf{At} + (\mathsf{At} \cdot \Sigma)^\omega \\ \mathsf{gs}_N(\mathsf{start}_N, -) : \mathsf{At}^\omega &\to (\mathsf{At} \cdot \Sigma)^* \cdot \mathsf{At} + (\mathsf{At} \cdot \Sigma)^\omega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

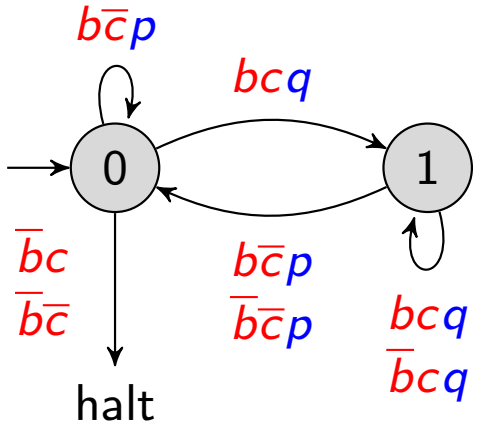
agree on the set $\{\sigma \mid \mathsf{gs}_M(\mathsf{start}_M, \sigma) \text{ is finite}\}$. This set is dense in At^ω. Moreover, the two functions are continuous, and continuous functions that agree on a dense set must agree everywhere. □

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

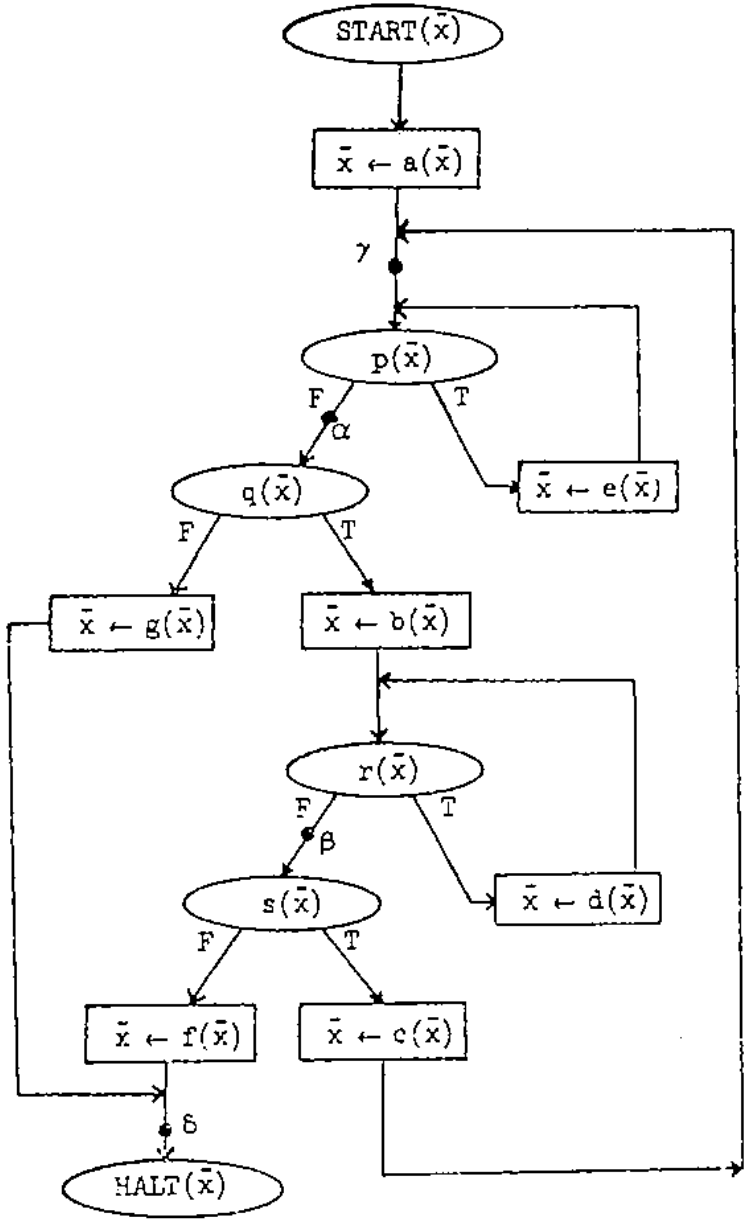
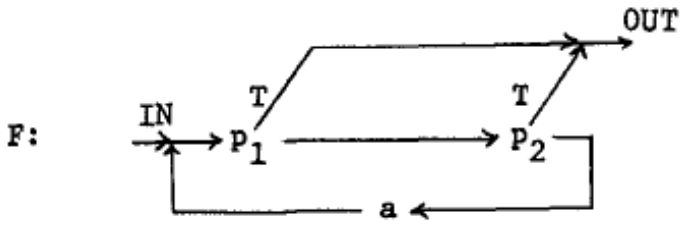


Fig. 1. The flowchart program P_1.

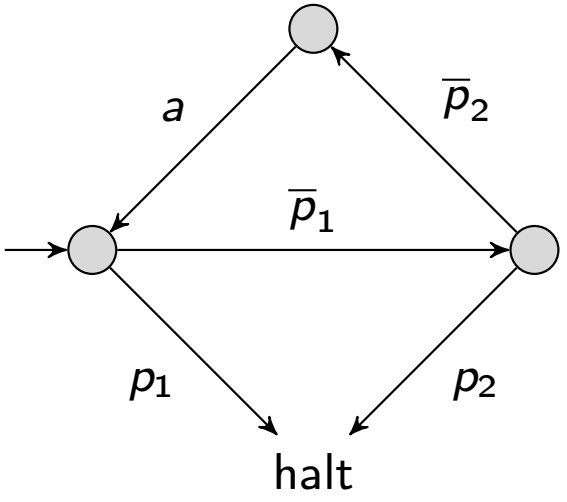
—from (Ashcroft & Manna 1972)

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:

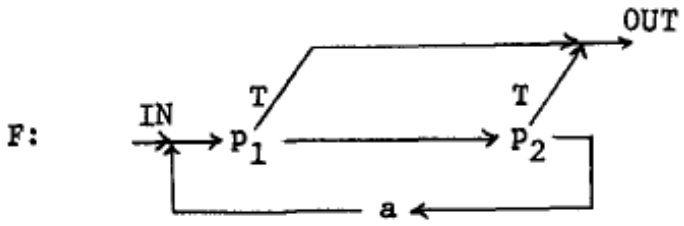


—from (Kosaraju 1973)

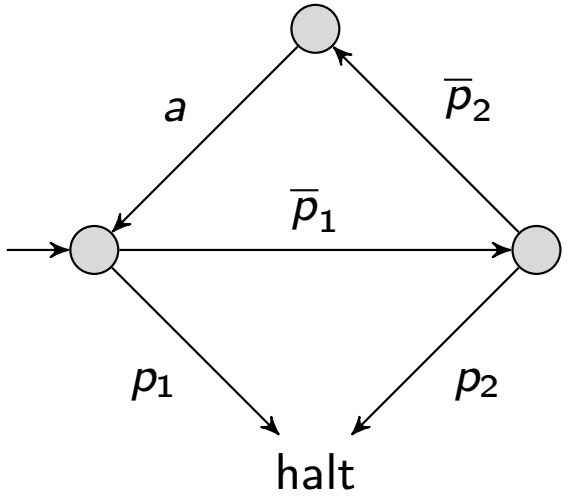


Kosaraju's Counterexample

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
    ⋮
}
```

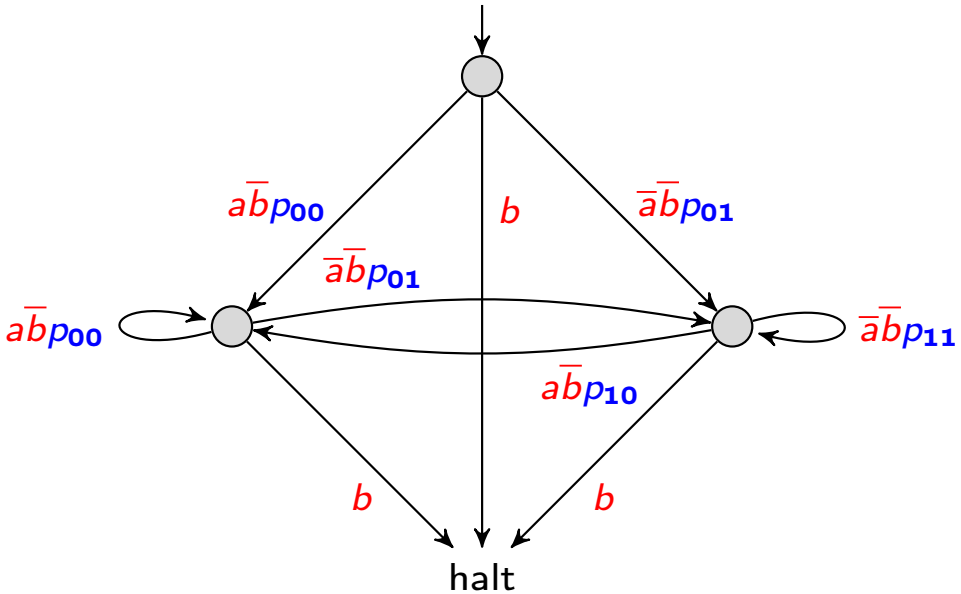
```
loop {
    ⋮
    loop {
        ⋮
        break 2;
        ⋮
    }
    ⋮
} ⟵ break 2 comes here
```

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).

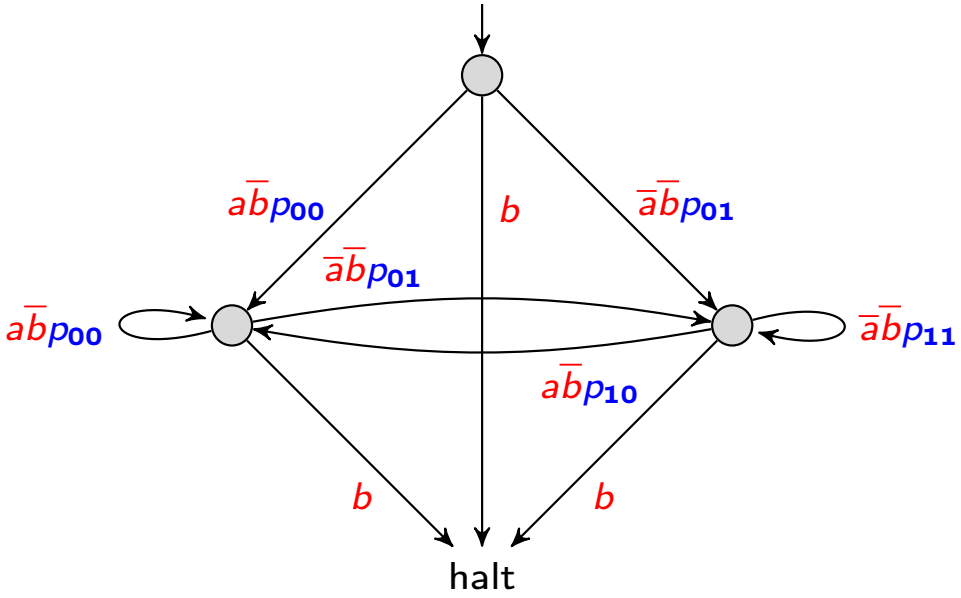
Proof: The proof is closely related to Böhm and Jacopini construction [2]. One should not have any problem in replacing the control variables used in Böhm and Jacopini's construction by RPT, END, EXIT constructs applicable to this class of flow charts. We leave the details to the reader. QED

—from (Kosaraju 1973)

A Two-State, Multi-entrance, Multi-exit Program



A Two-State, Multi-entrance, Multi-exit Program



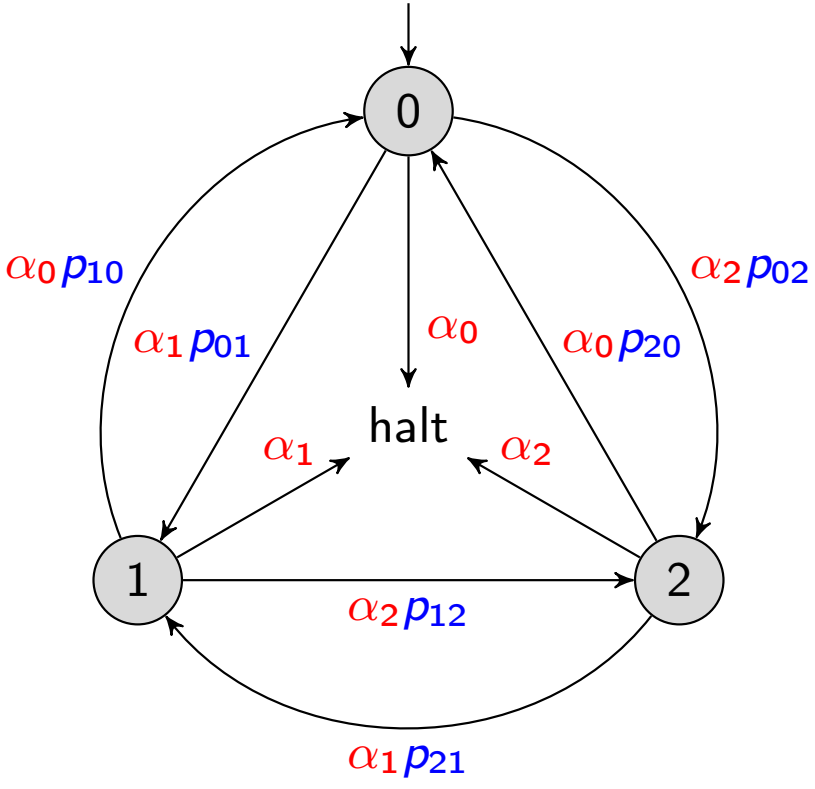

```
if b̄ then {
    if a then {
        p00;
        while b̄ do {
            while ab̄ do p00;
            while āb̄ do {
                p01;
                while āb̄ do p11;
                if b̄ then p10;
            }
        }
    } else {
        p01;
        while b̄ do {
            while āb̄ do p11;
            while ab̄ do {
                p10;
                while ab̄ do p00;
                if b̄ then p01;
            }
        }
    }
}
```

Use KAT to Prove Equivalence

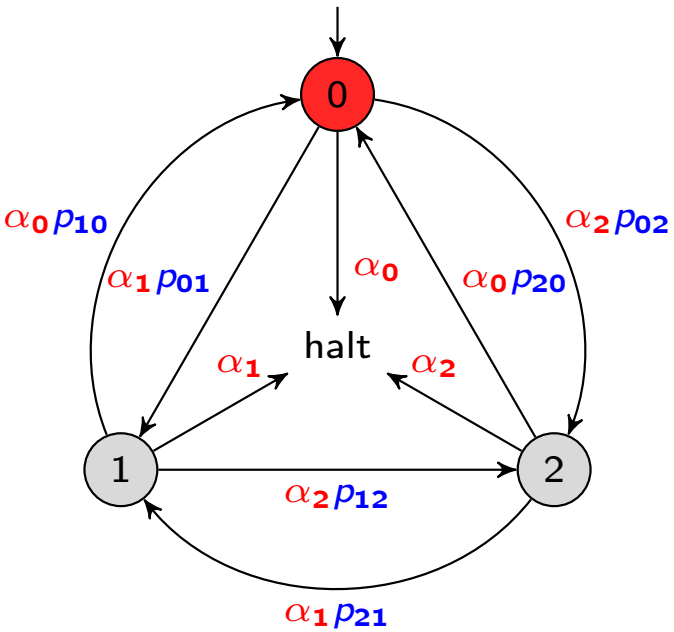
$$\overline{b}(ap_{00}(\overline{b}(a\overline{b}p_{00})^*\overline{a\overline{b}}(\overline{a}\overline{b}p_{01}(\overline{a}\overline{b}p_{11})^*\overline{\overline{a}\overline{b}}(\overline{b}p_{10}+b))^*\overline{\overline{a}\overline{b}})^*b$$
$$+\ \overline{a}p_{01}(\overline{b}(\overline{a}\overline{b}p_{11})^*\overline{\overline{a}\overline{b}}(a\overline{b}p_{10}(a\overline{b}p_{00})^*\overline{a\overline{b}}(\overline{b}p_{01}+b))^*\overline{a\overline{b}})^*b)$$
$$+\ b$$

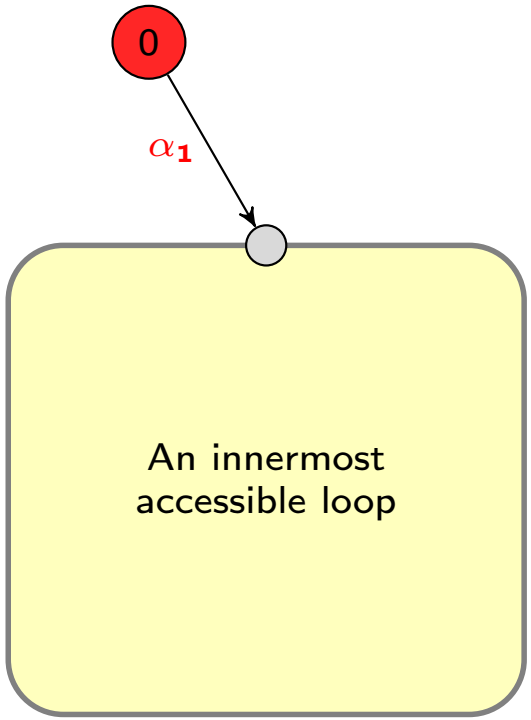
$$a\overline{b}p_{00}(a\overline{b}p_{00}+\overline{a}\overline{b}p_{01}(\overline{a}\overline{b}p_{11})^*a\overline{b}p_{10})^*(b+\overline{a}\overline{b}p_{01}(\overline{a}\overline{b}p_{11})^*b)$$
$$+\ \overline{a}\overline{b}p_{01}(\overline{a}\overline{b}p_{11}+a\overline{b}p_{10}(a\overline{b}p_{00})^*\overline{a}\overline{b}p_{01})^*(b+a\overline{b}p_{10}(a\overline{b}p_{00})^*b)$$
$$+\ b$$

A 3-State SDA not Equivalent to Any While Program

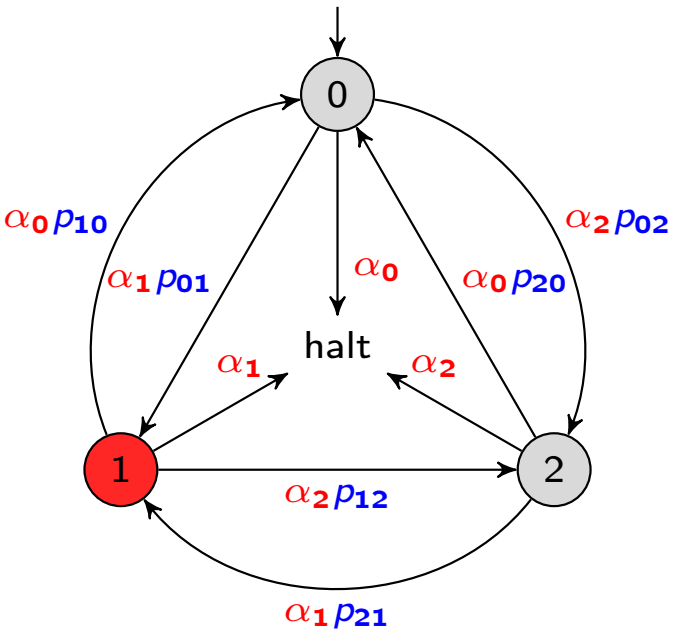


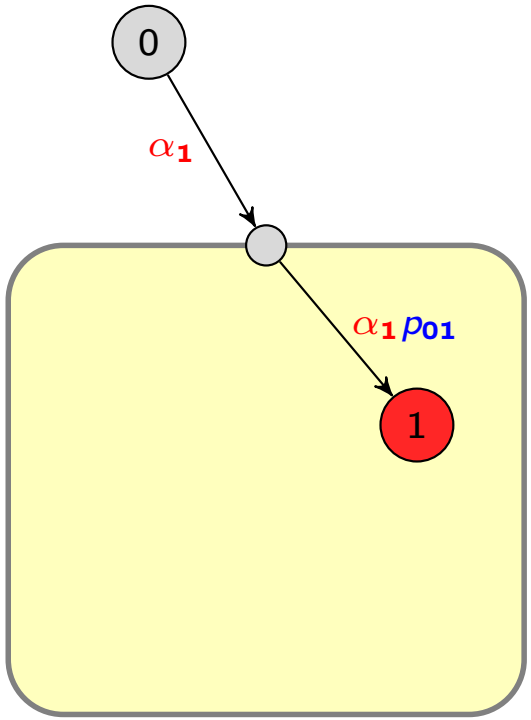
A 3-State SDA not Equivalent to Any While Program



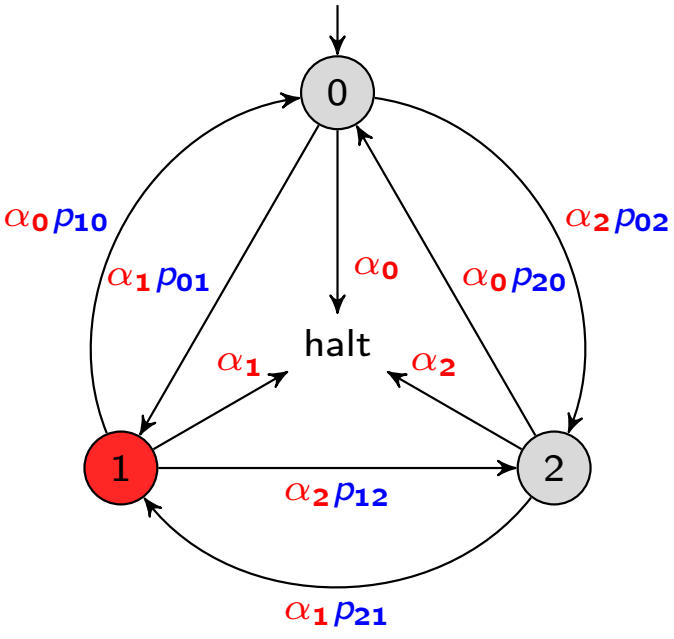


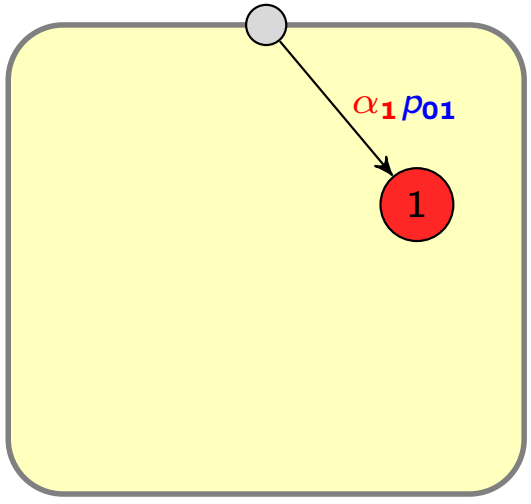
A 3-State SDA not Equivalent to Any While Program



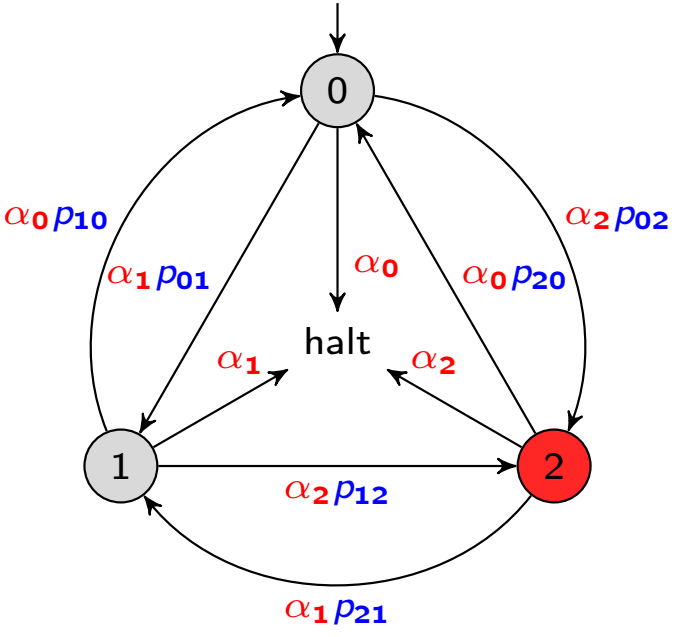


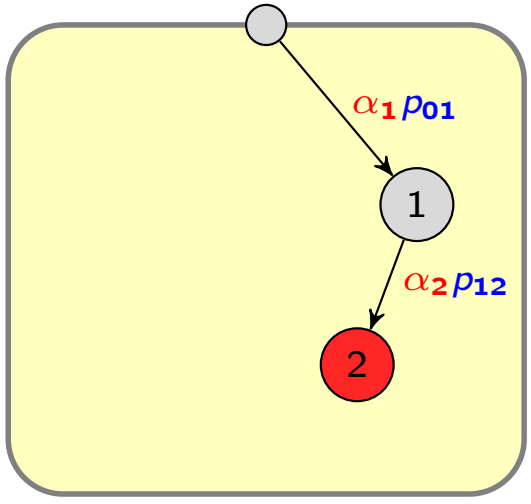
A 3-State SDA not Equivalent to Any While Program



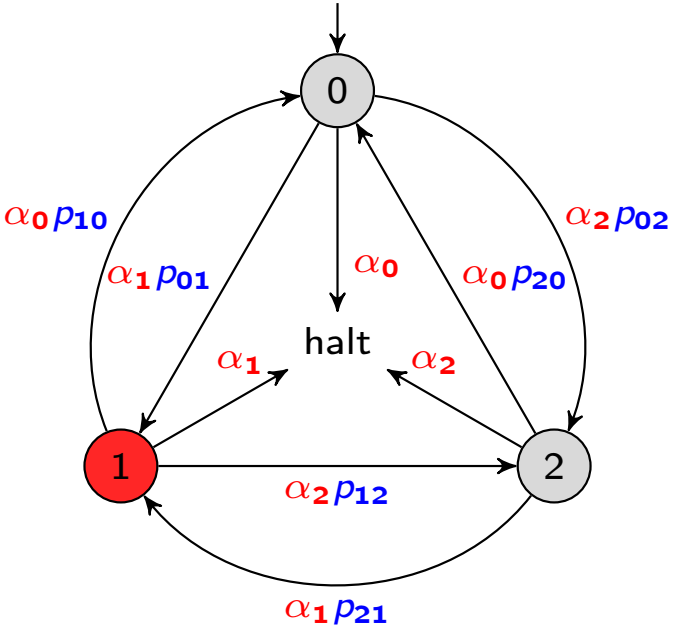


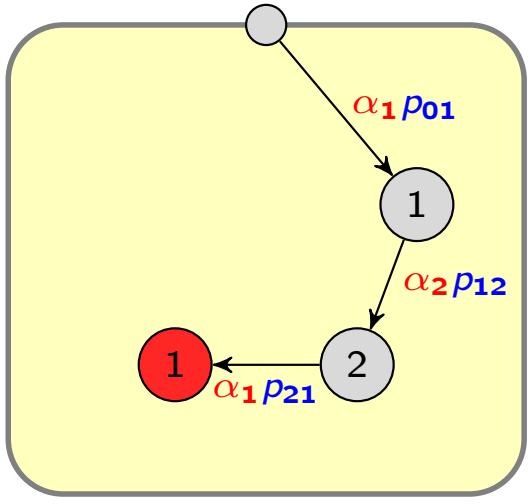
A 3-State SDA not Equivalent to Any While Program



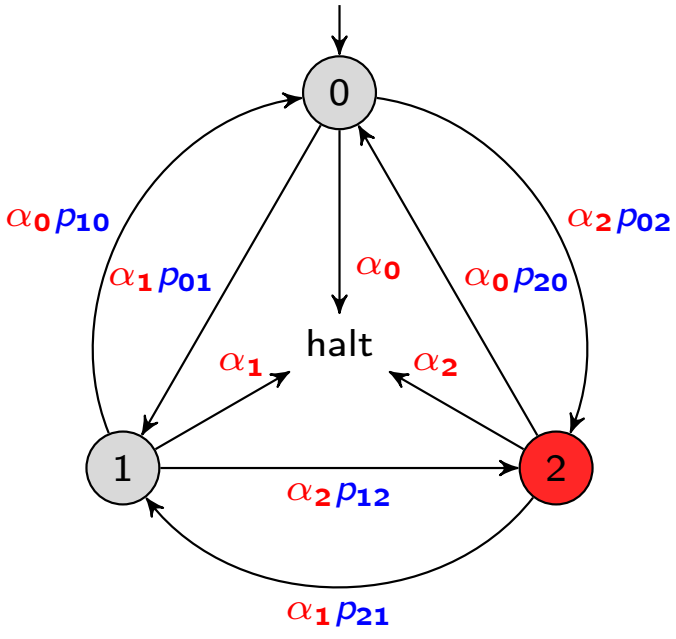


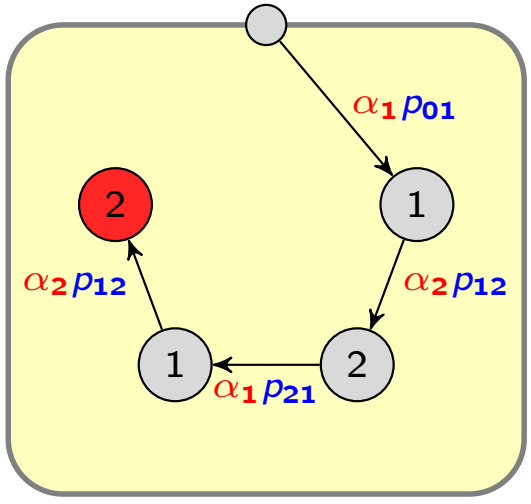
A 3-State SDA not Equivalent to Any While Program



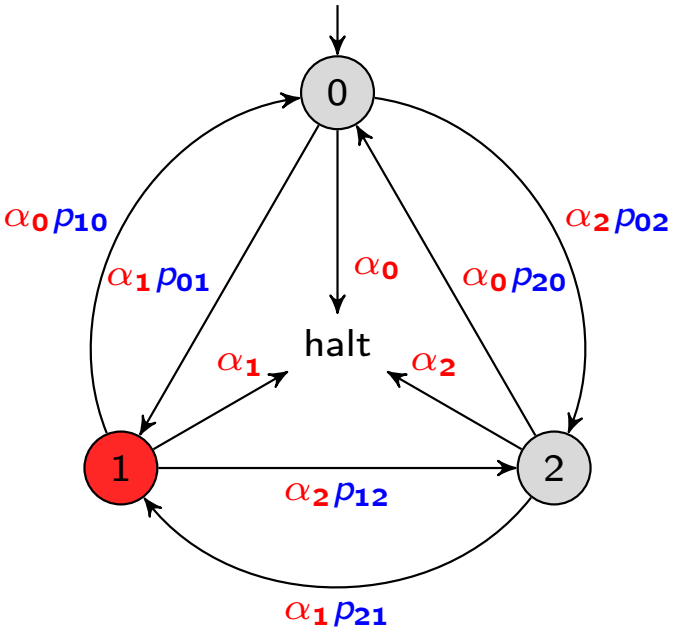


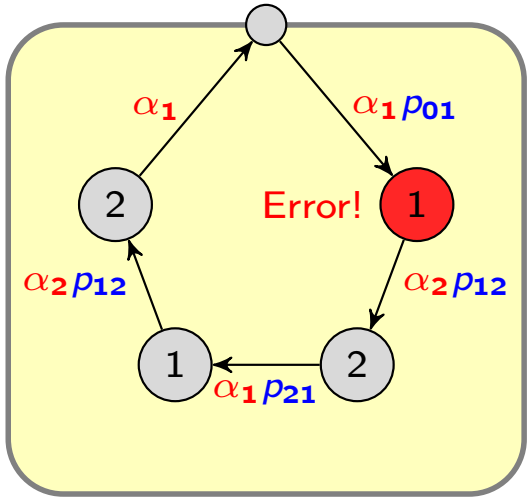
A 3-State SDA not Equivalent to Any While Program



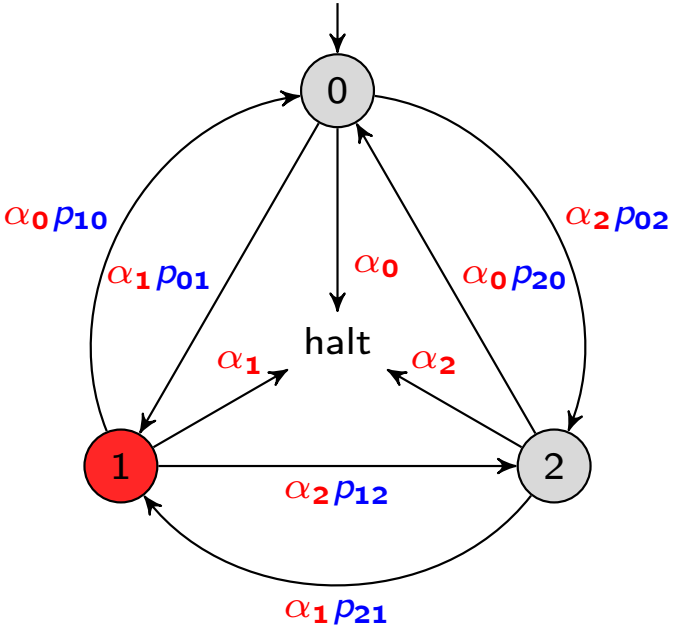


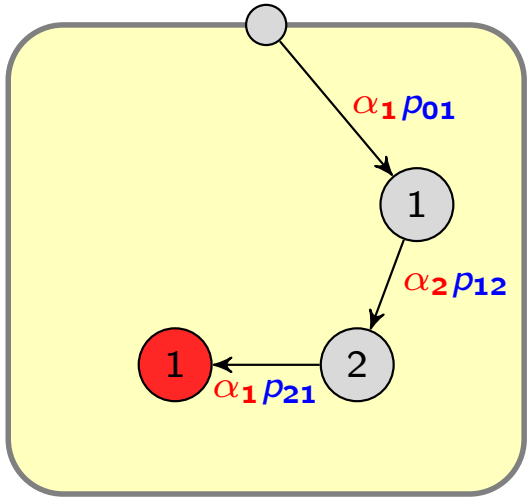
A 3-State SDA not Equivalent to Any While Program



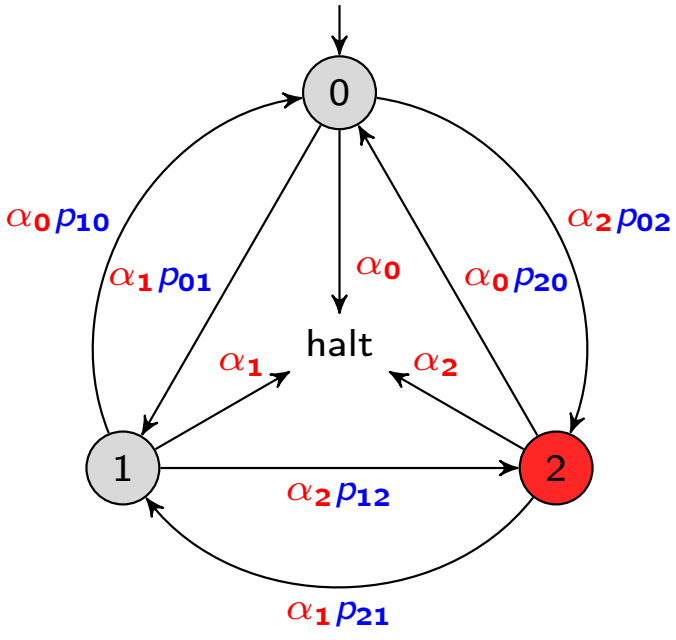


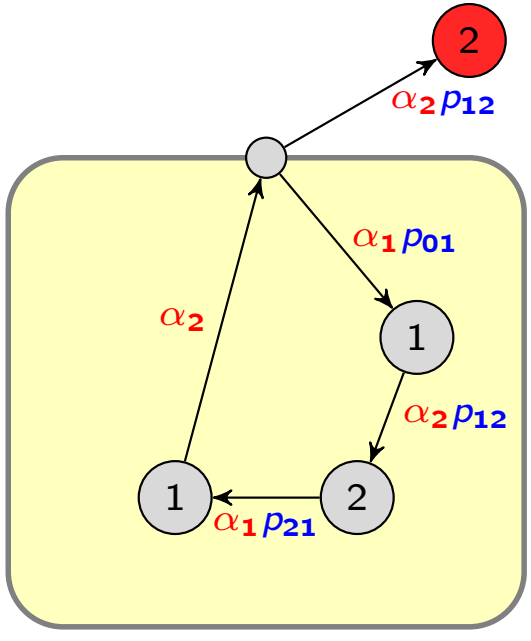
A 3-State SDA not Equivalent to Any While Program



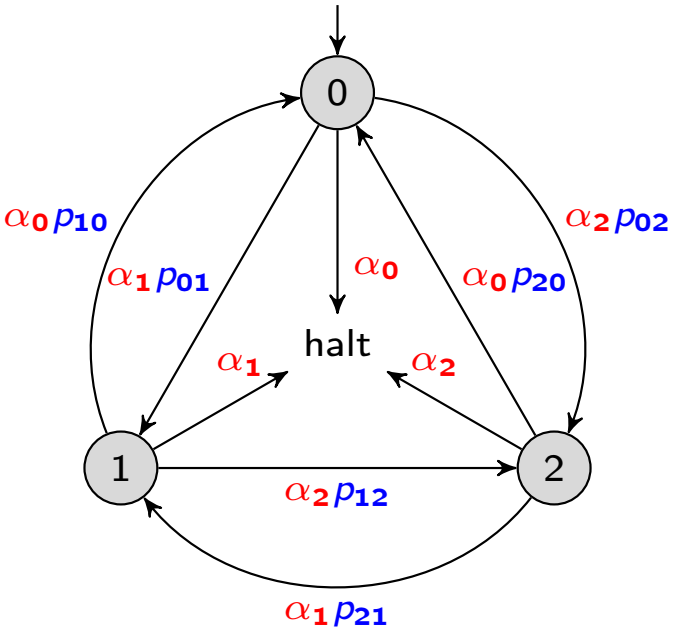


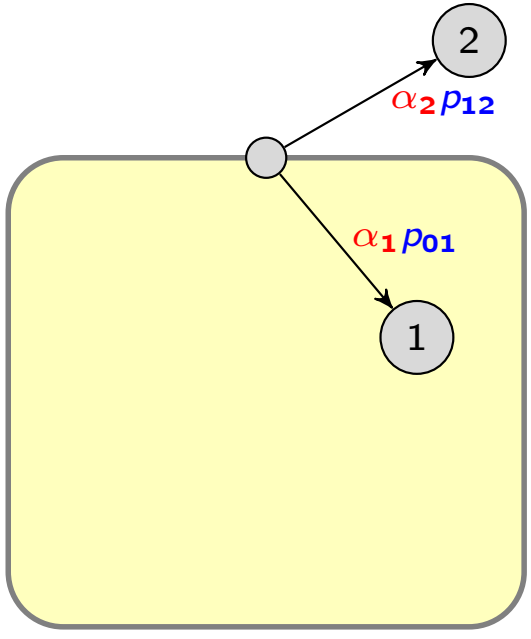
A 3-State SDA not Equivalent to Any While Program



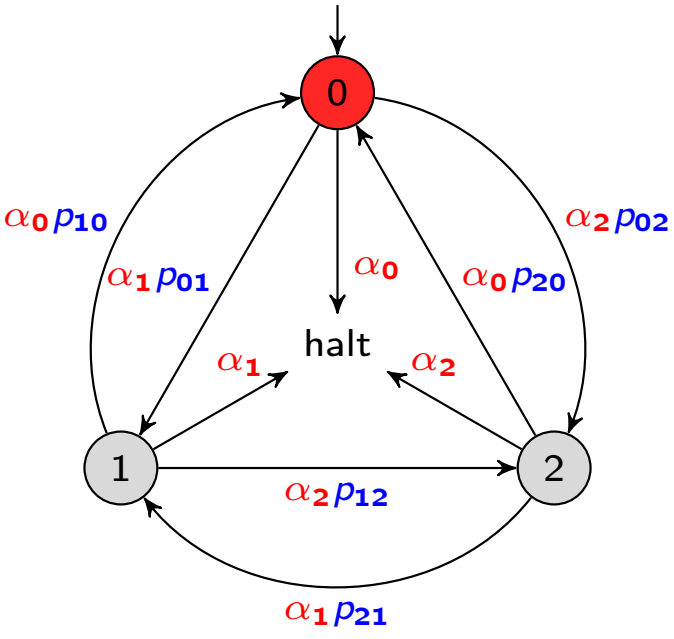


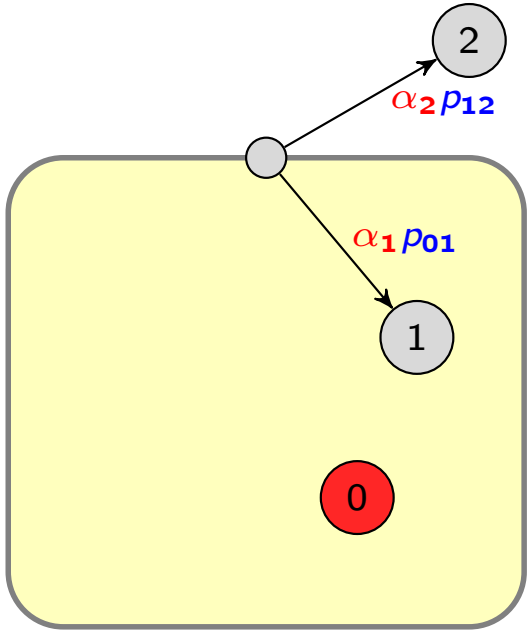
A 3-State SDA not Equivalent to Any While Program



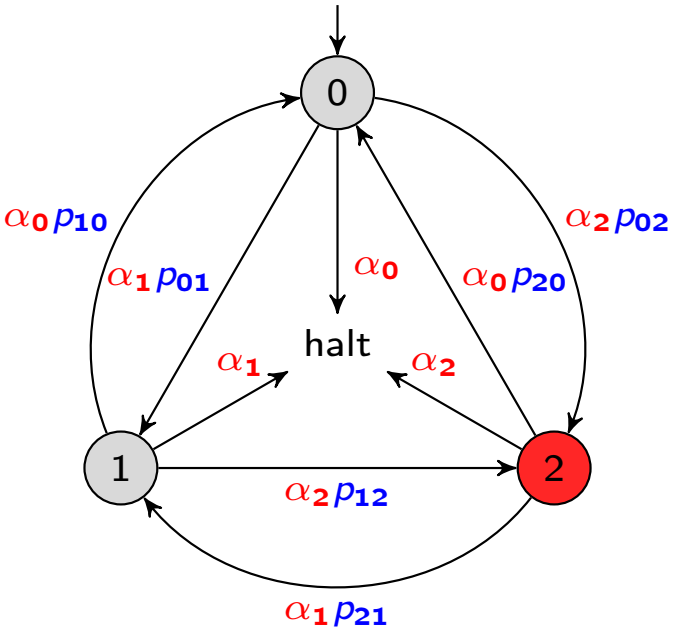


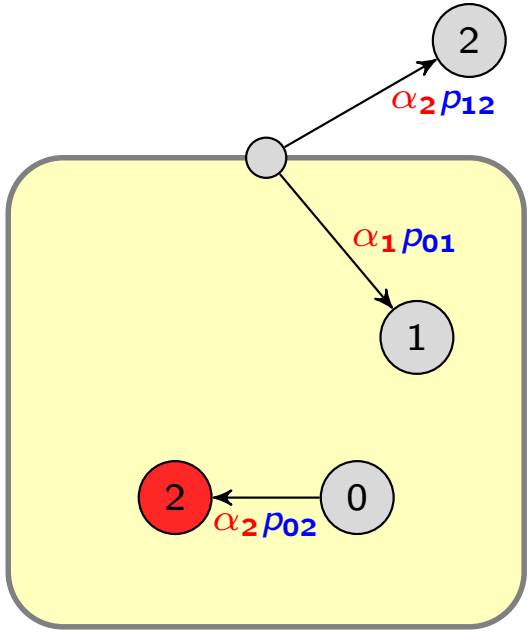
A 3-State SDA not Equivalent to Any While Program



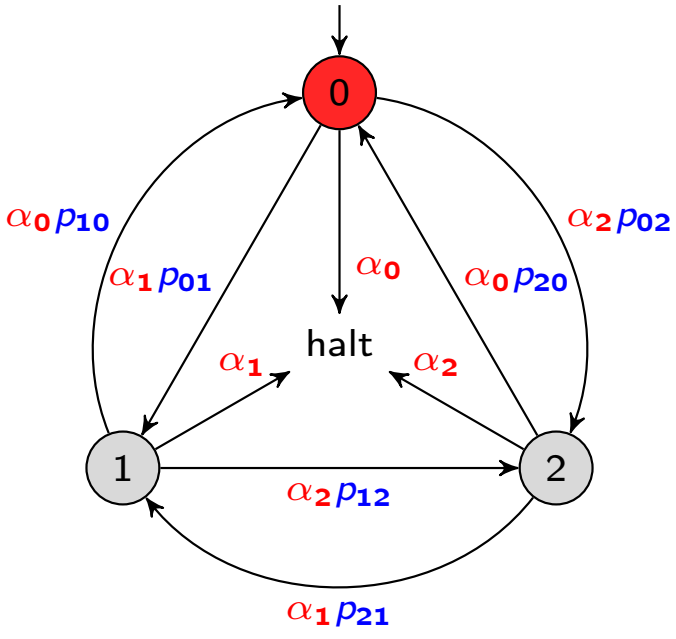


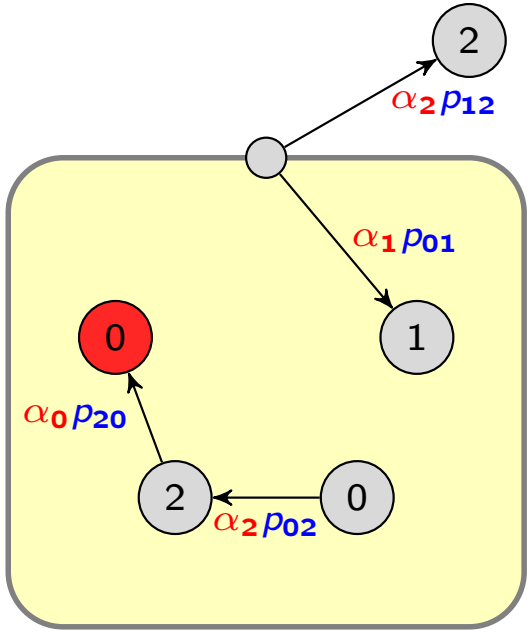
A 3-State SDA not Equivalent to Any While Program



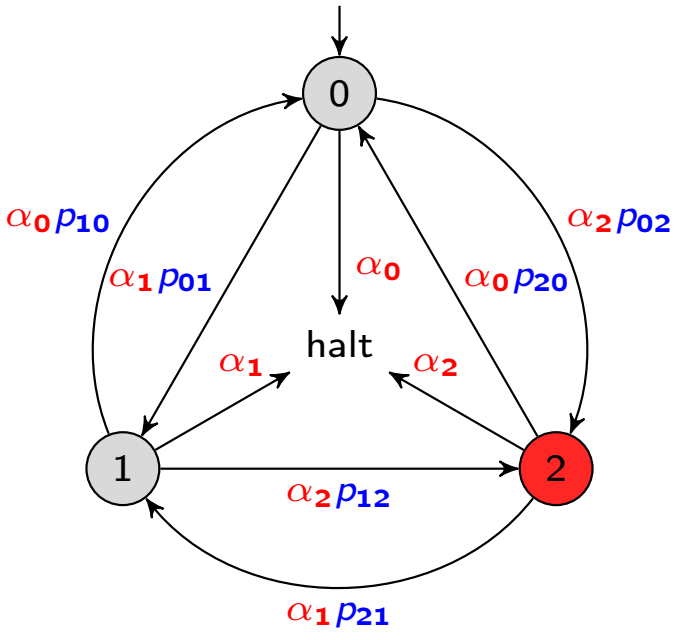


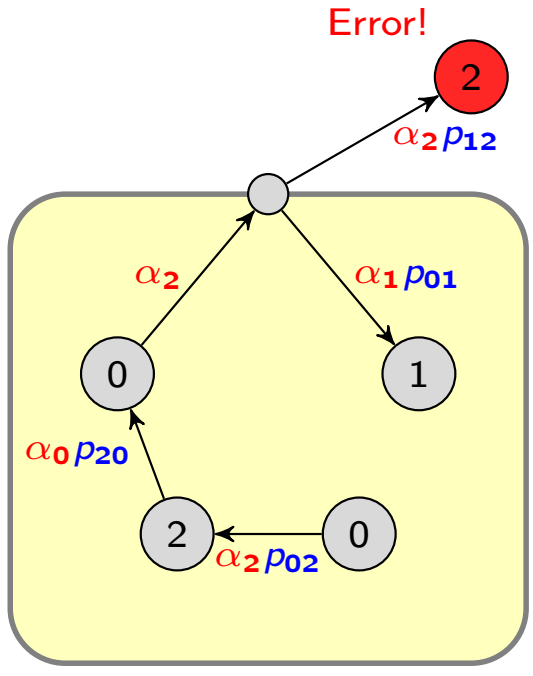
A 3-State SDA not Equivalent to Any While Program



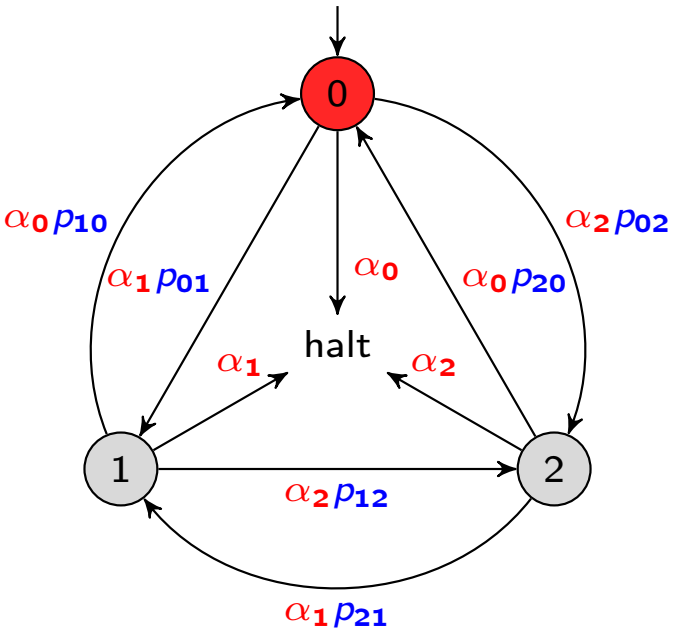


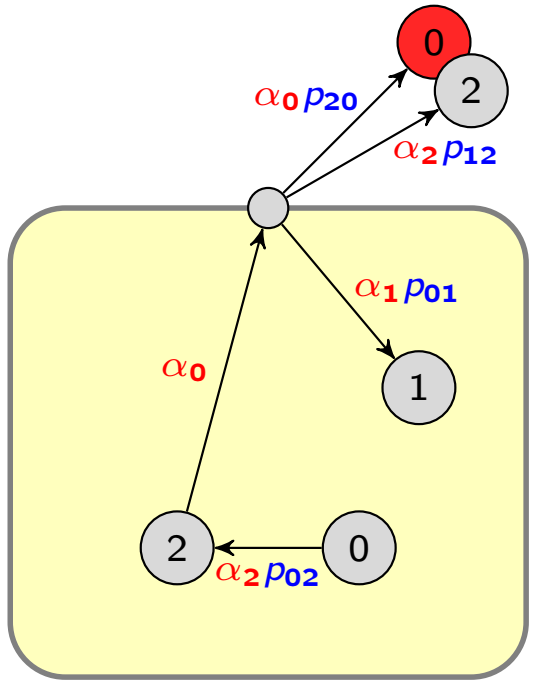
A 3-State SDA not Equivalent to Any While Program



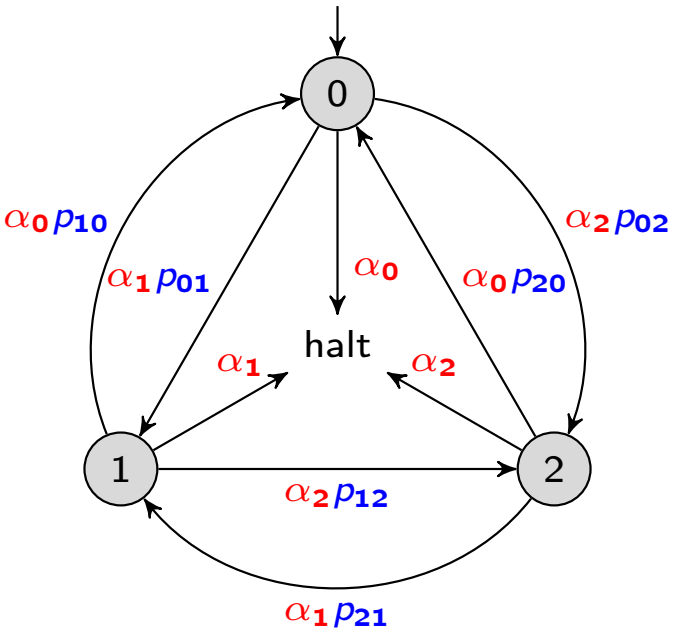


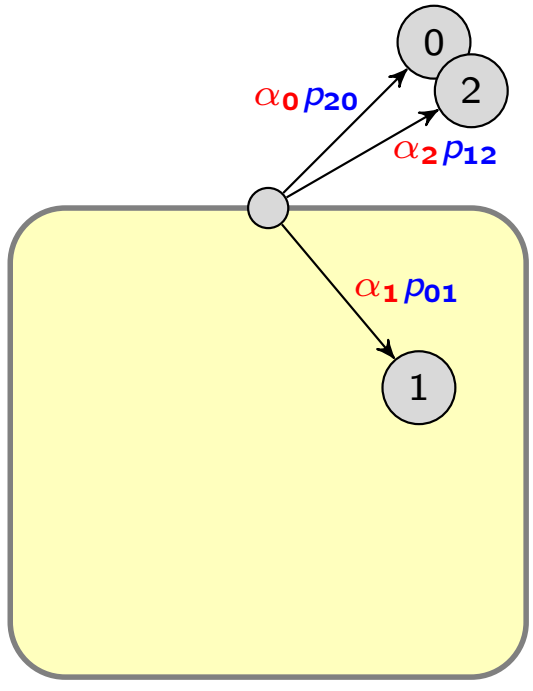
A 3-State SDA not Equivalent to Any While Program



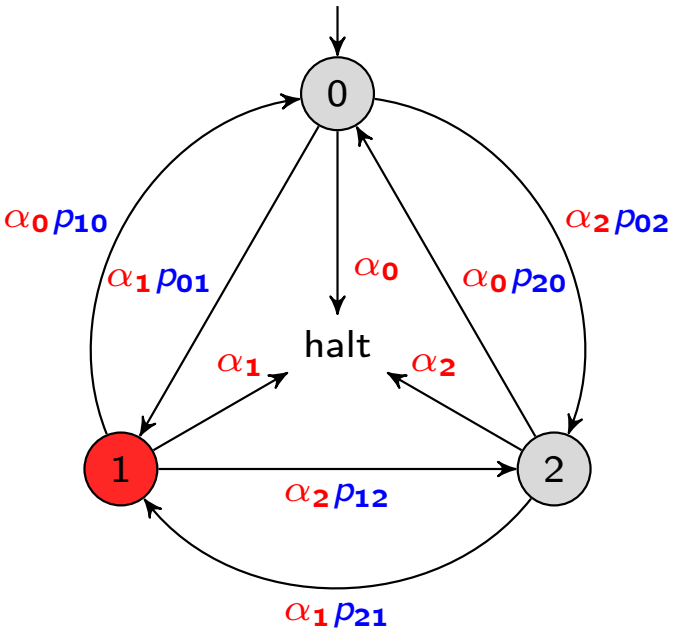


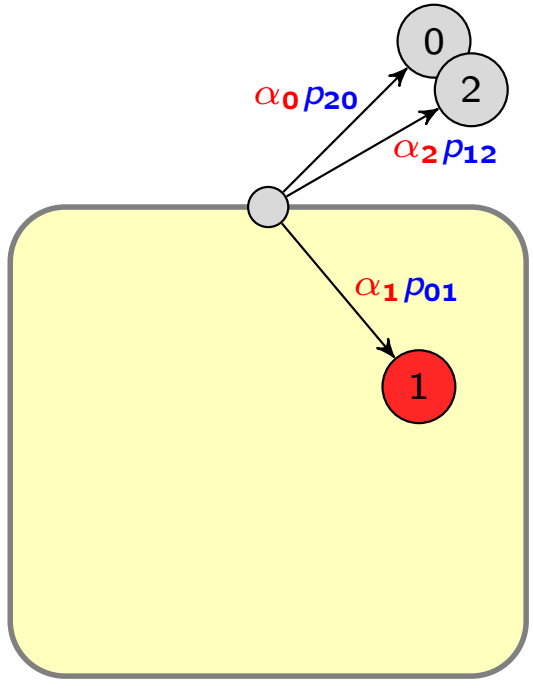
A 3-State SDA not Equivalent to Any While Program



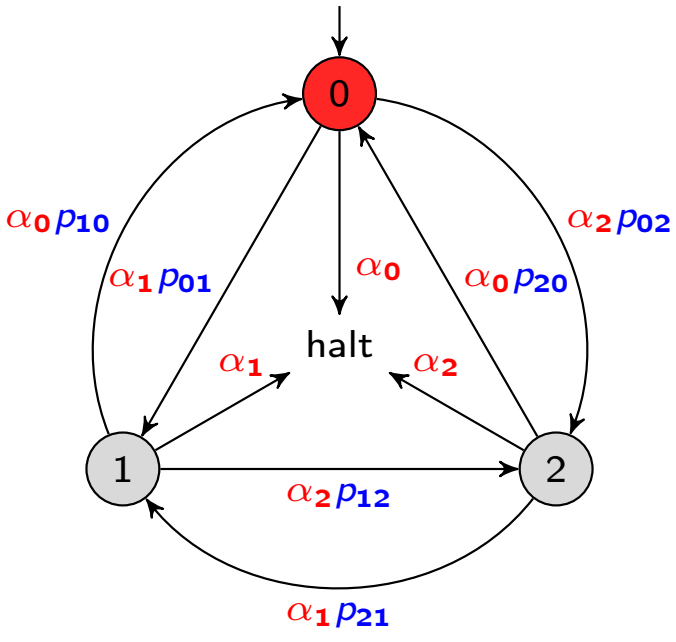


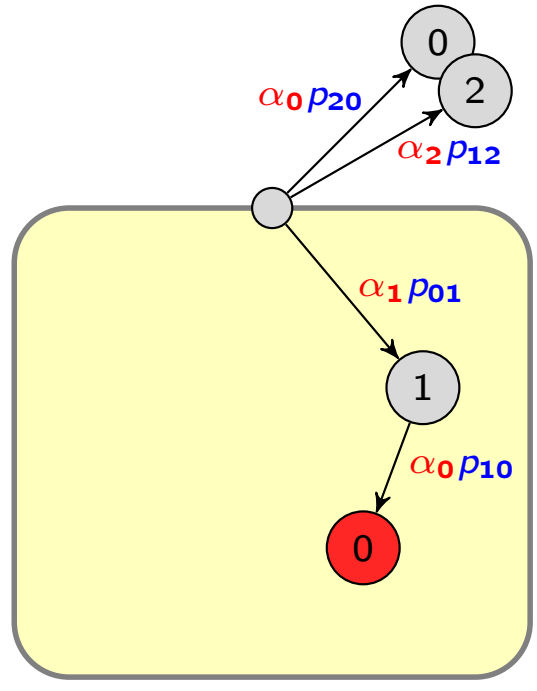
A 3-State SDA not Equivalent to Any While Program



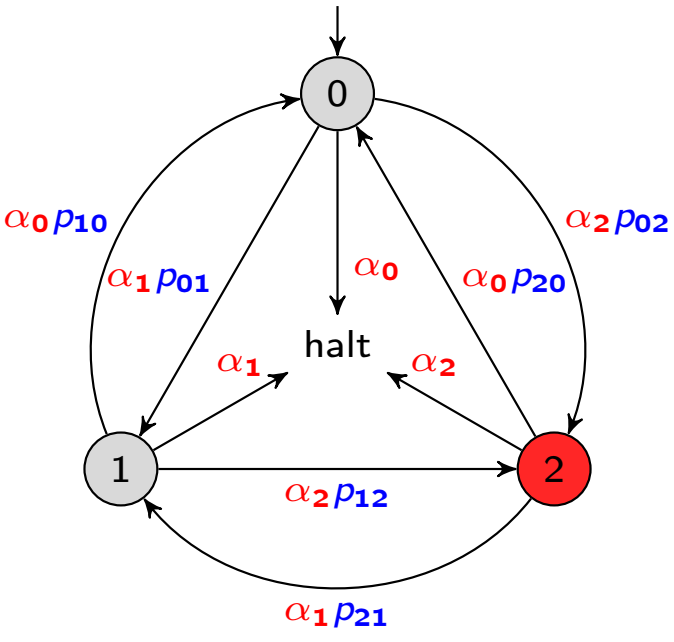


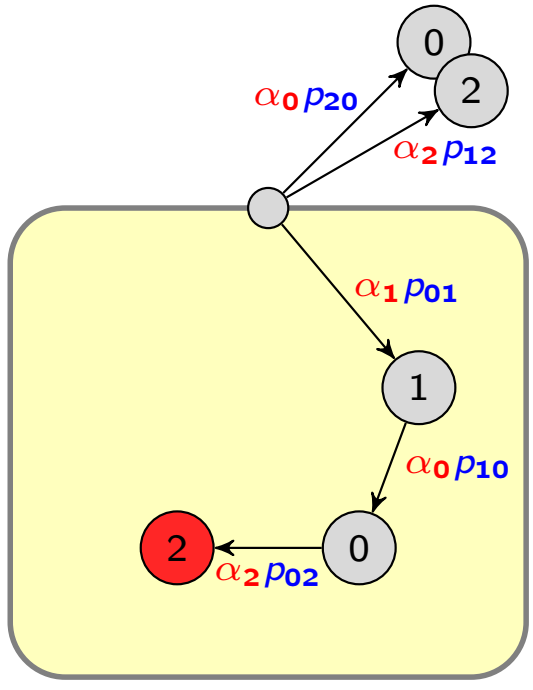
A 3-State SDA not Equivalent to Any While Program



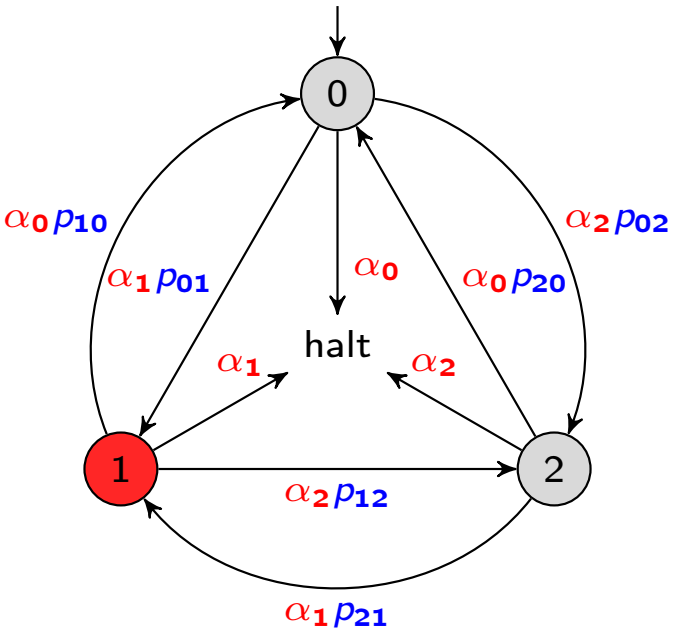


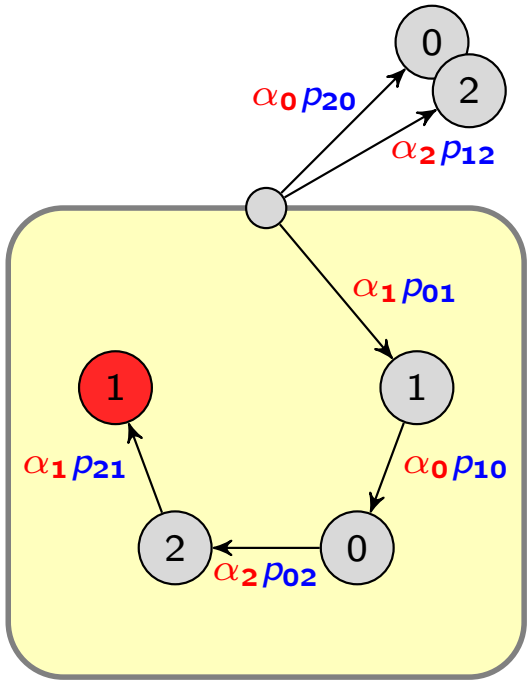
A 3-State SDA not Equivalent to Any While Program



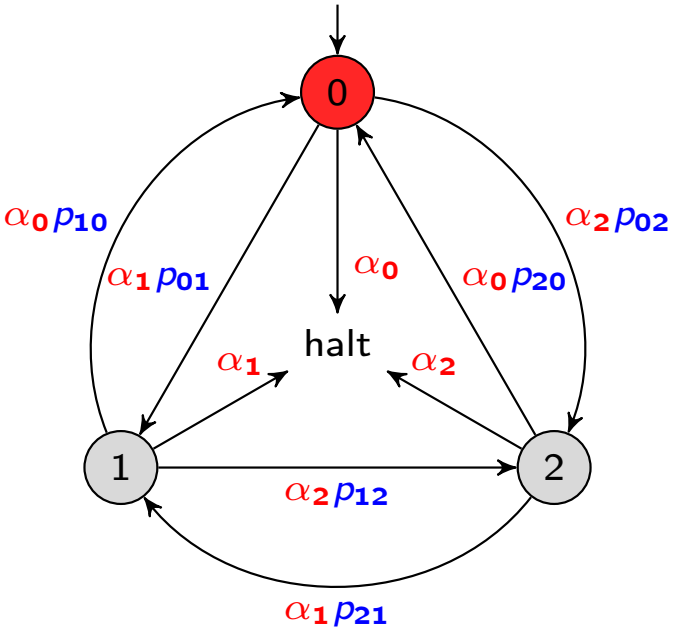


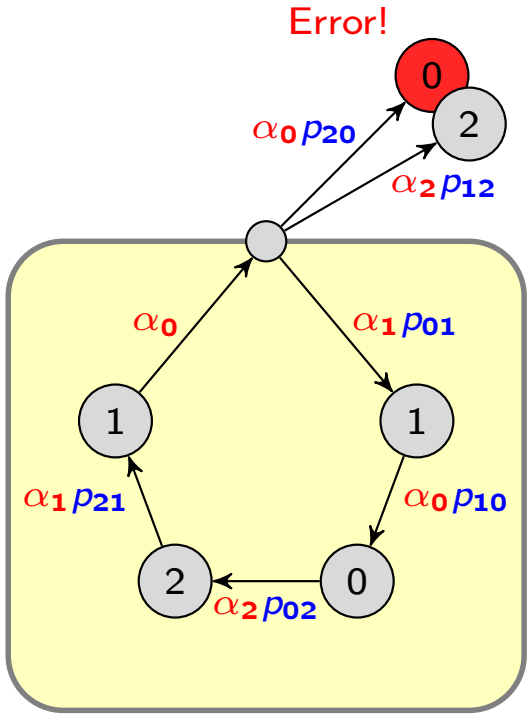
A 3-State SDA not Equivalent to Any While Program





A 3-State SDA not Equivalent to Any While Program

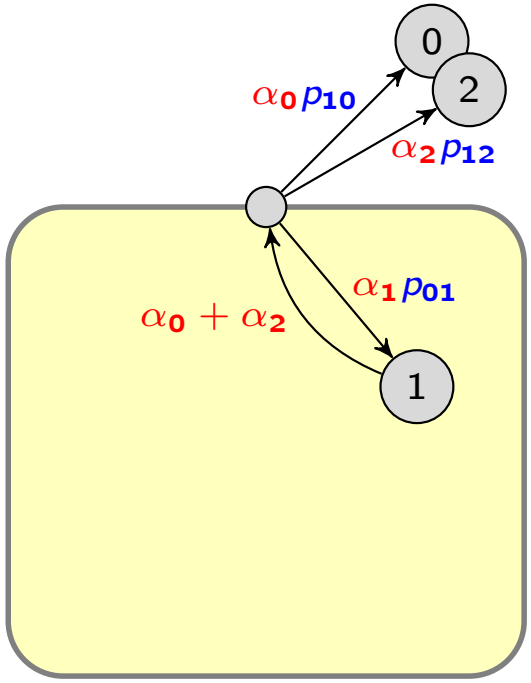




A 3-State SDA not Equivalent to Any While Program

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;
}
```



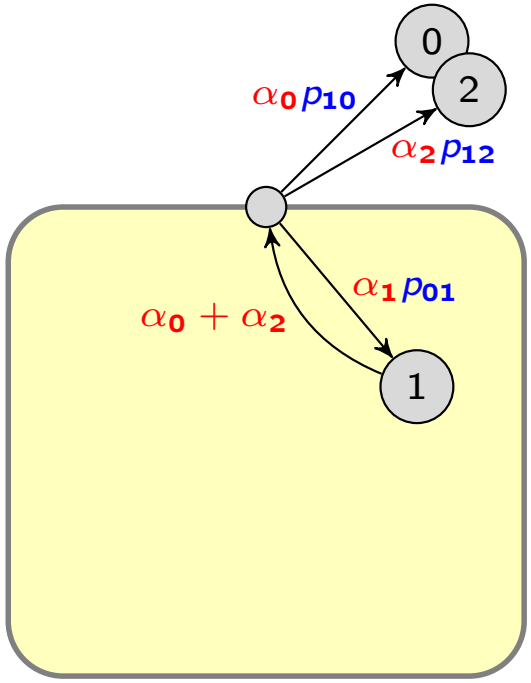
A 3-State SDA not Equivalent to Any While Program

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;
}
```

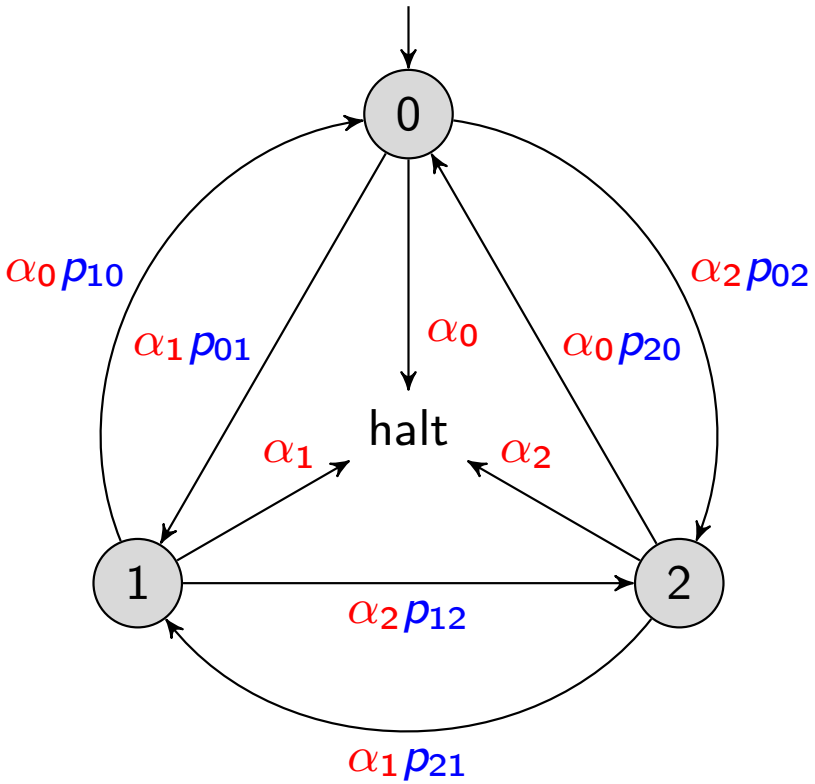


Loops Programs are Sufficient

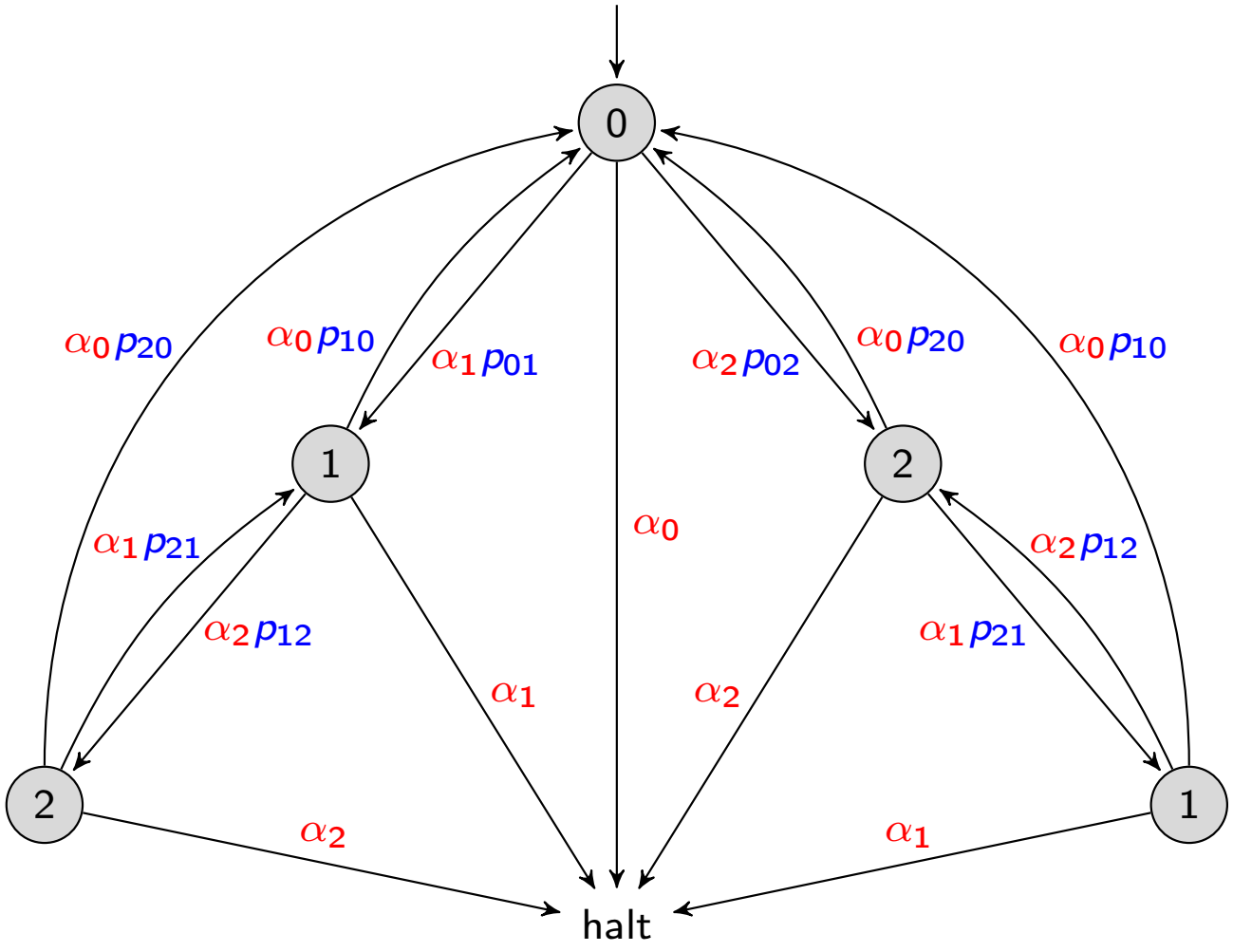
Theorem (Kosaraju 73)

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

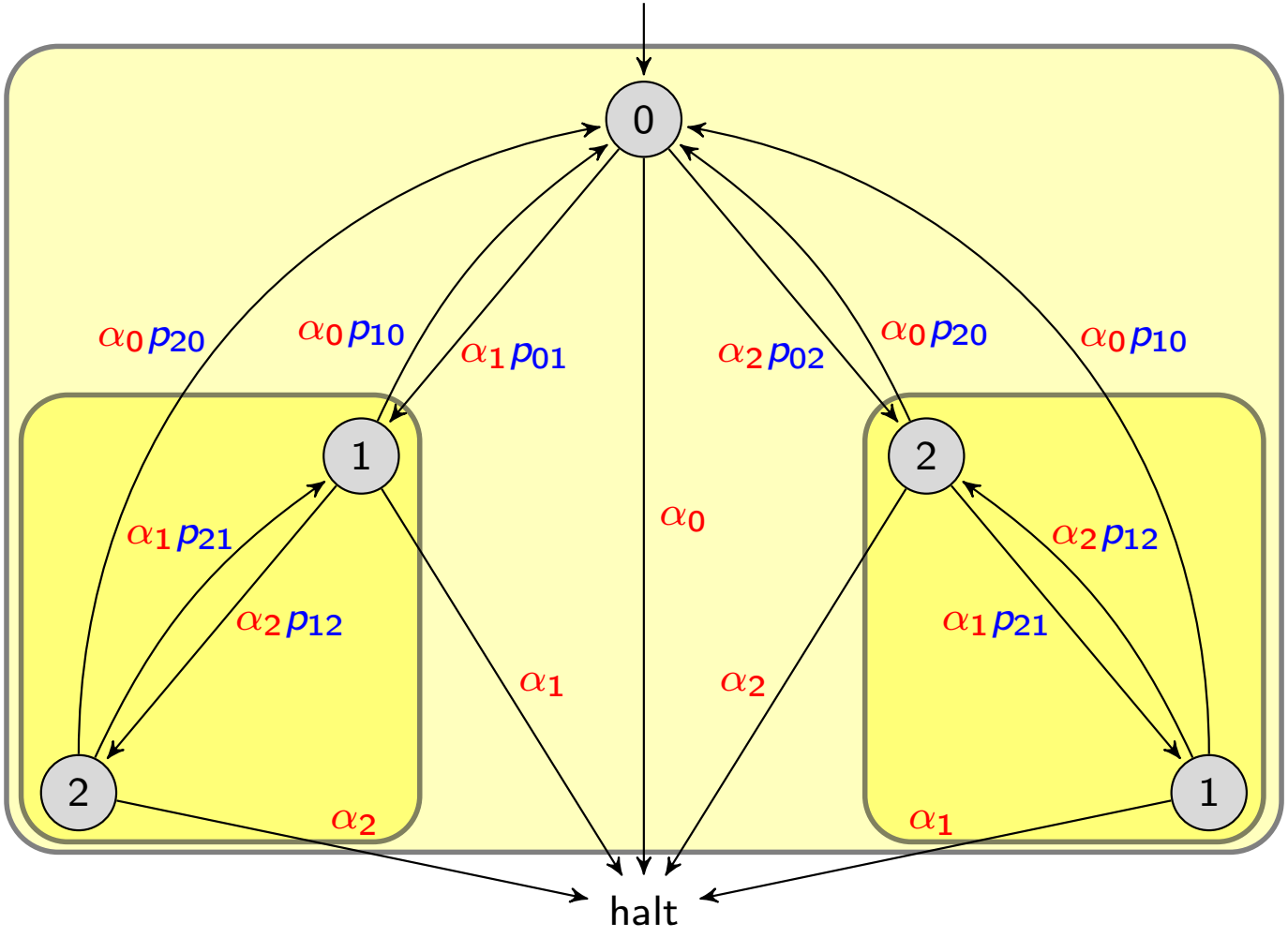
Example


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;
            }
        }
    }
}
```

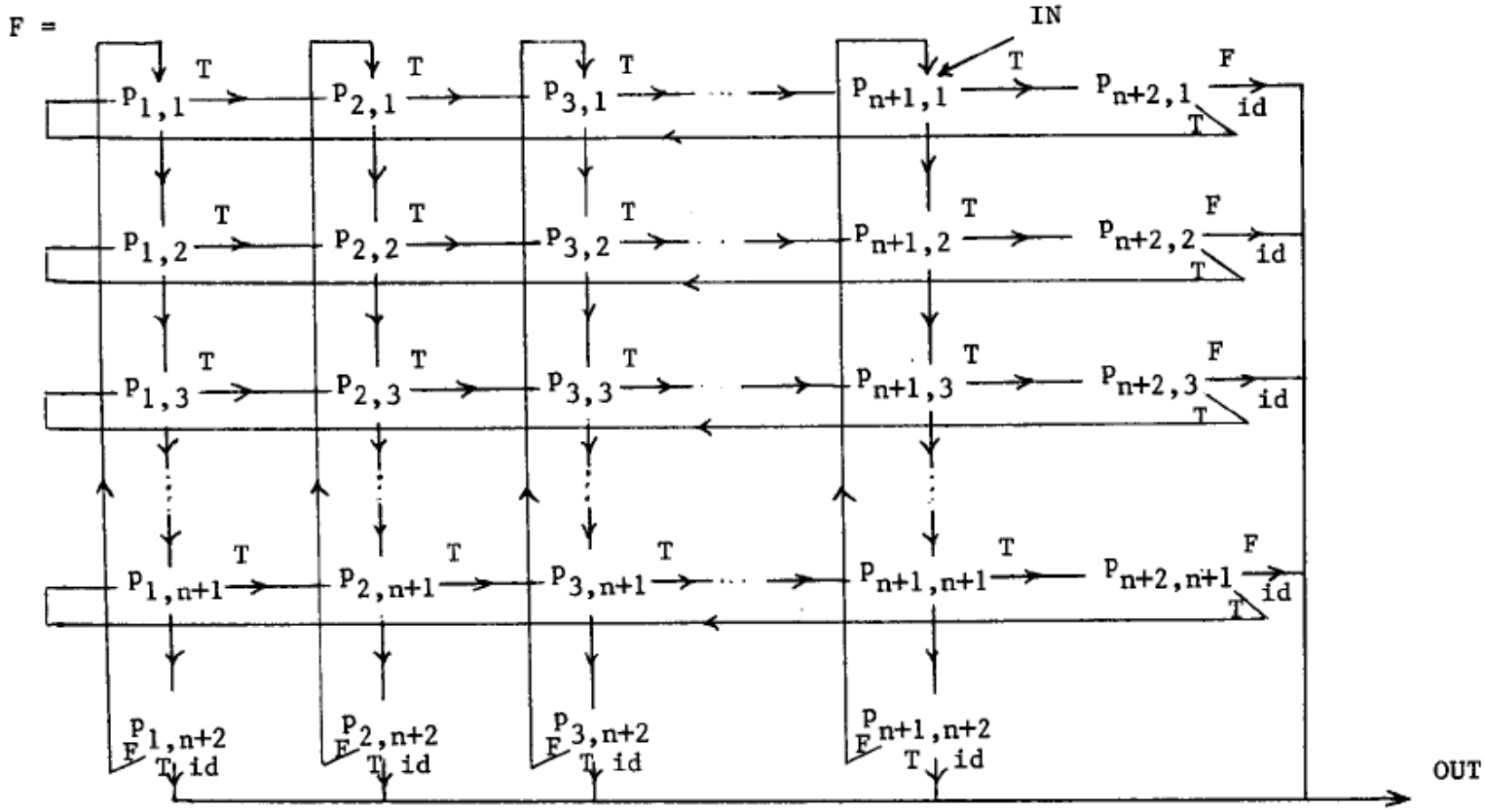
Equational Treatment of Nonlocal Control Flow

- 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
- Some novel technical features, including a treatment of multi-level breaks reminiscent of de Bruijn indices in the variable-free λ-calculus
- A purely calculational proof that every deterministic flowchart is equivalent to a loop/break program
- Completely rigorous and very amenable to automation

The Kosaraju Hierarchy Theorem

Theorem (Kosaraju 73)

There is a strict hierarchy of loop programs determined by the depth of nesting of loop instructions.



For the connections shown above: Function connecting $p_{i,j}$ and $p_{i',j'} = a_{i,j,i',j'}$

Figure 1

—from [Kosaraju 73]

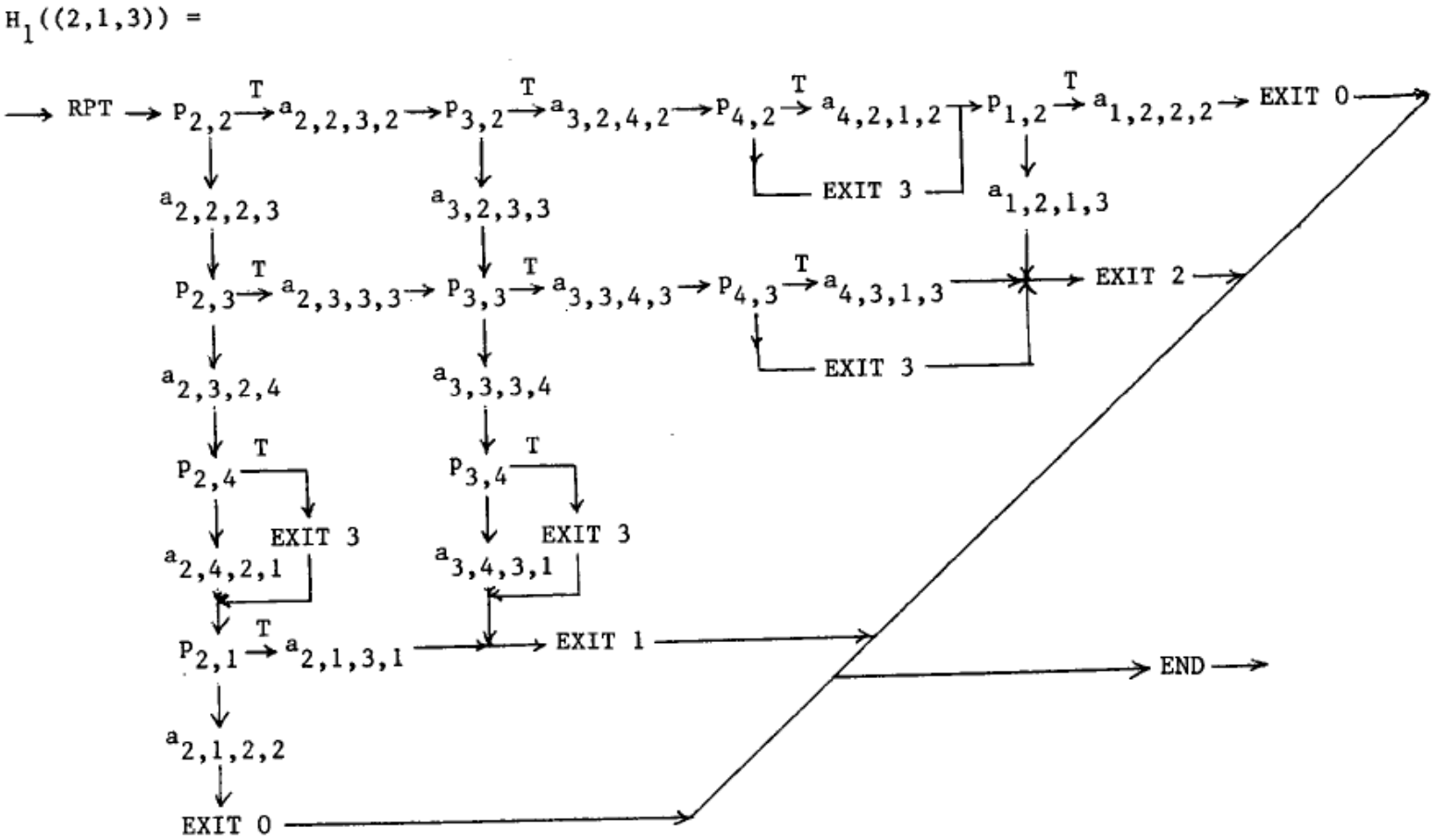


Figure 2

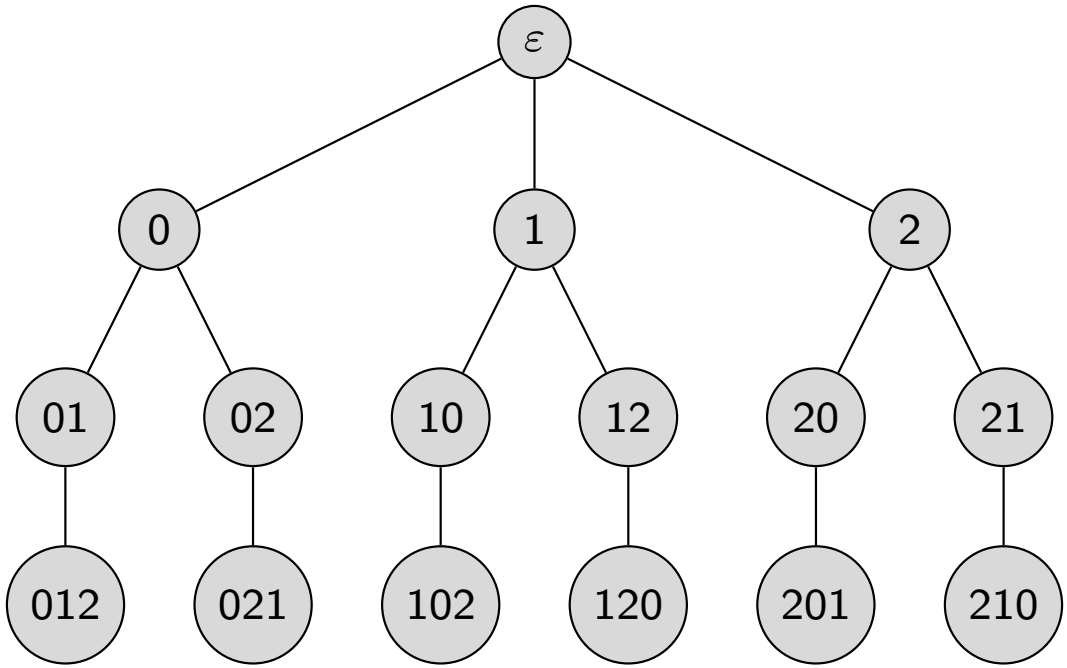
—from [Kosaraju 73]

An Alternative Construction

Construct an automaton P_n:

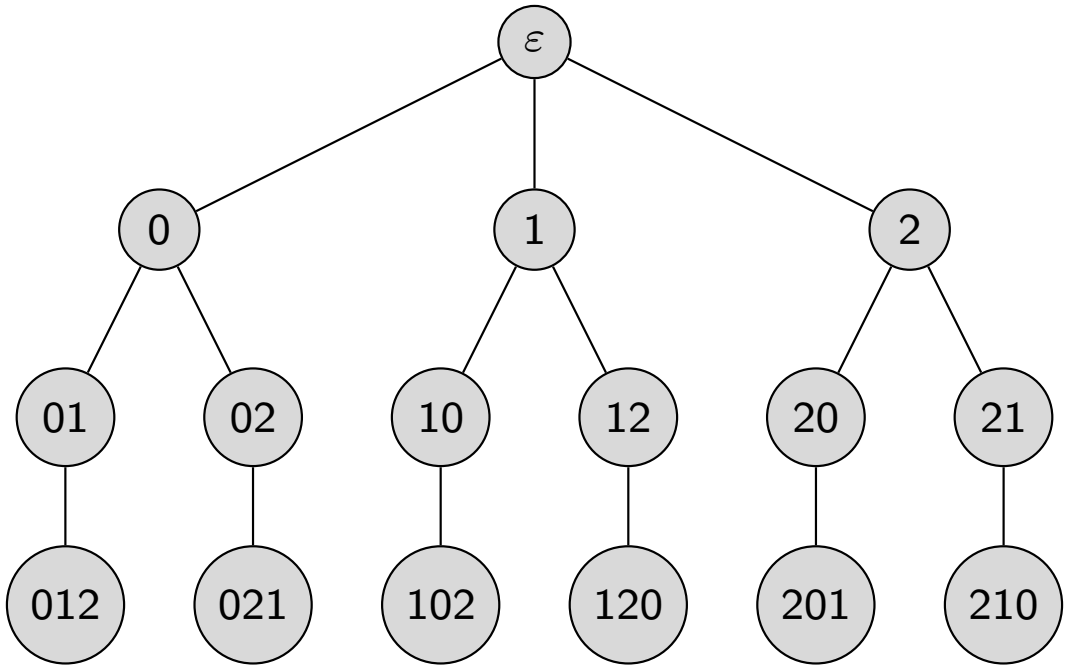
- states = all strings over $\{0, 1, \ldots, n-1\}$ with no repeated letters, including ε (there are roughly $n!e$ states)
- transitions $0, 1, \ldots, n-1$ and E
- transition i appends i to the current string if it does not already occur, or else truncates back to the prefix ending in i if it does occur
- E erases the string

Example: P_3



Q: What loop depth is necessary and sufficient to implement P_n?

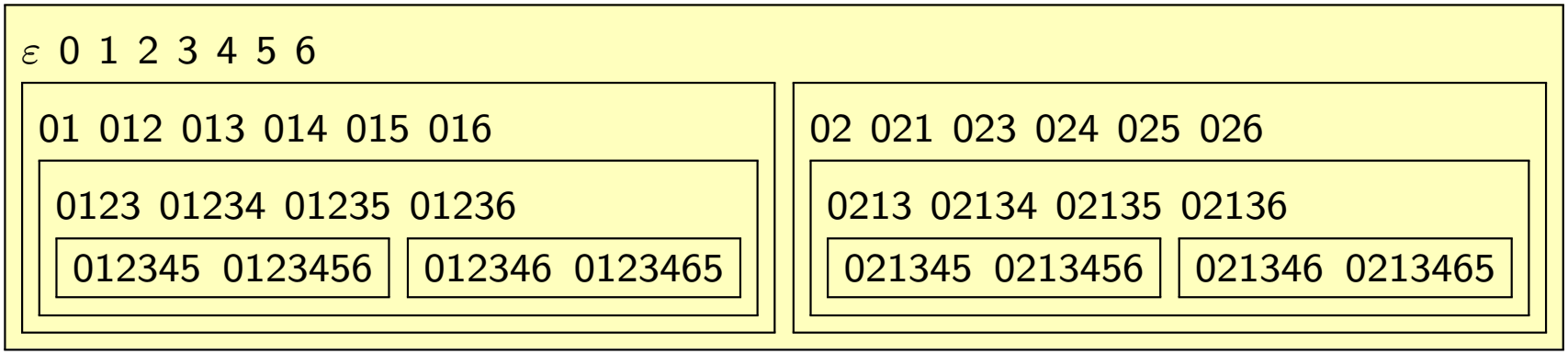
Example: P_3



Q: What loop depth is necessary and sufficient to implement P_n?

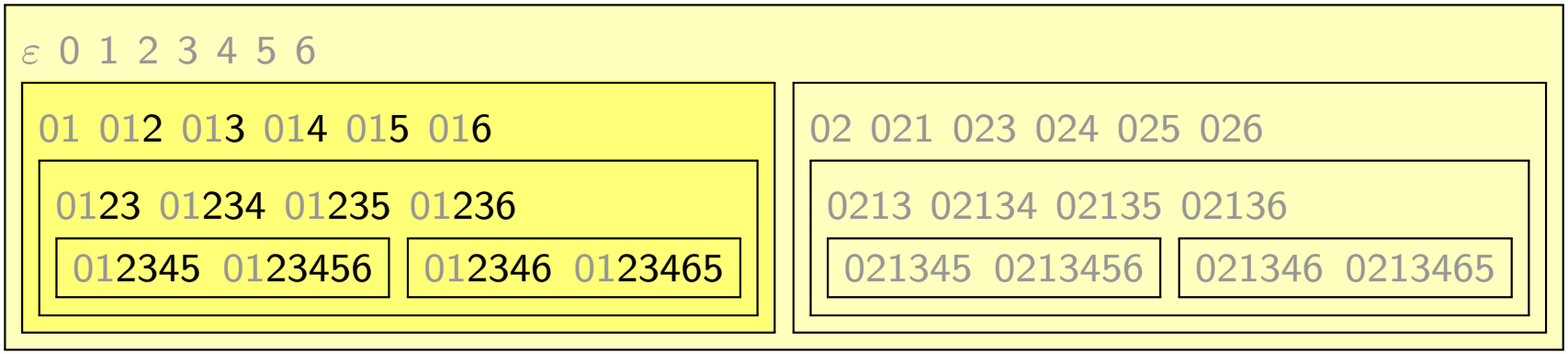
A: $\lfloor n/2 + 1 \rfloor$

Example: A depth-4 implementation of P_7



- only four paths of the nested loop program shown
  - one top-level loop
  - $\binom{n}{2}2!$ second-level loops
  - $\binom{n}{4}4!$ third-level loops within each second-level loop
  - etc.

Example: A depth-4 implementation of P_7



- only four paths of the nested loop program shown
  - one top-level loop
  - $\binom{n}{2}2!$ second-level loops
  - $\binom{n}{4}4!$ third-level loops within each second-level loop
  - etc.

Automata as Coalgebras

What is a Coalgebra?

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 $\approx$ image in the 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)

Some Technical Shortcomings of (C & P 2003)

- Automata work on primitive test symbols, not atoms
  - Limited class of KAT expressions
  - Requires a typing system to keep track of which Booleans have been tested
  - Automata must satisfy a path independence condition
- No complexity analysis

Some Improvements

- A new version of the Brzozowski derivative
- Works for all KAT expressions
- No typing system, no path independence
- Can produce coinductive equivalence and inequivalence proofs automatically in PSPACE, matching Worthington's bound

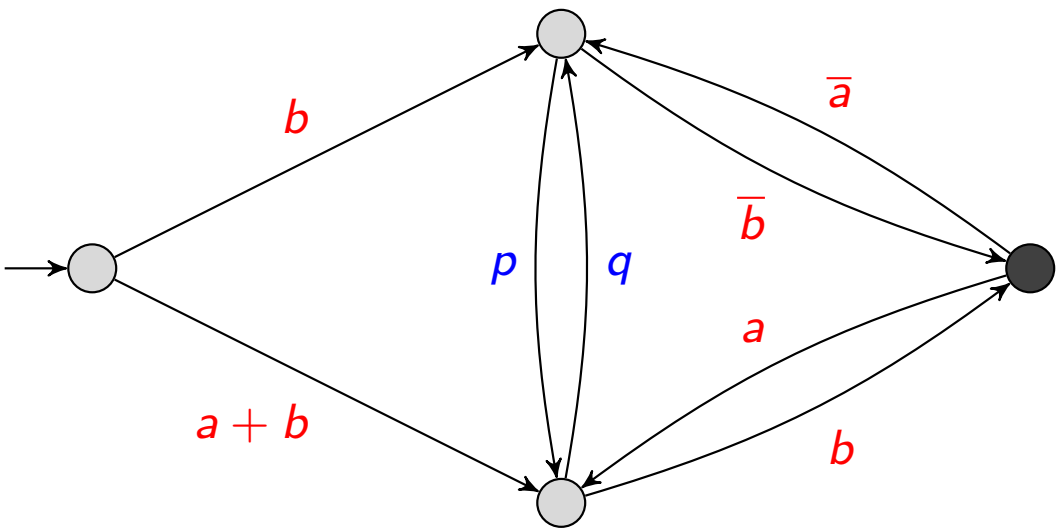
Automata on Guarded Strings

A Coalgebraic View

Nondeterministic automaton over Σ, T

$M = (Q, \Delta, \mathit{start}, \mathit{accept})$, where

$\Delta : (\Sigma + \mathrm{At}) \to Q \to 2^Q$ is a nondeterministic transition function



Automata on Guarded Strings

A Coalgebraic View

Acceptance is defined in terms of Kleisli composition $\bullet$ and Kleisli asterate $\dagger$ operations on maps $Q \to 2^Q$

$$(R \bullet S)(s) = \bigcup_{t \in S(s)} R(t)$$

$$R^0(s) = \{s\} \qquad R^{n+1} = R \bullet R^n \qquad R^\dagger = \bigcup_{n \geq 0} R^n$$

Define $\widehat{\Delta} : \mathcal{G} \to Q \to 2^Q$ inductively:

$$\widehat{\Delta}_\alpha = \Delta_\alpha^\dagger \qquad \widehat{\Delta}_{\alpha p y} = \widehat{\Delta}_y \bullet \Delta_p \bullet \Delta_\alpha^\dagger$$

M accepts x iff $\exists s \in \mathit{start}\ \exists t \in \mathit{accept}\ t \in \widehat{\Delta}_x(s)$

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

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

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

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

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

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

$L(A)(x) = 1$ 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

Determinization

$$M = (Q, \Delta, \mathit{start}, \mathit{accept}) \Rightarrow N = (2^Q, \delta, \varepsilon, \mathit{start})$$

$$\varepsilon_\alpha(R) = \begin{cases} 1 & \text{if } \exists s \in R\ \exists t \in \mathit{accept}\ t \in \Delta^\dagger_\alpha(s) \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_{\alpha p}(R) = \bigcup_{s \in R} \Delta_p \bullet \Delta^\dagger_\alpha(s)$$

$$L(\mathit{start})(x) = 1 \text{ iff } \exists s \in \mathit{start}\ \exists t \in \mathit{accept}\ t \in \widehat{\Delta}_x(s)$$

Complexity

Define expressions as labeled trees = partial functions $e : \omega^* \to \{+, \cdot, {}^*, 0, 1\}$ such that

- $\mathrm{dom}\ e$ nonempty and prefix-closed
- if $\sigma \in \mathrm{dom}\ e$, then $\sigma i \in \mathrm{dom}\ e \Leftrightarrow i \leq \mathrm{arity}(e(\sigma))$

$$F(0\sigma, e + e') = F(\sigma, e) \qquad F(\varepsilon, e) = e$$
$$F(1\sigma, e + e') = F(\sigma, e') \qquad F(0\sigma, e) = F(\sigma, e)$$
$$F(0\sigma, ee') = F(\sigma, e) \cdot e' \qquad F(0\sigma, e^*) = F(\sigma, e) \cdot e^*$$
$$F(1\sigma, ee') = F(\sigma, e')$$

Complexity

Theorem

For all KAT *expressions* e, $\sigma \in \mathrm{dom}\ e$, *and* $x \in \mathcal{G}$, *there exist* $a \in \{0, 1\}$ *and* $A \subseteq \mathrm{dom}\ e$ *such that*

$$D_x(F(\sigma, e)) \ \approx \ a + \sum_{\tau \in A} F(\tau, e),$$

where $\approx$ *denotes equivalence modulo the axioms of idempotent semigroups for* $+, 0$ *and the axioms* $0x = 0$, $1x = x$, *and* $(x + y)z = xz + yz$.

$\{D_x(e) \mid x \in \mathcal{G}\}$ is no larger than $|e| + 1$

Complexity

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, and inductively generate a bisimulation. □

Open Questions

- Duality?
- Context-free languages and derivatives of systems of polynomial inequalities?
- Premises?

Exercises

1. Consider infinite streams Σ^ω. Let $f_n : \Sigma^\omega \to \Sigma^\omega$ be the function "take multiples of n and drop the rest"; e.g., $f_3(a_0 a_1 a_2 \cdots) = a_0 a_3 a_6 \cdots$. Define the functions f_n coinductively. Prove by bisimulation that $f_3 \circ f_2 = f_2 \circ f_3$.
2. Let split : $\Sigma^\omega \to \Sigma^\omega \times \Sigma^\omega$ and merge : $\Sigma^\omega \times \Sigma^\omega \to \Sigma^\omega$ be the functions

$$\begin{aligned} \text{split}(a_0 a_1 a_2 \cdots) &= (a_0 a_2 a_4 \cdots, a_1 a_3 a_5 \cdots) \\ \text{merge}(a_0 a_1 a_2 \cdots, b_0 b_1 b_2 \cdots) &= a_0 b_0 a_1 b_1 \cdots . \end{aligned}$$

   Prove by bisimulation that split and merge are inverses.