# On the Complexity of the Equivalence Problem for Probabilistic Automata

Stefan Kiefer[1] Andrzej S. Murawski[2] Joël Ouaknine[1]
Björn Wachter[1] James Worrell[1]

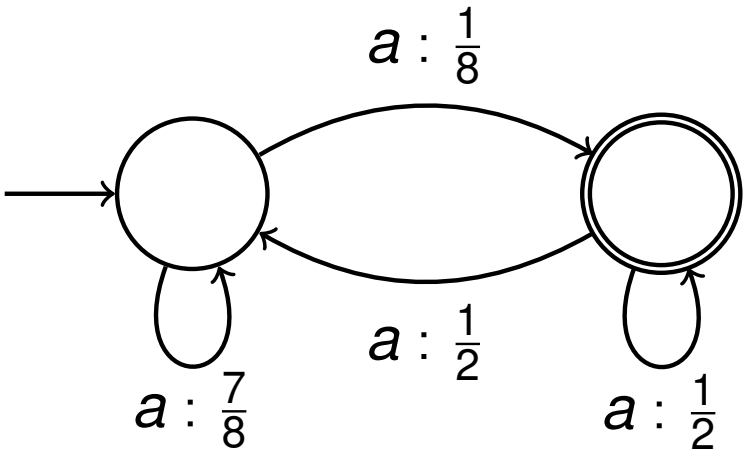
[1]University of Oxford, UK

[2]University of Leicester, UK

FoSSaCS, Tallinn
30 March 2012

# A Probabilistic Automaton

A probabilistic automaton $\mathcal{A}$

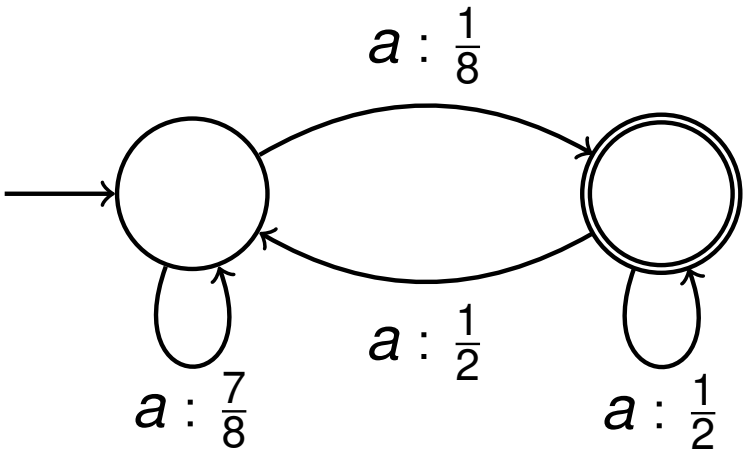



induces a language $\mathcal{A} : \Sigma^* \to [0, 1]$.

E.g.: $\mathcal{A}(aa) = \frac{7}{8} \cdot \frac{1}{8} + \frac{1}{8} \cdot \frac{1}{2}$

# A Probabilistic Automaton

A probabilistic automaton $\mathcal{A}$



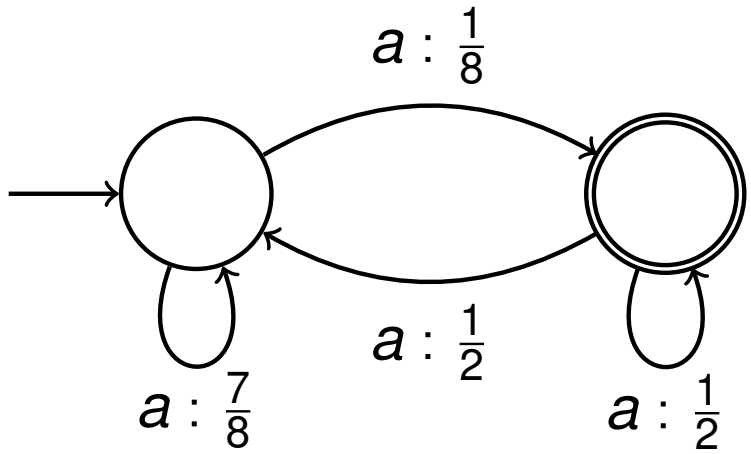

induces a language $\mathcal{A} : \Sigma^* \to [0, 1]$.

E.g.: $\mathcal{A}(aa) = \frac{7}{8} \cdot \frac{1}{8} + \frac{1}{8} \cdot \frac{1}{2}$

Alphabet $\Sigma$ may contain more than one symbol.

# A Probabilistic Automaton

A probabilistic automaton $\mathcal{A}$




induces a language $\mathcal{A} : \Sigma^* \to [0, 1]$.

E.g.: $\mathcal{A}(aa) = \frac{7}{8} \cdot \frac{1}{8} + \frac{1}{8} \cdot \frac{1}{2}$

Alphabet $\Sigma$ may contain more than one symbol.

Equivalence Problem:

Given two automata, do they induce the same language?

# Motivation: the APEX Tool

The APEX tool solves verification problems:

1. APEX translates two probabilistic programs into two automata (using game semantics).
2. APEX checks the resulting automata for equivalence.

$\longrightarrow$ APEX compares two open probabilistic programs, i.e., checks equivalence under all environments

$\longrightarrow$ especially well suited for verifying anonymity properties

# The APEX Tool

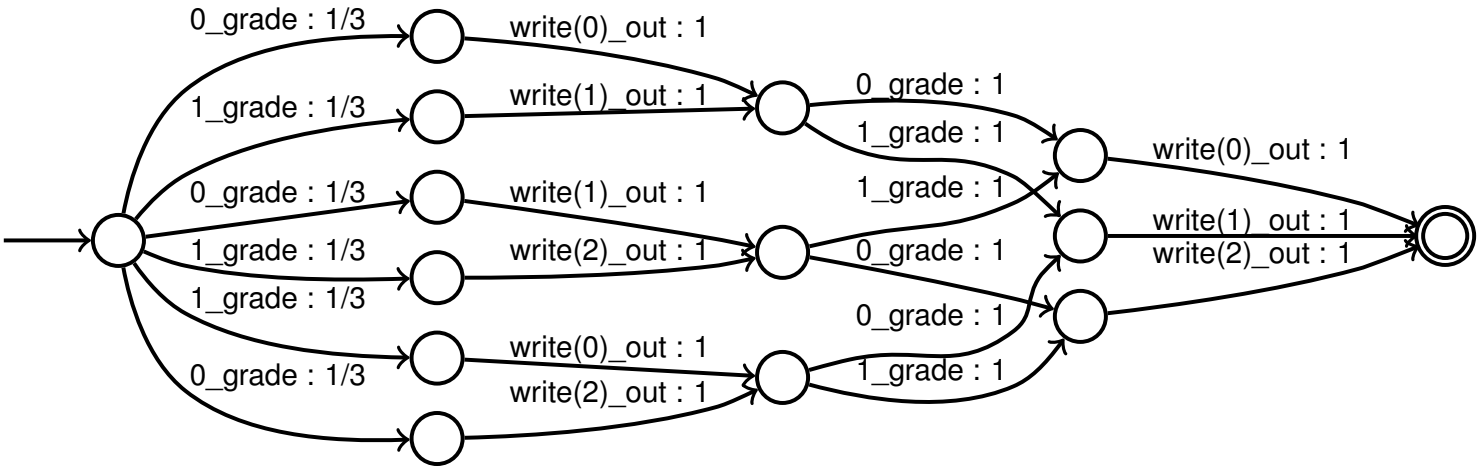
APEX input: two probabilistic programs

```
\\ Implementation

const N := S * (G-1) + 1;

grade:int%G, out:var%N |-
    var%(S+1) i; i := 0;
    var%N first; first := rand[N];
    var%N r; r := first;
    while(i<S) do {
      var%N l;
      i := succ(i);
      if(i=S) then
        left := first
      else
        left := rand[N];
        out := (grade + l) - r;
        r := l;
    }
```

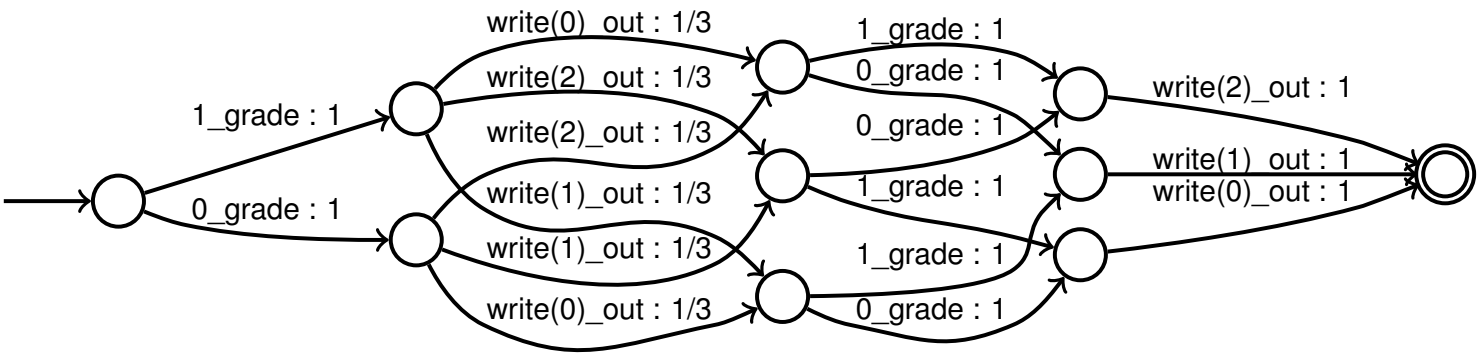
```
\\ Specification

const N := S * (G-1) + 1;

grade:int%G, out:var%N |-
    var%S i;
    var%N total;
    i := 1;

    while(i) do {
      total := grade + total;
      var%N r;
      r := rand[N];
      out := r;
      total := total - r;
      i := succ(i)
    };
    out := grade + total
```

# The APEX Tool

## Implementation:




## Specification:



# The Equivalence Problem is in P.

Good news:

**Theorem (Schützenberger'61, Tzeng'92)**

*Equivalence of probabilistic automata is decidable in polynomial time.*

In previous work (CAV'11) we gave a different algorithm:

- randomised
- based on polynomial identity testing (Schwartz-Zippel)
- more efficient on APEX examples

Theme of this paper:

Polynomial identity testing yields efficient algorithms for (generalisations of) the equivalence problem.

# The Equivalence Problem is in NC.

The equivalence problem is "efficiently parallelisable":

**Theorem (Tzeng'96)**

*Equivalence of probabilistic automata is in NC.*

Reminder:
NC = solvable in polylogarithmic parallel time
with polynomially many processors on a PRAM

$$\mathrm{NL} \subseteq \mathrm{NC} \subseteq \mathrm{RNC} \subseteq \mathrm{P}$$

NC contains e.g. solving linear equation systems.
RNC contains e.g. finding a maximum matching in a graph.

# The Equivalence Problem is in NC.

The equivalence problem is "efficiently parallelisable":

**Theorem (Tzeng'96)**

*Equivalence of probabilistic automata is in NC.*

Reminder:
NC = solvable in polylogarithmic parallel time
with polynomially many processors on a PRAM

$$\mathrm{NL} \subseteq \mathrm{NC} \subseteq \mathrm{RNC} \subseteq \mathrm{P}$$

NC contains e.g. solving linear equation systems.
RNC contains e.g. finding a maximum matching in a graph.

Tzeng's algorithm does not yield a counterexample.
counterexample = word $w$ with $\mathcal{A}_1(w) \neq \mathcal{A}_2(w)$

# A Counterexample Can be Computed in RNC.

**Theorem**

*There is an RNC procedure for the equivalence problem that outputs a counterexample if there is one.*

# A Counterexample Can be Computed in RNC.

**Theorem**

*There is an RNC procedure for the equivalence problem that outputs a counterexample if there is one.*

The proof uses 2 main ingredients:

- If there is a counterexample, then there is a short one (of length $< n_1 + n_2$).
- Isolating Lemma

# The Isolating Lemma

**Lemma (Isolating Lemma; Mulmuley,Vazirani,Vazirani '87)**

*Given a set* $\{e_1, \ldots, e_N\}$ *and a family of subsets.*

*For each* $e_i$ *choose a weight independently and uniformly at random from* $\{1, \ldots, 2N\}$.

*Define the weight of a subset straightforwardly.*

*Then with prob.* $\geq 0.5$ *there is a unique minimum weight set.*

$e_1$ $e_2$ $e_3$ $e_4$ $e_5$

# The Isolating Lemma

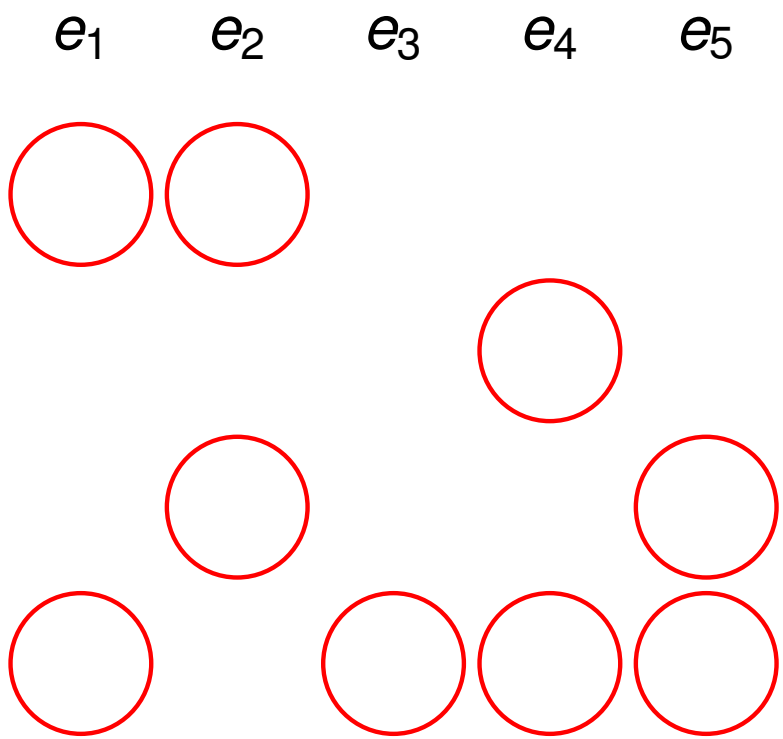
**Lemma (Isolating Lemma; Mulmuley,Vazirani,Vazirani '87)**

*Given a set* $\{e_1, \ldots, e_N\}$ *and a family of subsets.*

*For each* $e_i$ *choose a weight independently and uniformly at random from* $\{1, \ldots, 2N\}$.

*Define the weight of a subset straightforwardly.*

*Then with prob.* $\geq 0.5$ *there is a unique minimum weight set.*



# The Isolating Lemma

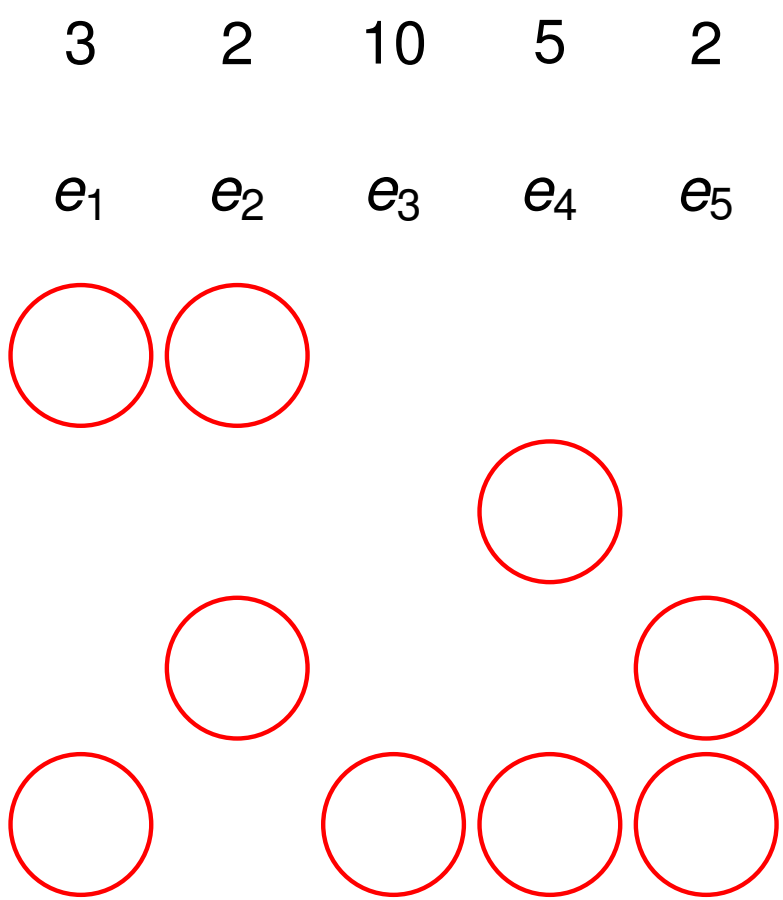
**Lemma (Isolating Lemma; Mulmuley,Vazirani,Vazirani '87)**

*Given a set* $\{e_1, \ldots, e_N\}$ *and a family of subsets.*

*For each* $e_i$ *choose a weight independently and uniformly at random from* $\{1, \ldots, 2N\}$.

*Define the weight of a subset straightforwardly.*

*Then with prob.* $\geq 0.5$ *there is a unique minimum weight set.*



# The Isolating Lemma

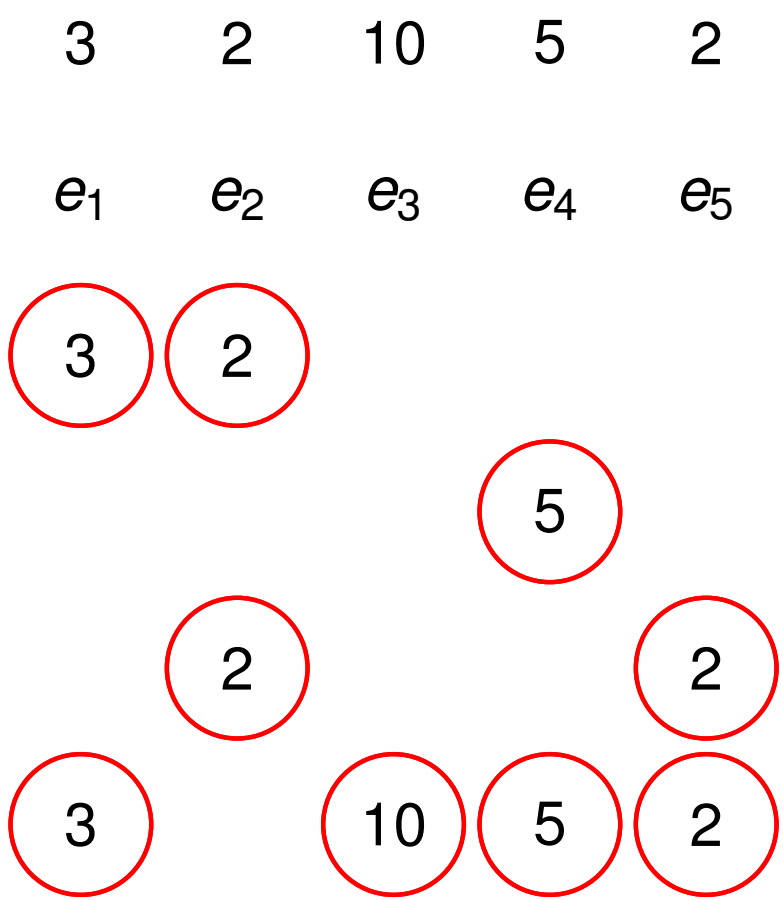
**Lemma (Isolating Lemma; Mulmuley,Vazirani,Vazirani '87)**

*Given a set $\{e_1, \ldots, e_N\}$ and a family of subsets.*

*For each $e_i$ choose a weight independently and uniformly at random from $\{1, \ldots, 2N\}$.*

*Define the weight of a subset straightforwardly.*

*Then with prob. $\geq 0.5$ there is a unique minimum weight set.*



# The Isolating Lemma

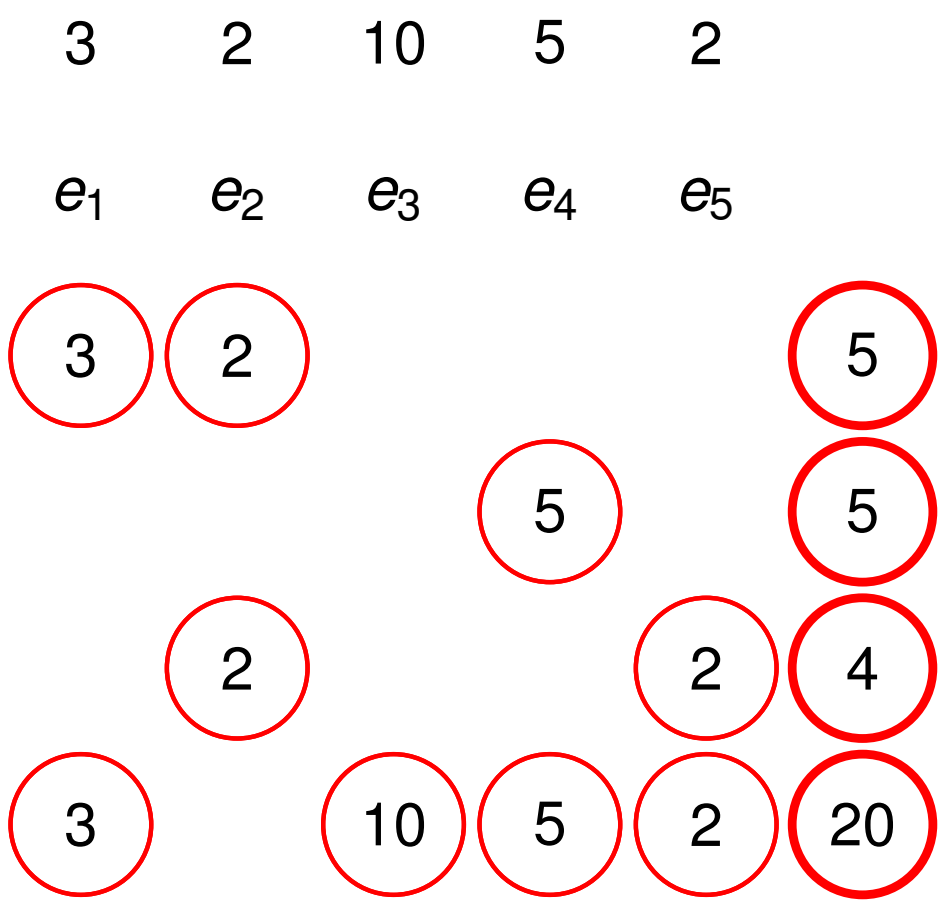
**Lemma (Isolating Lemma; Mulmuley,Vazirani,Vazirani '87)**

*Given a set $\{e_1, \ldots, e_N\}$ and a family of subsets.*

*For each $e_i$ choose a weight independently and uniformly at random from $\{1, \ldots, 2N\}$.*

*Define the weight of a subset straightforwardly.*

*Then with prob. $\geq 0.5$ there is a unique minimum weight set.*



# The Isolating Lemma

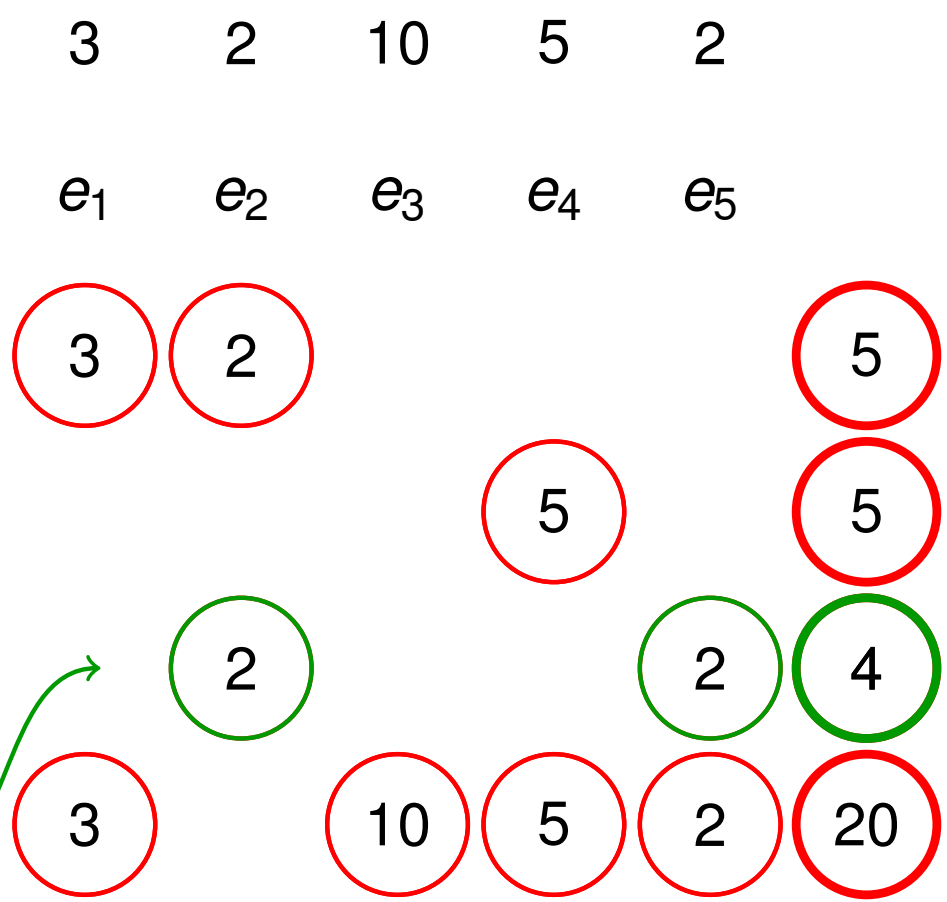
**Lemma (Isolating Lemma; Mulmuley,Vazirani,Vazirani '87)**

*Given a set $\{e_1, \ldots, e_N\}$ and a family of subsets.*

*For each $e_i$ choose a weight independently and uniformly at random from $\{1, \ldots, 2N\}$.*

*Define the weight of a subset straightforwardly.*

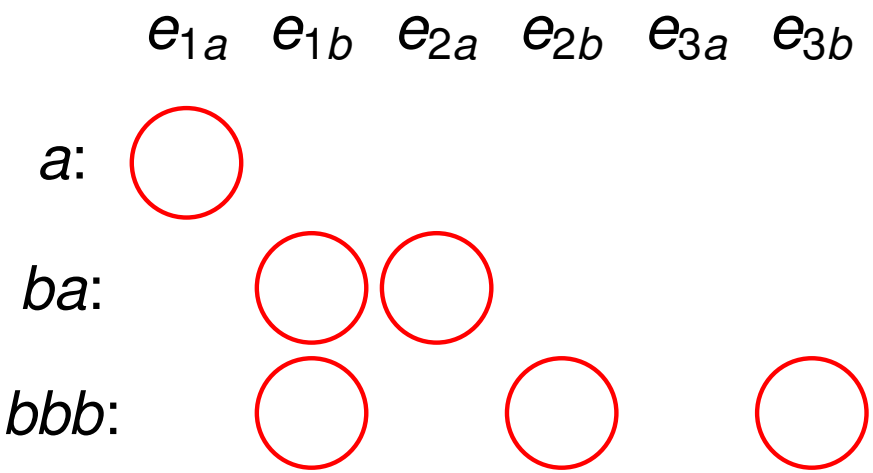
*Then with prob. $\geq 0.5$ there is a unique minimum weight set.*



# Proof Idea

Consider the short counterexamples as subsets.

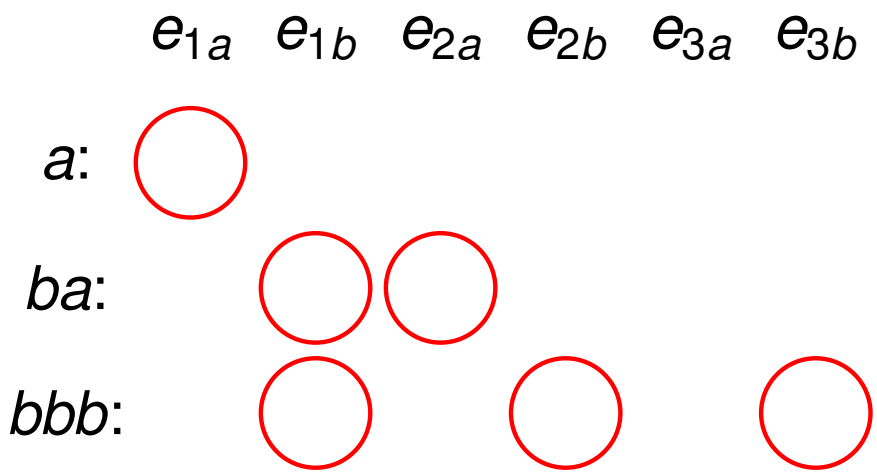
Isolating Lemma:
Probably there will be a unique minimum weight word.



# Proof Idea

Consider the short counterexamples as subsets.

Isolating Lemma:
Probably there will be a unique minimum weight word.




Consider the univariate polynomial

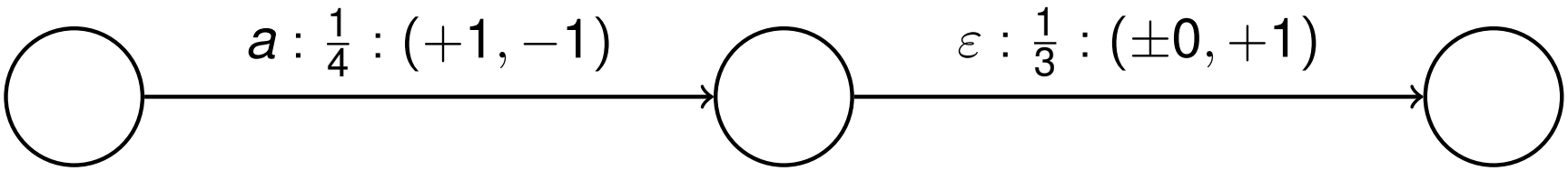
$$p(x) \quad := \sum_{w \in \Sigma^{\leq n_1+n_2-1}} (\mathcal{A}_1(w) - \mathcal{A}_2(w))\, x^{\text{weight}(w)}\,.$$

- if $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent, then $p(x) \equiv 0$;
- if $\mathcal{A}_1$ and $\mathcal{A}_2$ are not equivalent, then probably $p(x) \not\equiv 0$ (consider the coefficient of $x^m$ where $m =$ minimum weight).

The polynomial $p(x)$ can be efficiently computed in parallel.

# Probabilistic Cost Automata

Probabilistic cost automata = automata as before plus counters:




Motivation: consumption of resources: time, memory, energy, ...

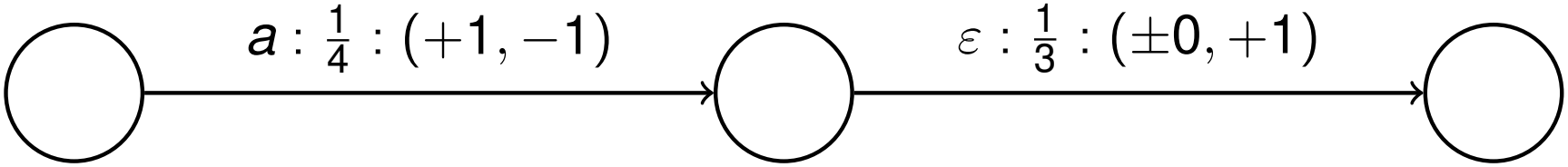
A cost automaton $\mathcal{A}$ induces a cost language $\mathcal{A} : \Sigma^* \times \mathbb{Z}^s \to [0, 1]$.

Equivalence Problem:

Given two automata, do they induce the same cost language?

# Probabilistic Cost Automata

Probabilistic cost automata = automata as before plus counters:




Motivation: consumption of resources: time, memory, energy, ...

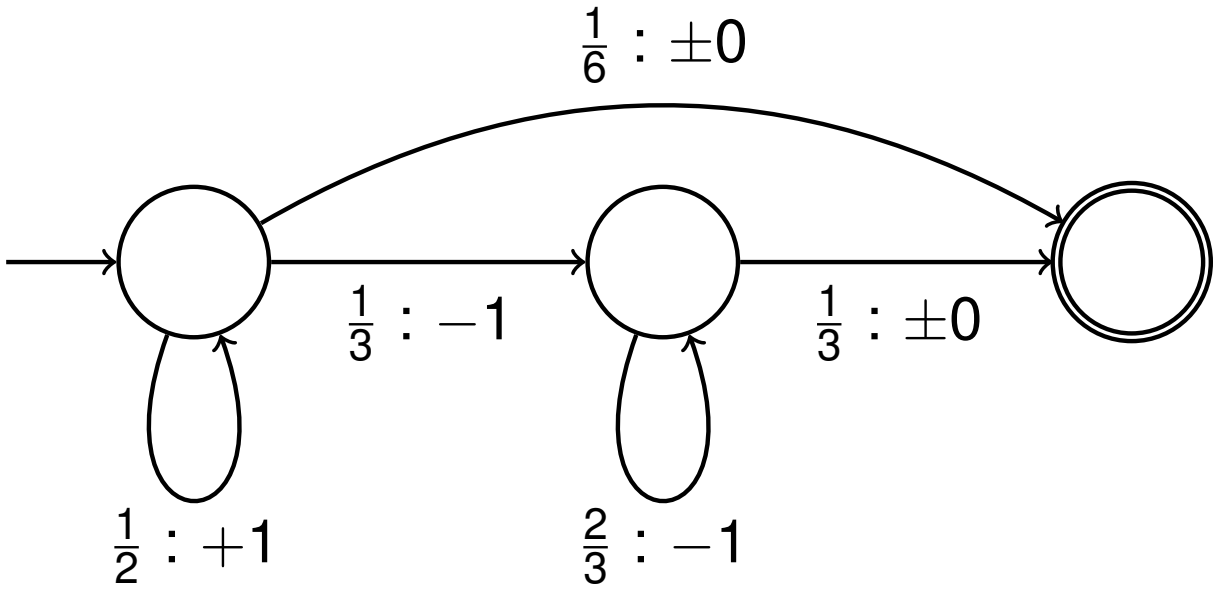
A cost automaton $\mathcal{A}$ induces a cost language $\mathcal{A} : \Sigma^* \times \mathbb{Z}^s \to [0, 1]$.

Equivalence Problem:
Given two automata, do they induce the same cost language?

$\varepsilon$-transitions now require care.

| | | |
|---|---|---|
| paper: | finite alphabet $\Sigma = \{a, b, \ldots\}$; | finitely many counters |
| talk: | empty alphabet $\Sigma = \emptyset$; | 1 counter |
| | $\longrightarrow \mathcal{A} : \mathbb{Z} \to [0, 1]$ | |

# Probabilistic Cost Automata



# Two APEX programs

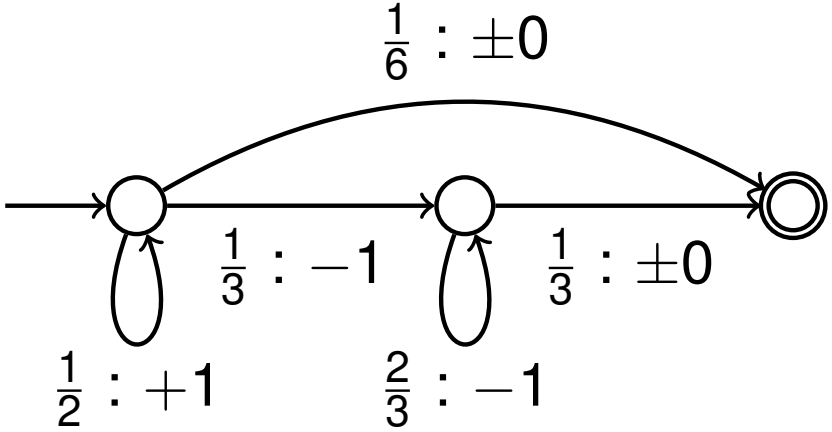
Two APEX programs for the difference between two geometrically distributed random variables:

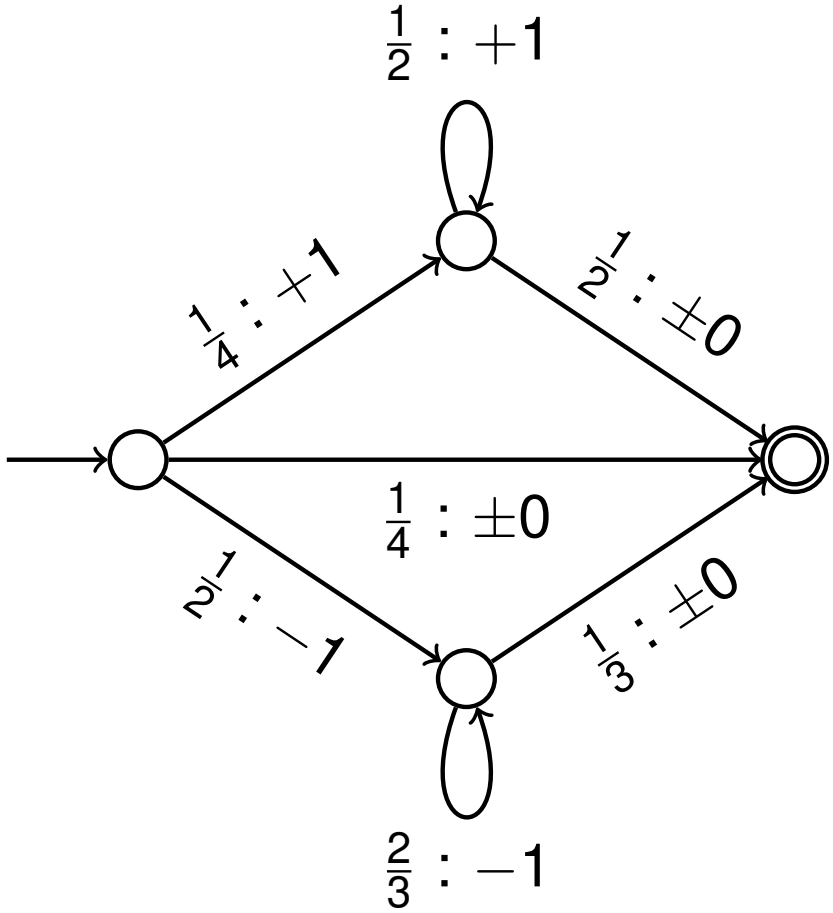
```
inc:com, dec:com |-
  var%2 flip;
  flip := 0;
  while (flip = 0) do {
    flip := coin[0:1/2,1:1/2];
    if (flip = 0) then {
     inc;
    };
  };
  flip := 0;
  while (flip = 0) do {
    flip := coin[0:2/3,1:1/3];
    if (flip = 0) then {
     dec;
    };
  }
```

```
inc:com, dec:com |-
  var%2 flip;
  flip := coin[0:1/2,1:1/2];
  if (flip = 0) then {
    while (flip = 0) do {
      flip := coin[0:1/2,1:1/2];
      if (flip = 0) then {
       inc;
      };
    };
  } else {
    flip := 0;
    while (flip = 0) do {
      dec;
      flip := coin[0:2/3,1:1/3];
    };
  }
```

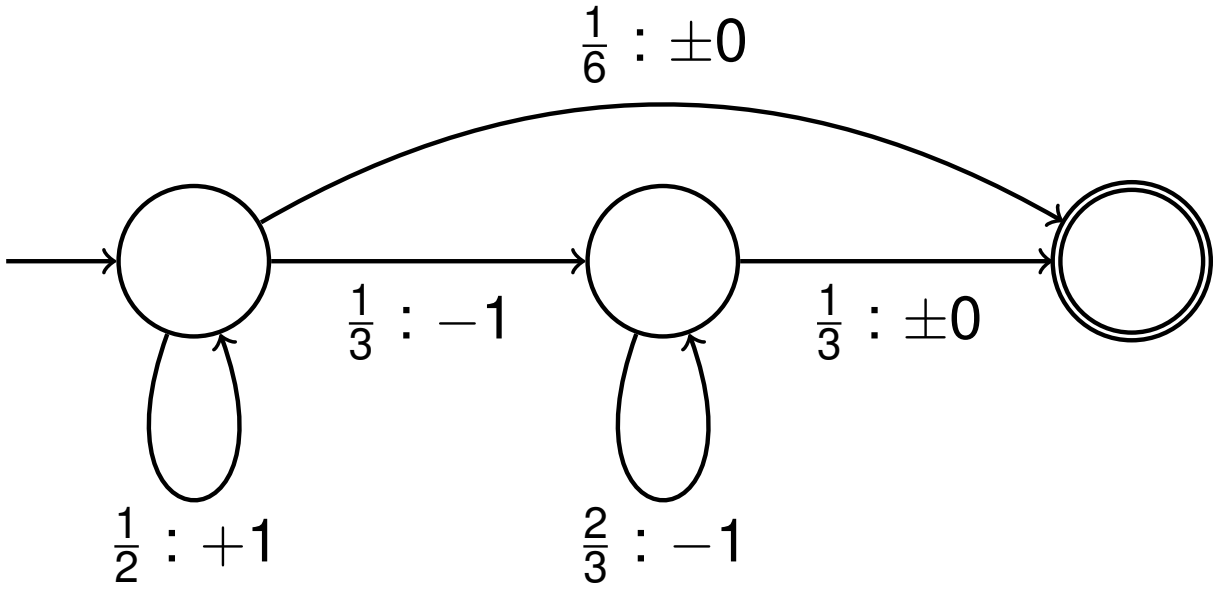
# The Resulting Automata

APEX computes these two automata:

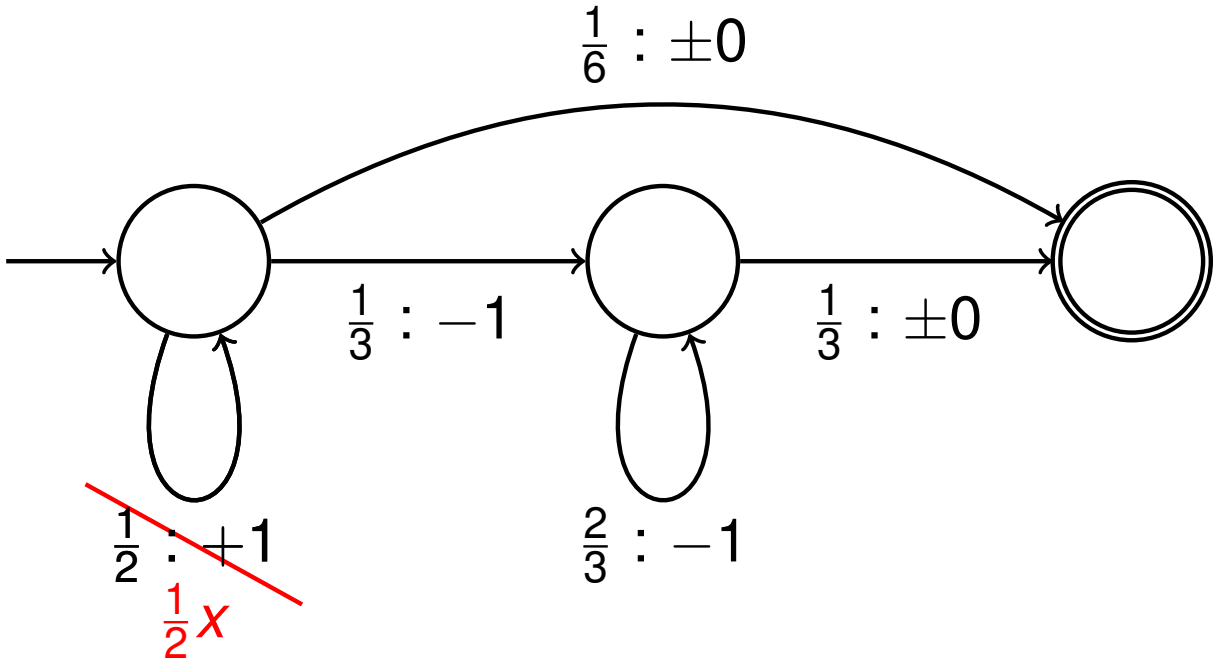


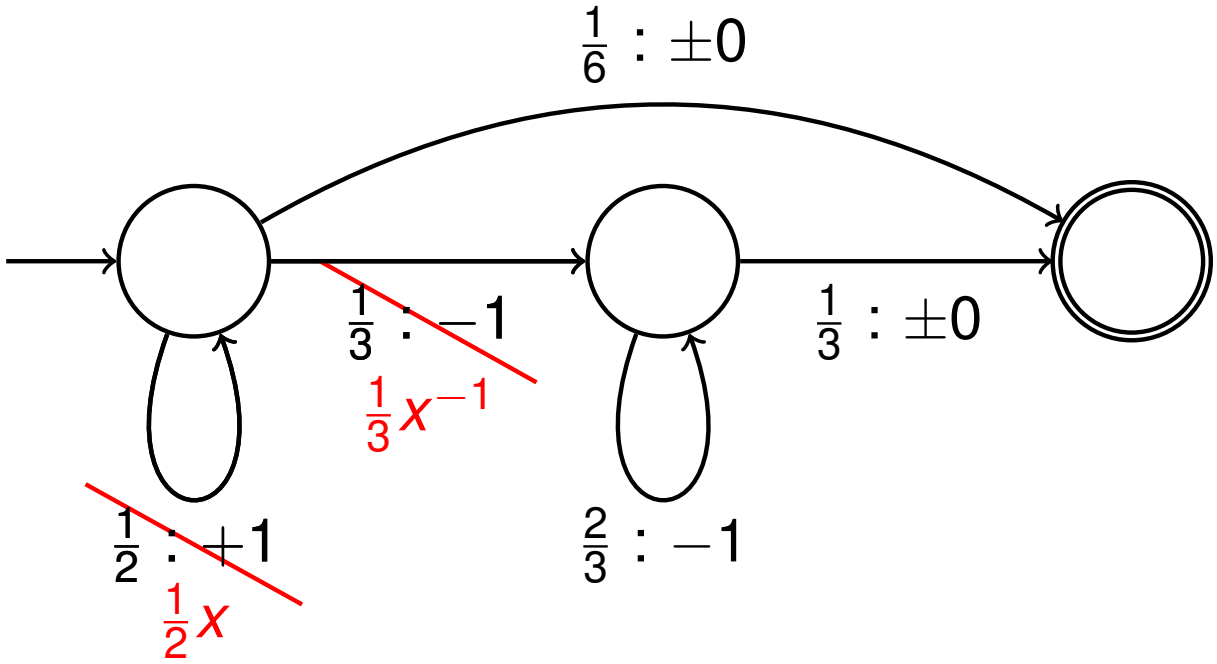
# Analysing Cost Automata



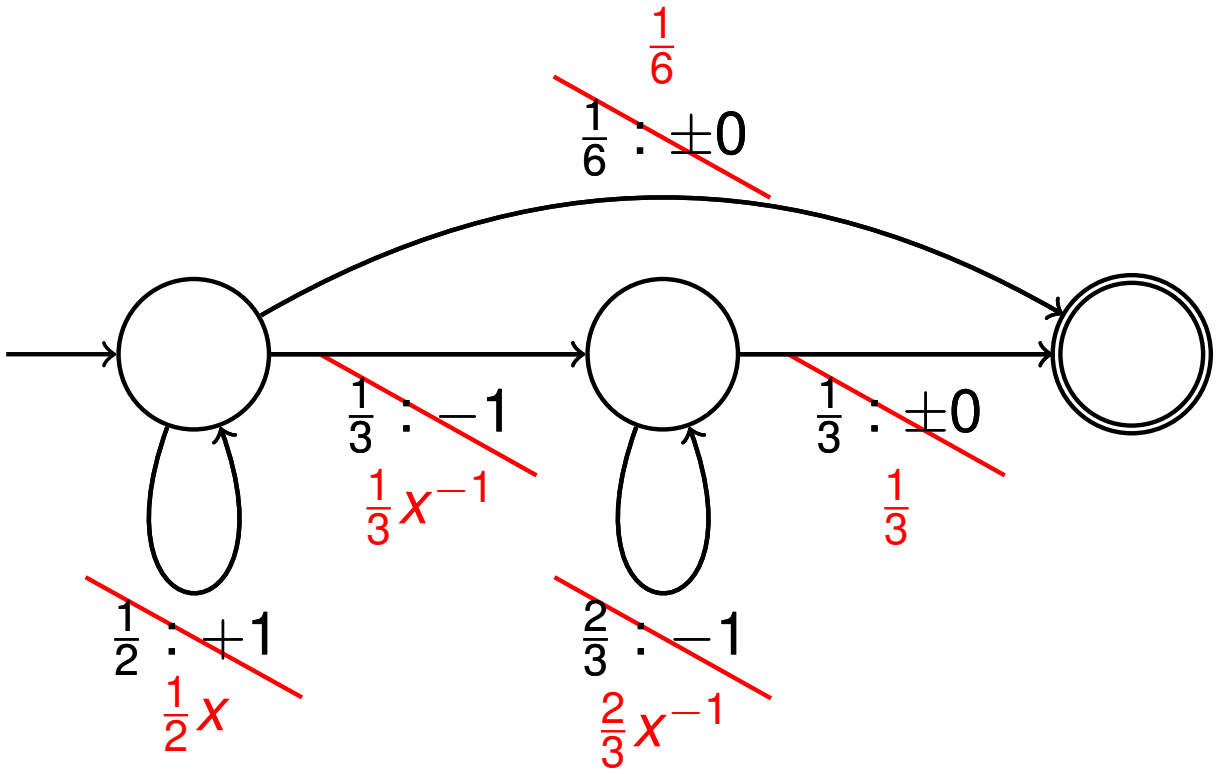
# Analysing Cost Automata



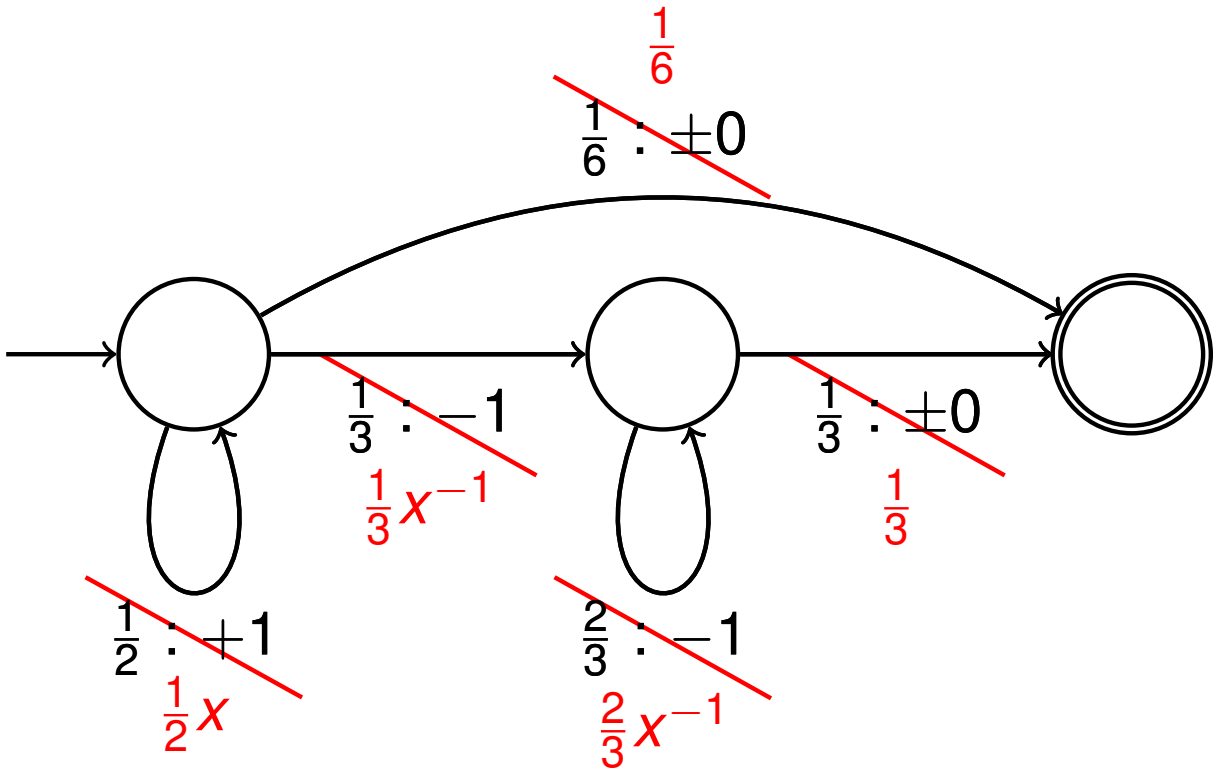
# Analysing Cost Automata



# Analysing Cost Automata



# Analysing Cost Automata



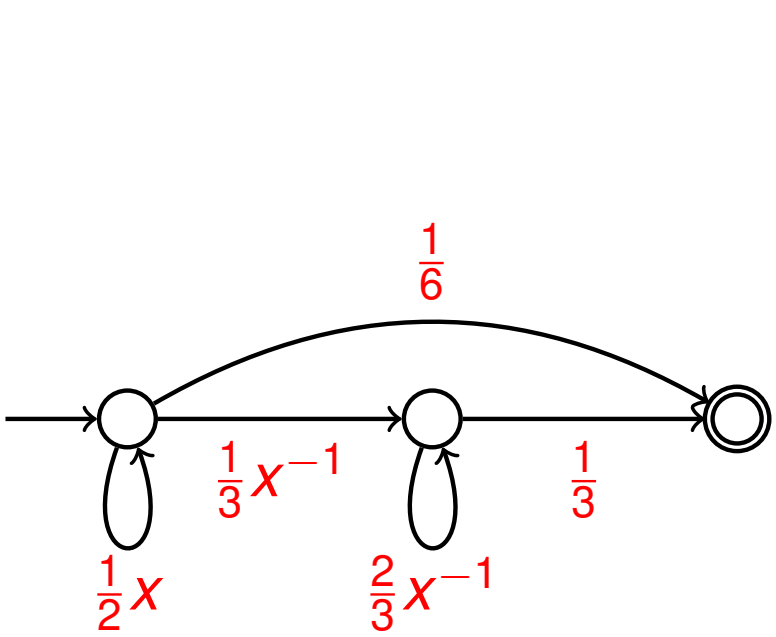

Sum over all paths: $p(x) = \frac{1}{6} + \frac{1}{3}x^{-1} \cdot \frac{1}{3} + \frac{1}{2}x \cdot \frac{1}{6} + \cdots$

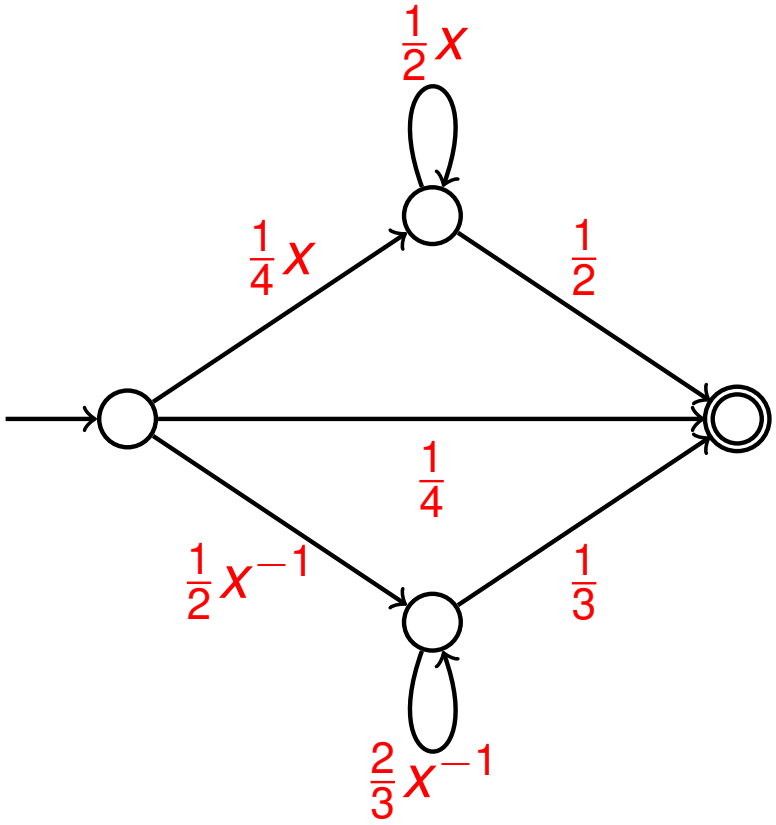
The coefficient of $x^k$ in this power series is exactly $\mathcal{A}(k)$.

$$p(x) = \frac{-x}{(3x-2)(x-2)}$$

# Comparing Two Automata

Do this for the two previous automata:

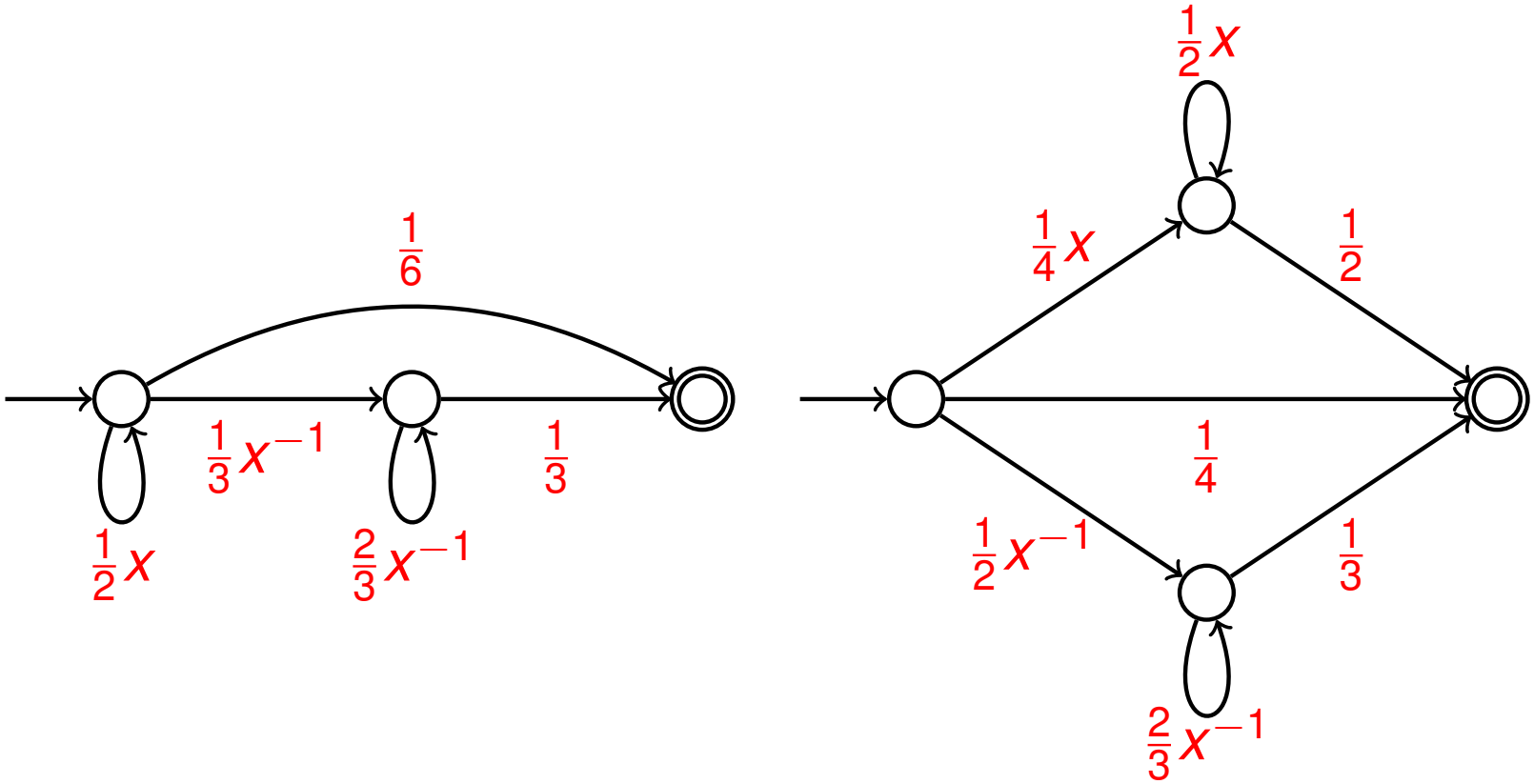



$$p_1(x) = \frac{-x}{(3x-2)(x-2)} \qquad p_2(x) = \frac{-x}{(3x-2)(x-2)}$$

# Comparing Two Automata

Do this for the two previous automata:




$$p_1(x) = \frac{-x}{(3x-2)(x-2)} \qquad p_2(x) = \frac{-x}{(3x-2)(x-2)}$$

Don't compute $p_1(x)$ and $p_2(x)$, only evaluate at random points:
Is $p_1(8) = p_2(8)$? Is $p_1(42) = p_2(42)$? etc.

# Results

Using the Schwartz-Zippel lemma we get:

**Theorem**

*The equivalence problem for probabilistic cost automata is decidable in randomised polynomial time.*

The procedure can be derandomised:

**Theorem**

*For each fixed number of counters*
*the equivalence problem for probabilistic cost automata is decidable in deterministic polynomial time.*

# Application: RSA Timing Attacks

Application: modeling timing attacks: counter = time

RSA encryption with and without probabilistic blinding

Vulnerability to timing attack
= Inequivalence of probabilistic cost automata

# Probabilistic Visibly Pushdown Automata

probabilistic pushdown automaton:

(current configuration, input letter) $\mapsto$ distribution of configurations

probabilistic visibly pushdown automaton:

3 types of input letters: push, pop, internal

# Probabilistic Visibly Pushdown Automata

probabilistic pushdown automaton:

(current configuration, input letter) $\mapsto$ distribution of configurations

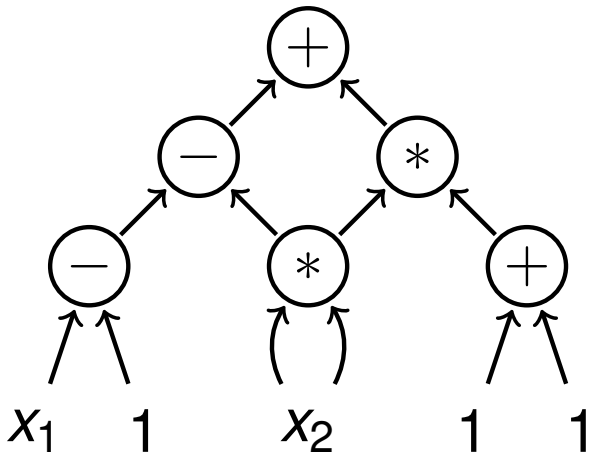
probabilistic visibly pushdown automaton:

3 types of input letters: push, pop, internal

**Theorem**

*Equivalence of probabilistic visibly pushdown automata is logspace equivalent to the ACIT problem.*

ACIT = Acyclic Circuit Identity Testing




$=: p(x) \equiv 0?$

# Probabilistic Visibly Pushdown Automata

probabilistic pushdown automaton:

(current configuration, input letter) $\mapsto$ distribution of configurations

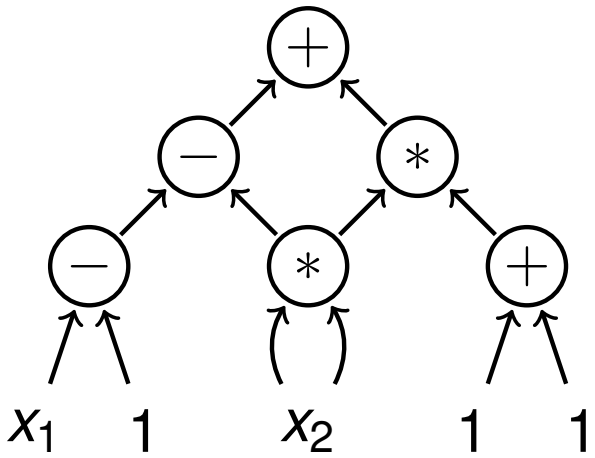
probabilistic visibly pushdown automaton:

3 types of input letters: push, pop, internal

**Theorem**

*Equivalence of probabilistic visibly pushdown automata is logspace equivalent to the ACIT problem.*

ACIT = Acyclic Circuit Identity Testing is in coRP.




$=: p(x) \equiv 0?$

# Summary

Motivated by verification of probabilistic programs
we studied equivalence of probabilistic automata.

New complexity results for various models:

| | |
|---|---|
| finite aut. + counterexample | $\in$ RNC |
| finite aut. + counters | $\in$ coRP |
| finite aut. + fixed number of counters | $\in$ P |
| visibly pushdown aut. | ACIT-complete |

# End of Talk

Thank you!