Model Checking Probabilistic Recursive Programs

MOVES Seminar

Tobias Winkler

13.04.2021

UnRAVeL | RWTH AACHEN UNIVERSITY | Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline

- Probabilistic Recursive Programs
- Probabilistic Push-Down Automata
- From Programs to Automata
- Implementation & Examples

What are Probabilistic Recursive Programs?

Probabilistic Recursive programs
=
Imperative programs + coin flips + recursive procedures

This talk

```
int faultyFib(int n) {
    if(n = 0) return 0;
    if(n = 1) return 1;
    if(flip(0.01)) n = n+1; // error!
    return faultyFib(n-1) + faultyFib(n-2);
}
```

Termination? (Expected) Runtime? Difference to correct Fib?

Motivation

Randomized Recursive Algorithms

```
void quicksort(int[] a, int left, int right) {
    if(left < right) {
        int pivot = randomPartition(a, left, right);
        quicksort(a, left, pivot-1);
        quicksort(a, pivot+1, right);
    }
}
```

Expected Runtime?

Motivation

Survival Probabilities

```
infectYoung() {
    int y = geom(0.3);
    int o = geom(0.1);
    do y times { infectYoung(); }
    do o times { infectOld(); }
}

infectOld() {
    int y = geom(0.1);
    int o = geom(0.5);
    do y times { infectYoung(); }
    do o times { infectOld(); }
}
```

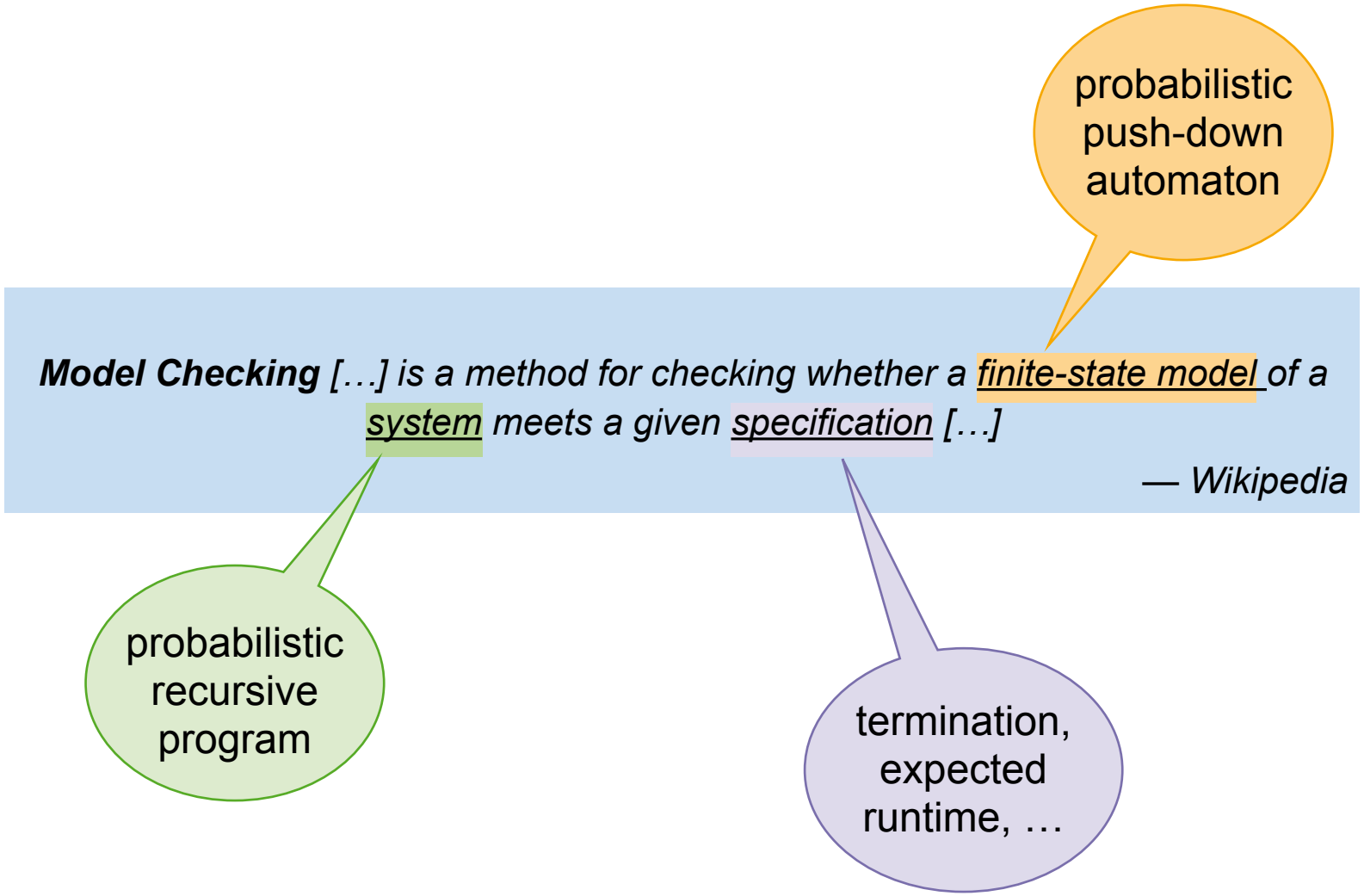
Will the virus die out without external measures? (termination probability)

Motivation

Application Overview

- ✦ General Randomized Recursive Algorithms
- ✦ Branching Processes
  - ‣ Epidemiology, nuclear physics, genomics, ecology, …
- ✦ Boltzmann Samplers
  - ‣ Testing, benchmarking, empiric exploration of complex structures
- ✦ Probabilistic Programming Languages for Bayesian Inference (e.g. WebPPL)

What is Model Checking?



Outline

- [x] Probabilistic Recursive Programs
- [ ] Probabilistic Push-Down Automata
- [ ] From Programs to Automata
- [ ] Implementation & Examples

Probabilistic Pushdown Automata

Definition

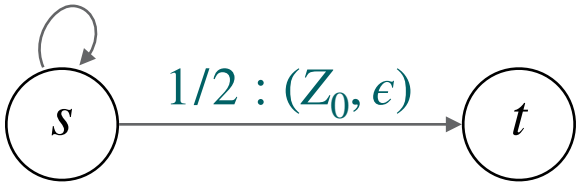
A probabilisitic push-down automaton (PPDA) is a tuple (Q, Γ, Z_0, P), where

- Q *is a set of states*
- Γ *is a set of stack symbols*
- $Z_0 \in \Gamma$ *is the bottom stack symbol*
- $P\colon Q \times \Gamma \dashrightarrow Dist(Q \times \Gamma^{\leq 2})$ *is a (partial) probabilistic transition function.*

In state $s \in Q$ pop $Z \in \Gamma$ from the stack…

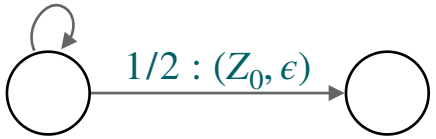
… and go to $t \in Q$, pushing $\gamma \in \Gamma^{\leq 2}$ on the stack.

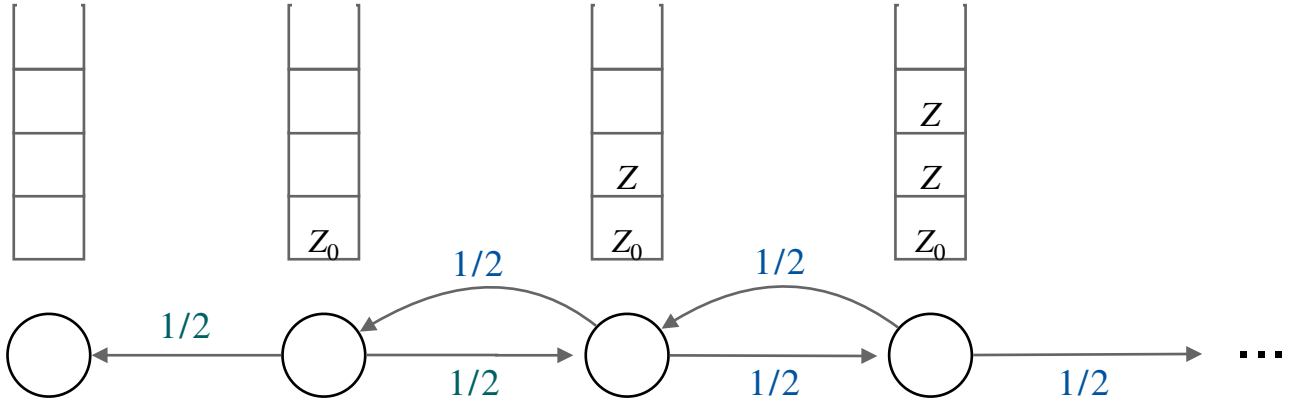
$1/2 : (Z_0, Z_0 Z)$
$1/2 : (Z, ZZ)$
$1/2 : (Z, \epsilon)$



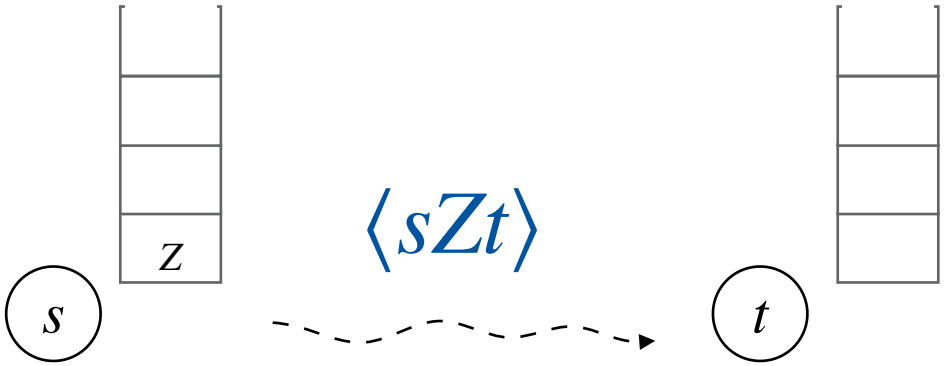
Markov chain semantics of PPDA

$1/2 : (Z_0, Z_0 Z)$
$1/2 : (Z, ZZ)$
$1/2 : (Z, \epsilon)$





Computing Termination Probabilities



$\langle sZt \rangle$ = “probability to reach t with empty stack starting in s with a stack containing only Z”

$$\langle sZt \rangle = \sum_{sZ \xrightarrow{p} rXY} \sum_{q \in Q} p\langle rYq \rangle \langle qXt \rangle + \sum_{sZ \xrightarrow{p} rX} p\langle rXt \rangle + \sum_{sZ \xrightarrow{p} t\epsilon} p$$

push transitions

internal transitions

pop transitions

Termination Probabilities as Fixed Points

Theorem *(Esparza+ LICS '04, Etessami+ STACS '05)*

Let $[sZt], s, t \in Q, Z \in \Gamma$ *be* $|Q|^2 \cdot |\Gamma|$ *many variables.*
The least non-negative solution of the polynomial equation system

$$[sZt] = \sum_{sZ \xrightarrow{p} rXY} \sum_{q \in Q} p[rYq][qXt] + \sum_{sZ \xrightarrow{p} rX} p[rXt] + \sum_{sZ \xrightarrow{p} t\epsilon} p$$

is given by the termination probabilities $\langle sZt \rangle = [sZt]$.

Corollary
Quantitative termination in PPDA is decidable in PSPACE.

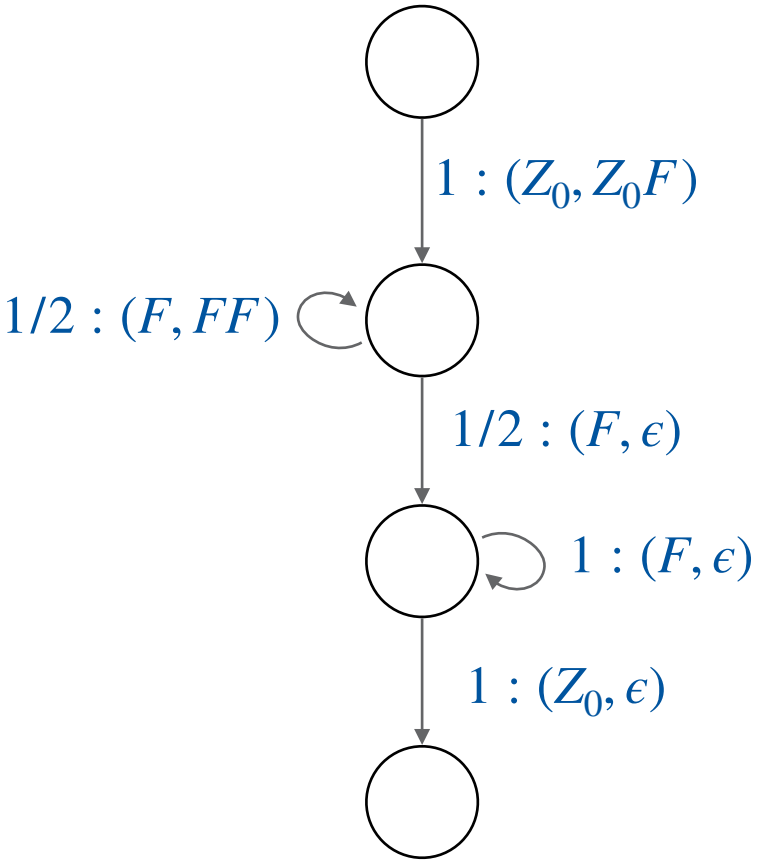
Outline

- [x] Probabilistic Recursive Programs
- [x] Probabilistic Push-Down Automata
- [ ] From Programs to Automata
- [ ] Implementation & Examples

From Programs to Automata, 1st attempt

Idea: Encode call stack in automaton stack

```
void F() {
  if flip(1/2) {
    F(); // recursive call
  }
  return;
}

F(); // main
```



From Programs to Automata, 2nd attempt

```
int faultyFib(int n) {
    if(n = 0) return 0;
    if(n = 1) return 1;
    if(flip(0.01)) n = n+1;
    return faultyFib(n-1) + faultyFib(n-2);
}

faultyFib(10); // main
```

?

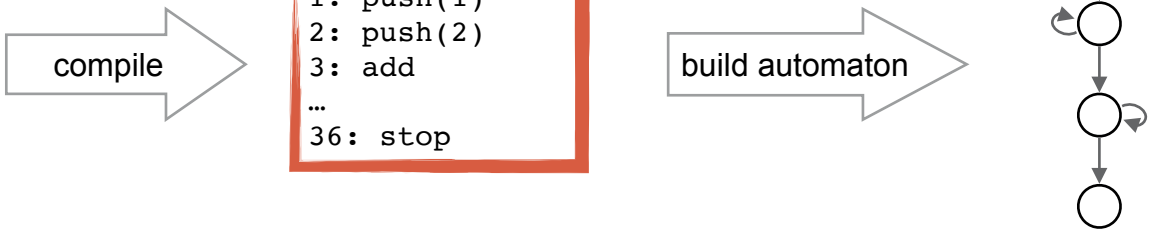
My experience:

Defining a PPDA semantics for a rich programming language directly is hard.

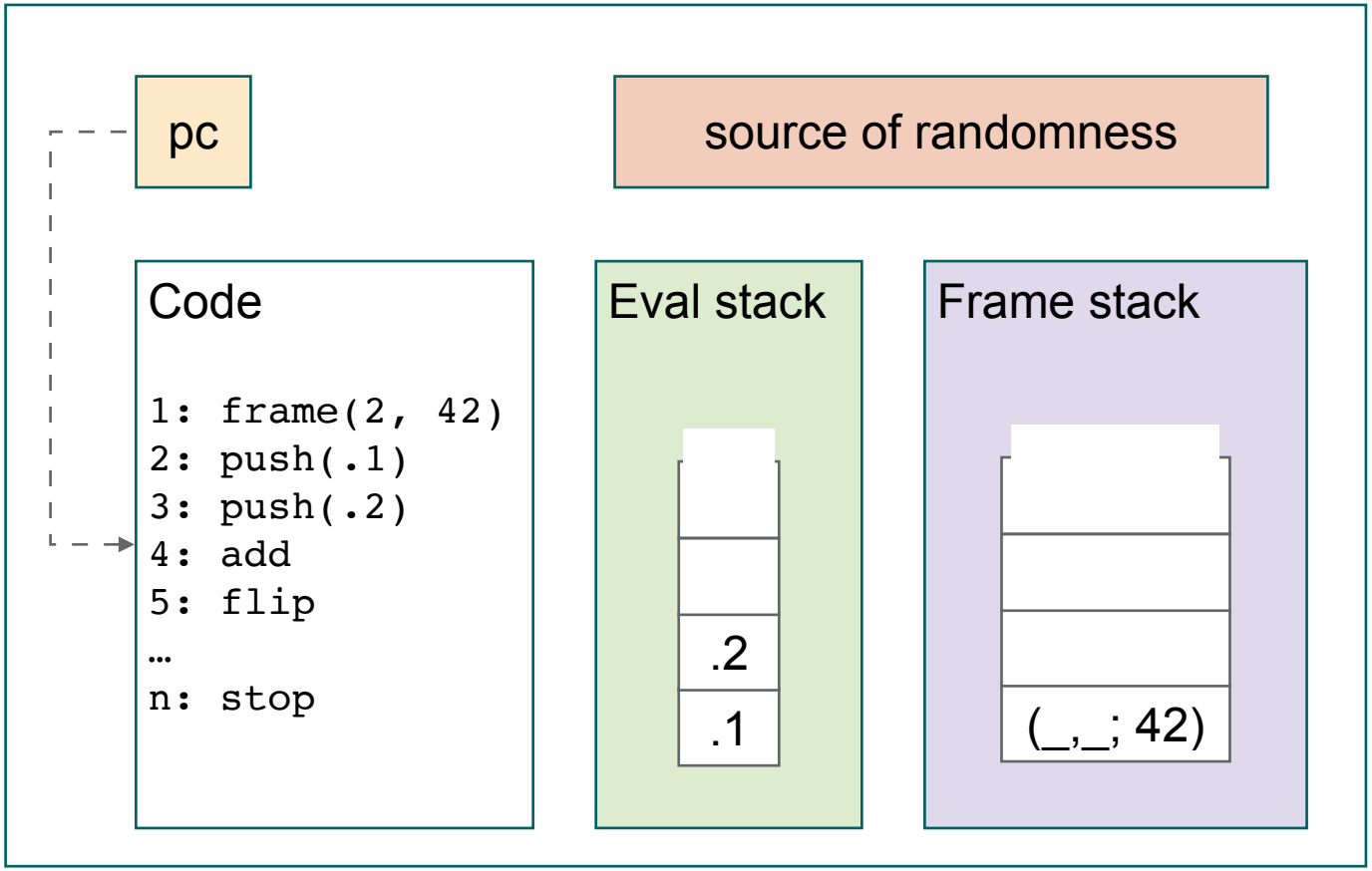
An intermediate step

Idea: Compile high level input language to a (simple) low level machine language

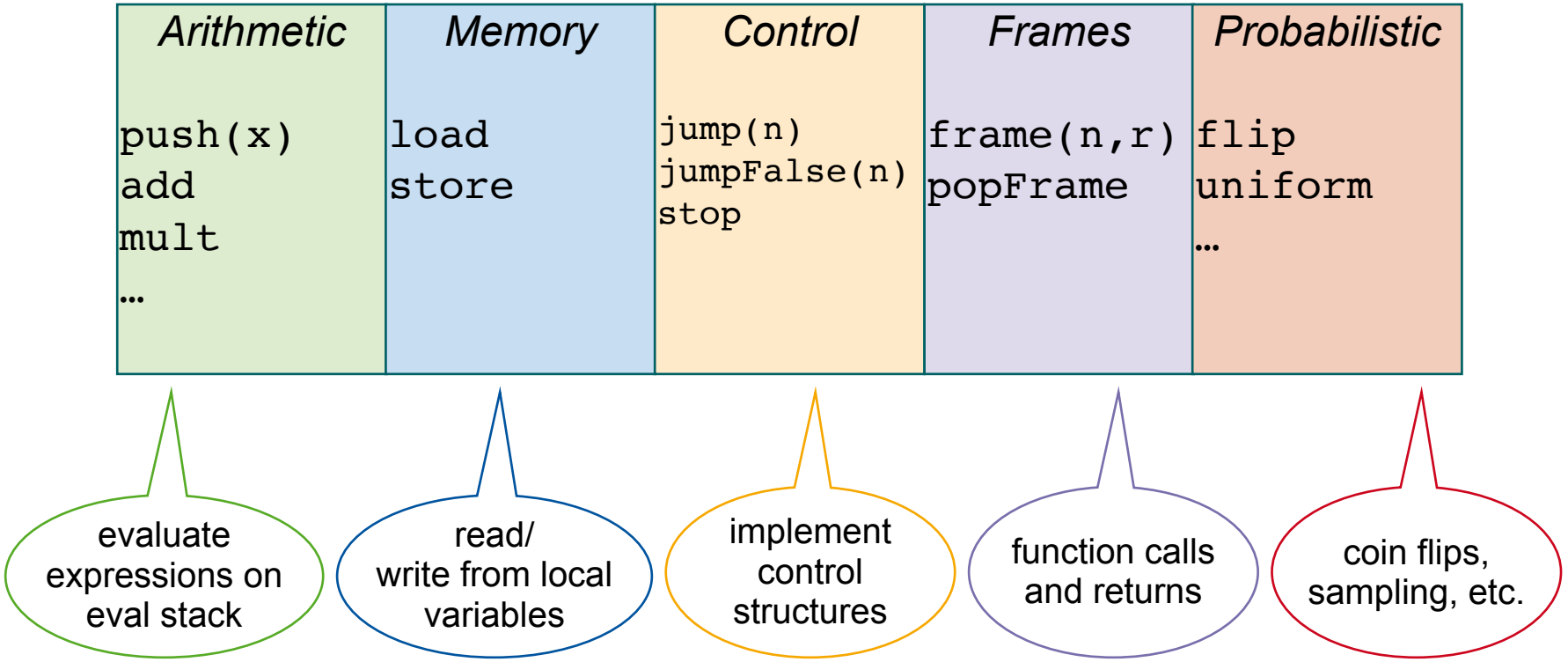
```
int faultyFib(int n) {
    if(n = 0) return 0;
    if(n = 1) return 1;
    if(flip(0.01)) n = n+1;
    return faultyFib(n-1) +
faultyFib(n-2);
}
```



Probabilistic Abstract Machine — Architecture



Probabilistic Abstract Machine — Code



Probabilistic Abstract Machine — Compilation

```
void F() {
  if flip(1/2) {
    F(); // recursive call
  }
  return;
}

F(); // main
```

```
1: newFrame(0, 3)
2: jump(4)
3: stop
4: push(0.5)
5: flip
6: jumpFalse(9)
7: newFrame(0, 9)
8: jump(4)
9: popFrame
```

PPDA semantics of the Probabilistic Abstract Machine

Local states: $\mathscr{L} = \underbrace{\mathbb{N}}_{\text{PC}} \times \underbrace{Val^*}_{\text{eval stack}} \times \underbrace{\mathscr{F}}_{\text{current frame}}$

Frames: $\underbrace{\mathscr{F}}_{} = \underbrace{Val^*}_{\text{local vars}} \times \underbrace{\mathbb{N}}_{\text{return address}}$

Define the PPDA (Q, Γ, P, Z_0) with

- $Q \subseteq \mathscr{L}$
- $\Gamma \subseteq \mathscr{F} \cup \{Z_0\}$

according to the following rules

PPDA semantics of the Probabilistic Abstract Machine

Arithmetic	*Memory*	*Control*	*Frames*	*Probabilistic*

`push(x)`
`add`
`mult`
…
	`load`
	`store`
		`jump(n)`
		`jumpFalse(n)`
		`stop`
			`frame(n,r)`
			`popFrame`
				`flip`
				`uniform`
				…

$$\frac{}{\langle 0, \epsilon, Z_0 \rangle \in Q} \text{ (initial state)}$$

PPDA semantics of the Probabilistic Abstract Machine

Arithmetic	*Memory*	*Control*	*Frames*	*Probabilistic*

`push(x)`
`add`
`mult`
…
	`load`
	`store`
		`jump(n)`
		`jumpFalse(n)`
		`stop`
			`frame(n,r)`
			`popFrame`
				`flip`
				`uniform`
				…

$$\frac{pc : \texttt{push(x)} \quad \langle pc, \sigma, \phi \rangle \in Q}{\langle pc, \sigma, \phi \rangle \xrightarrow{1} \langle pc + 1,\ \sigma \,.\, x,\ \phi \rangle}$$

$$\frac{pc : \texttt{add} \quad \langle pc, \sigma \,.\, x \,.\, y, \phi \rangle \in Q}{\langle pc, \sigma, \phi \rangle \xrightarrow{1} \langle pc + 1,\ \sigma \,.\, (x + y),\ \phi \rangle}$$

Other arithmetic instructions similar

PPDA semantics of the Probabilistic Abstract Machine

Arithmetic	*Memory*	*Control*	*Frames*	*Probabilistic*

`push(x)`
`add`
`mult`
…
	`load`
	`store`
		`jump(n)`
		`jumpFalse(n)`
		`stop`
			`frame(n,r)`
			`popFrame`
				`flip`
				`uniform`
				…

$$\frac{pc : \texttt{load} \quad \langle pc, \sigma\,.\,x, \phi\rangle \in Q \quad \phi = (y_1, \ldots, y_x, \ldots y_n; r)}{\langle pc, \sigma\,.\,x, \phi\rangle \xrightarrow{1} \langle pc + 1,\ \sigma\,.\,y_x,\ \phi\rangle}$$

$$\frac{pc : \texttt{store} \quad \langle pc, \sigma\,.\,x\,.\,y, \phi\rangle \in Q \quad \phi = (y_1, \ldots, y_x, \ldots y_n; r)}{\langle pc, \sigma\,.\,x\,.\,y, \phi\rangle \xrightarrow{1} \langle pc + 1,\ \sigma,\ (y_1, \ldots, y, \ldots y_n; r)\rangle}$$

PPDA semantics of the Probabilistic Abstract Machine

Arithmetic	*Memory*	*Control*	*Frames*	*Probabilistic*

`push(x)`
`add`
`mult`
…
	`load`
	`store`
		`jump(n)`
		`jumpFalse(n)`
		`stop`
			`frame(n,r)`
			`popFrame`
				`flip`
				`uniform`
				…

$$\frac{pc : \texttt{jump(n)} \quad \langle pc, \sigma, \phi \rangle \in Q}{\langle pc, \sigma, \phi \rangle \xrightarrow{1} \langle n, \sigma, \phi \rangle}$$

$$\frac{pc : \texttt{jumpFalse(n)} \quad \langle pc, \sigma.0, \phi \rangle \in Q}{\langle pc, \sigma.0, \phi \rangle \xrightarrow{1} \langle n, \sigma, \phi \rangle}$$

$$\frac{pc : \texttt{jumpFalse(n)} \quad \langle pc, \sigma.1, \phi \rangle \in Q}{\langle pc, \sigma.1, \phi \rangle \xrightarrow{1} \langle pc + 1, \sigma, \phi \rangle}$$

PPDA semantics of the Probabilistic Abstract Machine

Arithmetic	*Memory*	*Control*	*Frames*	*Probabilistic*

`push(x)`
`add`
`mult`
…
	`load`
	`store`
		`jump(n)`
		`jumpFalse(n)`
		`stop`
			`frame(n,r)`
			`popFrame`
				`flip`
				`uniform`
				…

$$\frac{pc : \texttt{flip} \quad \langle pc, \sigma \,.\, p, \phi \rangle \in Q}{\langle pc, \sigma \,.\, p, \phi \rangle \xrightarrow{p} \langle pc + 1,\ \sigma.1,\ \phi \rangle \quad \langle pc, \sigma \,.\, p, \phi \rangle \xrightarrow{1-p} \langle pc + 1,\ \sigma.0,\ \phi \rangle}$$

Other probabilistic instructions similar

PPDA semantics of the Probabilistic Abstract Machine

Arithmetic	*Memory*	*Control*	*Frames*	*Probabilistic*

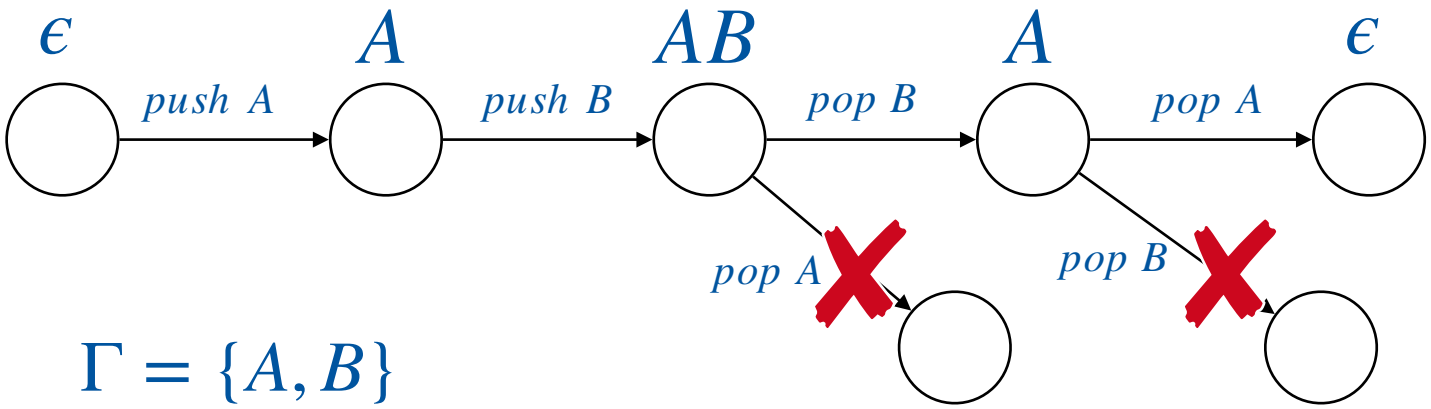
`push(x)`
`add`
`mult`
…
	`load`
	`store`
		`jump(n)`
		`jumpFalse(n)`
		`stop`
			`frame(n,r)`
			`popFrame`
				`flip`
				`uniform`
				…

$$\frac{pc : \texttt{frame(n, r)} \quad \langle pc, \sigma, \phi \rangle \in Q}{\langle pc, \sigma, \phi \rangle \xrightarrow[\phi]{push} \langle pc + 1,\ \sigma,\ (\underbrace{\_, \ldots, \_}_{n}; r) \rangle \quad \phi \in \Gamma}$$

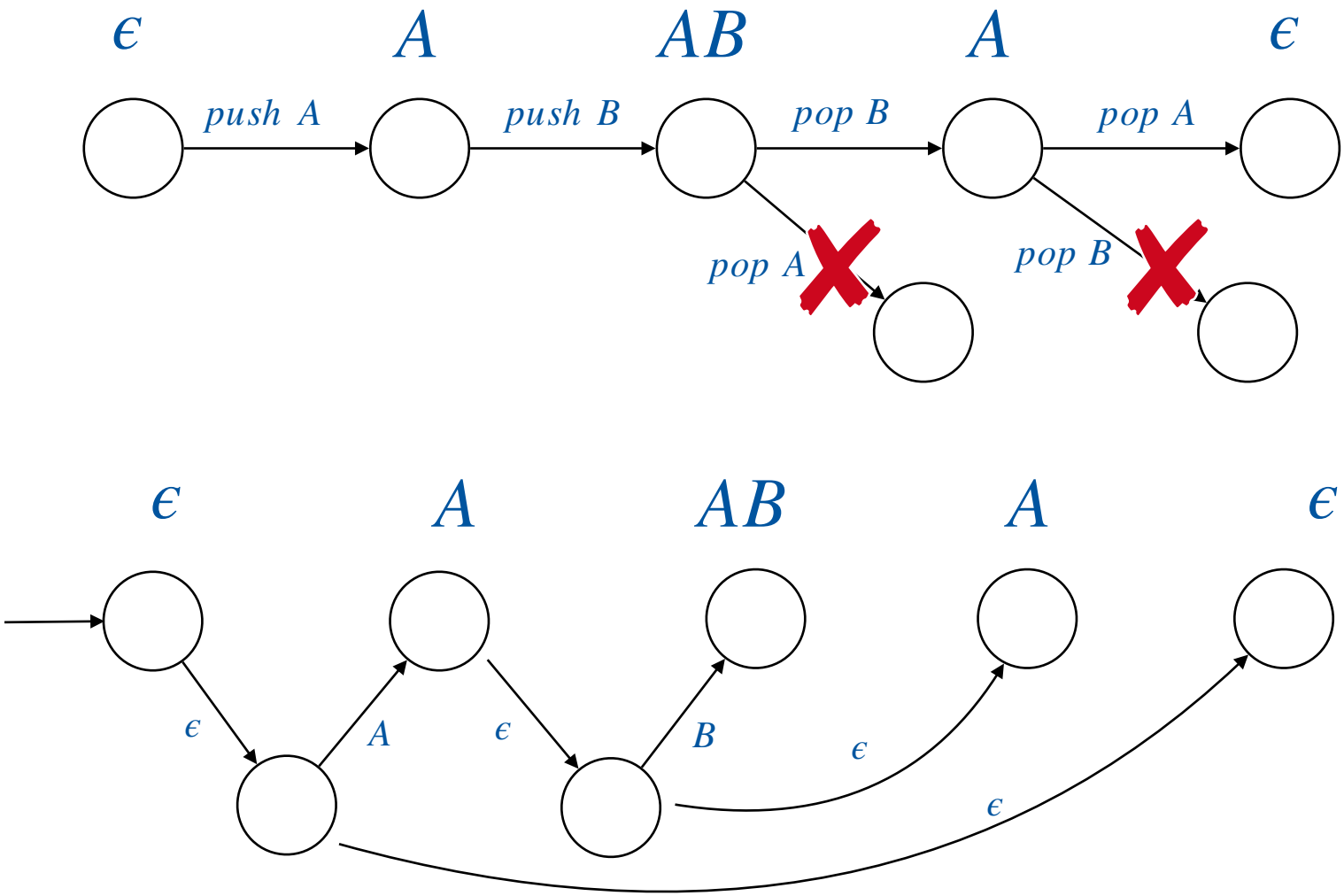
$$\frac{pc : \texttt{popframe} \quad \langle pc, \sigma, \phi \rangle \in Q \quad \phi = (\ldots, r) \quad \phi' \in \Gamma \setminus \{Z_0\}}{\langle pc, \sigma, \phi \rangle \xrightarrow[\phi']{pop} \langle pc + 1,\ \sigma \cdot r,\ \phi' \rangle}$$

Unreachable transitions

Automaton construction may lead to (many) unreachable transitions!



Removing Unreachable Transitions [Grune et al.]

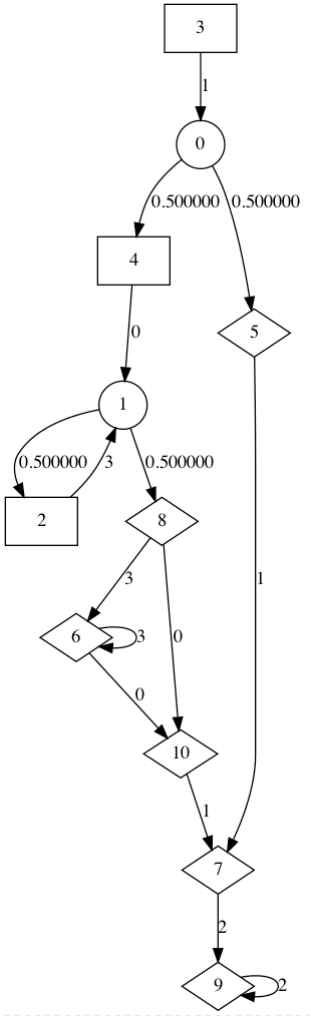


Outline

- ☑ Probabilistic Recursive Programs
- ☑ Probabilistic Push-Down Automata
- ☑ From Programs to Automata
- ☐ Implementation & Examples

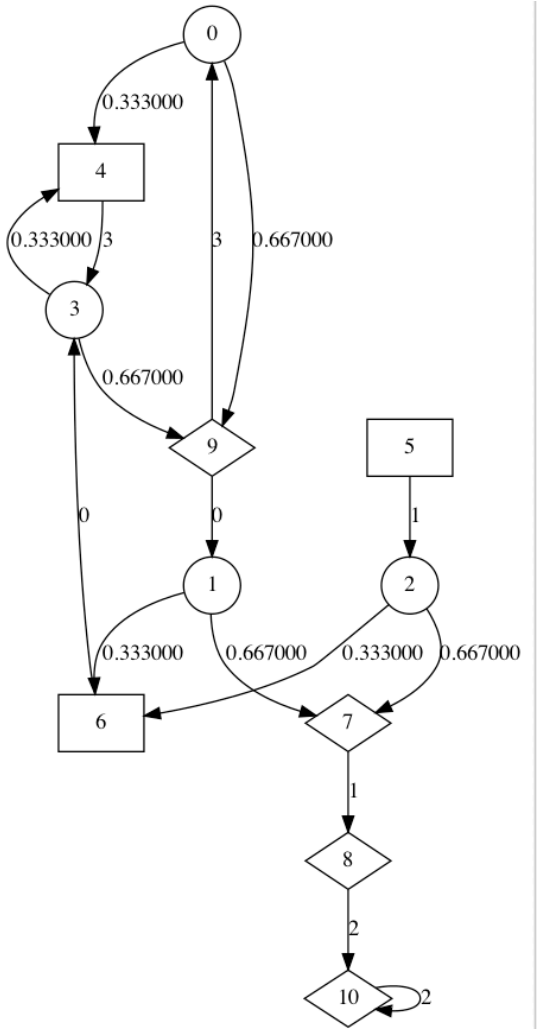
A simple program

```
proc void f() {
    if flip 0.5 then {
        f();
    }
}

{
    f(); // main
}
```



Another program

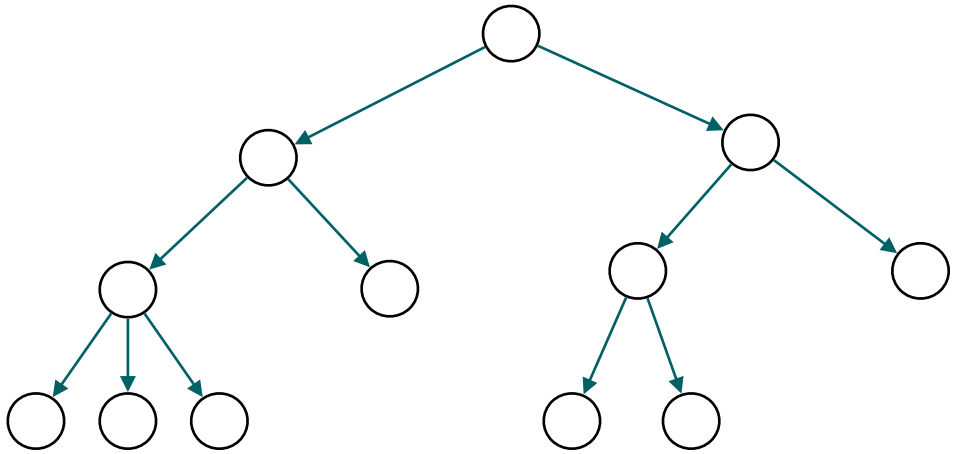
```
proc void offspring() {
    while flip 0.333 {
        offspring();
    }
}

{
    offspring(); // main
}
```



Terminates with probability 1
(numerical approximation)

A more complicated program — Sample 2-3 trees [Duchon et al.]

```
global int leaves;

proc void tree(int n, int k) {
    if flip 0.555 then {
        leaves += 1;
        return;
    }
    tree((n+1) % k, k);
    tree((n+1) % k, k);
    if n == 0 then {
        if flip 0.25 then {
            tree((n+1) % k, k);
        }
    }
}

{
    leaves = 0;
    tree(0, 2);
}
```



Randomly sampled trees have ≈ 5.5 leaves on average

Conclusion, future and ongoing work

- Implemented effective translation High-level program → PPDA
  - low-level machine language as additional abstraction layer
  - transition reachability analysis crucial
- Ongoing work: expected values, higher moments, Newton iteration, more applications
- Future work: Temporal logics (LTL, CaRet,…)
- Very future work: Release as a tool

Thank you very much!

References

- Javier Esparza, Antonín Kucera, Richard Mayr:
  Model Checking Probabilistic Pushdown Automata. LICS 2004: 12-21

- Kousha Etessami, Mihalis Yannakakis:
  Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations. STACS 2005: 340-352

- Tomás Brázdil, Javier Esparza, Stefan Kiefer, Antonín Kucera:
  Analyzing probabilistic pushdown automata. Formal Methods Syst. Des. 43(2): 124-163 (2013)

- Dick Grune, Wan J. Fokkink, Evangelos Chatzikalymnios, Brinio Hond, Peter Rutgers:
  Detecting Useless Transitions in Pushdown Automata. LATA 2017: 421-434

- Philippe Duchon, Philippe Flajolet, Guy Louchard, Gilles Schaeffer:
  Boltzmann Samplers for the Random Generation of Combinatorial Structures. Comb. Probab. Comput. 13(4-5): 577-625 (2004)