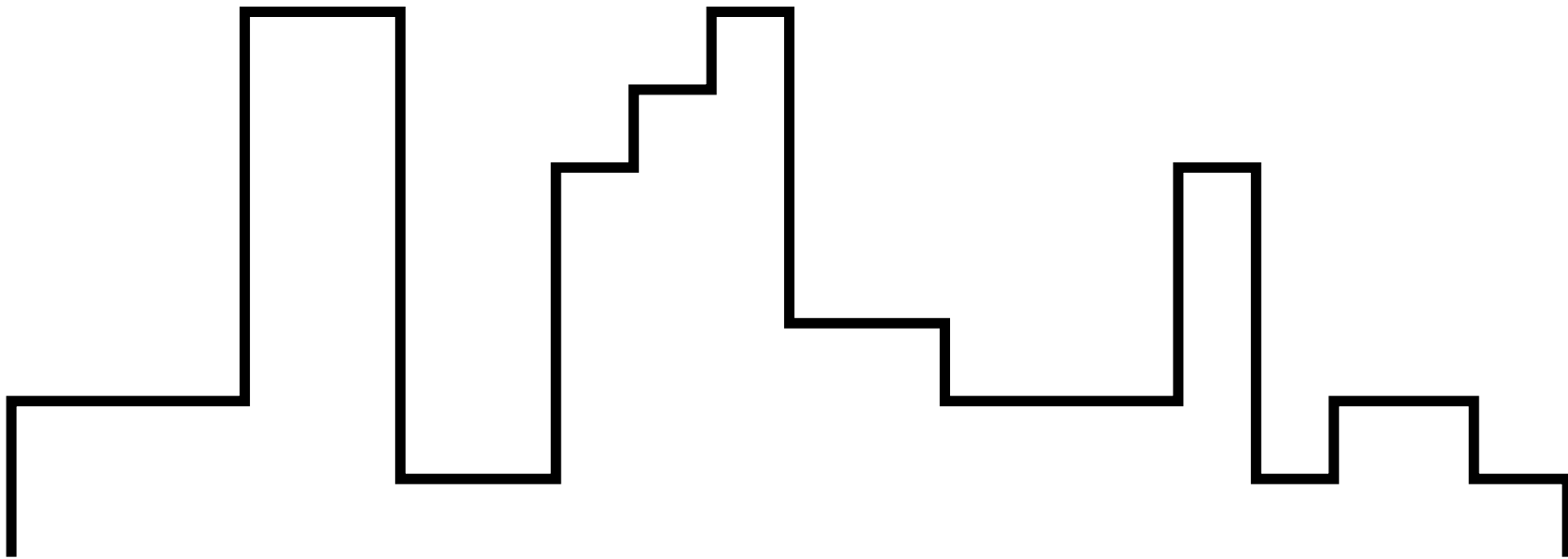

Verifying probabilistic procedural programs

Javier Esparza

Technical University of Munich

Based on work by Javier Esparza, Kousha Etessami, Andreas Gaiser, Stefan Kiefer, Antonín Kučera, Michael Luttenberger, Richard Mayr, Alistair Stewart, Mihalis Yannakakis, and Dominik Wojtczak



Drawing skylines

```
static Random r = new Random();  
static void m() {  
    if (r.nextBoolean()) {  
        s(); right(); if (r.nextBoolean()) m();  
    }  
    else { up(); m(); down(); }  
}  
static void s() {  
    if (r.nextBoolean()) return;  
    up(); m(); down();  
}  
public static void main() { s(); }
```

What is the probability of termination ?

What is the probability of terminating after at most 30 steps?

What is the probability of producing at least one building of height 5?

What is the average length of a skyline ?

Probabilistic verification

Finite Markov chains as model of probabilistic while-programs with finite datatypes.

Decidability and complexity of verification problems extensively studied, very good tool support

Probabilistic programs with recursive procedures may be infinite-state, even if all variables have a finite range (unbounded call stack).

Non-recursive procedure calls can be eliminated using inlining, but inlining may cause an exponential blow-up in the size of the program. This is **inefficient (and unnecessary)**.

Verification of probabilistic procedural programs

We introduce **probabilistic pushdown systems** as a model of **procedural** programs with finite datatypes

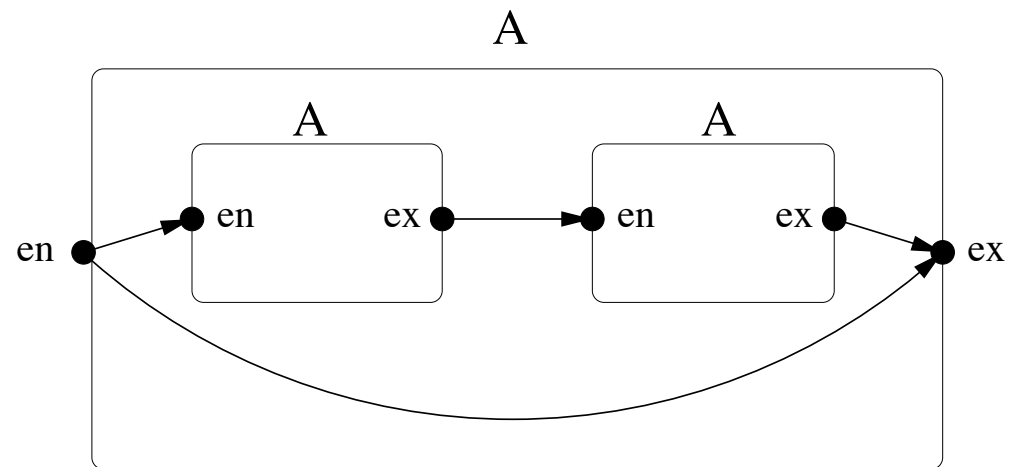
Goal: Design model checkers that work directly on the procedural representation

Abstract models:

- **probabilistic recursive state machines** (PRSM)
- **probabilistic pushdown systems** (PPDS)

RSMs and PDSs

Recursive state machines



Pushdown systems

$$\begin{aligned} pA &\longrightarrow p\epsilon \\ pA &\longrightarrow pAA \end{aligned}$$

Pushdown systems

A pushdown system (PDS) is a triple (P, Γ, δ) , where

- P is a finite set of **control locations**
- Γ is a finite **stack alphabet**
- $\delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of **rules**.

A **configuration** is a pair $p\alpha$, where $p \in P$, $\alpha \in \Gamma^*$

Semantics: A (possibly infinite) transition system with configurations as states and transitions given by

If $pX \hookrightarrow q\alpha \in \delta$ then $pX\beta \longrightarrow q\alpha\beta$ for every $\beta \in \Gamma^*$

Normalisation: $|\alpha| \leq 2$, termination only by empty stack

From programs to pushdown systems

State of a procedural program: $(g, n, l, (n_1, l_1) \dots (n_k, l_k))$, where

- g is a valuation of the global variables,
- n is the value of the program pointer,
- l is a valuation of local variables of the current active procedure,
- n_i is a return address, and
- l_i is a saved valuation of the local variables of the

Modelled as a configuration $pXY_1 \dots Y_k$ where

- $p = g$,
- $X = (n, l)$, and
- $Y_i = (n_i, l_i)$

Correspondence between program statements and rules

procedure call $pX \hookrightarrow qYX$

return $pX \hookrightarrow q\varepsilon$

statement $pX \hookrightarrow qY$

Model

```
static void s() {
```

```
   $s_0$ : if (r.nextBoolean())
```

```
   $s_1$ : return;
```

```
   $s_2$ : up();
```

```
   $s_3$ : m();
```

```
   $s_4$ : down();
```

```
   $s_5$ :
```

```
}
```

```
var st: stack of { $s_0, \dots, s_5, \dots$ }
```

```
 $s_0 \rightarrow s_1$        $s_0 \rightarrow s_2$ 
```

```
 $s_1 \rightarrow \epsilon$ 
```

```
 $s_2 \rightarrow up_0 s_3$ 
```

```
 $s_3 \rightarrow m_0 s_4$ 
```

```
 $s_4 \rightarrow down_0 s_5$ 
```

```
 $s_5 \rightarrow \epsilon$ 
```

Probabilistic pushdown systems

A **probabilistic pushdown system** (PPDS) is a tuple $P = (P, \Gamma, \delta, \textit{Prob})$, where

- (P, Γ, δ) is a PDS, and
- $\textit{Prob}: \delta \rightarrow (0..1]$ such that for every pair pX :

$$\sum_{pX \hookrightarrow q\alpha} \textit{Prob}(pX \hookrightarrow q\alpha) = 1$$

Notation: We write $pX \xrightarrow{x} q\alpha$ for $\textit{Prob}(pX \hookrightarrow q\alpha) = x$

Semantics: A (possibly infinite) Markov chain with configurations as states and transition probabilities given by

If $pX \xrightarrow{x} q\alpha \in \delta$ then $pX\beta \xrightarrow{x} q\alpha\beta$ for every $\beta \in \Gamma^*$

Normalisation: $|\alpha| \leq 2$, termination only by empty stack.

With some more effort: $|\alpha| = 0$ or $|\alpha| = 2$

Probabilistic verification

Qualitative properties: does a program property hold with probability 1 ?

(Has the set of program runs satisfying the property measure 1 ?)

Quantitative properties: does a program property hold with probability at least ρ ?

(Is the measure of the set of program runs satisfying the property at least ρ ?)

Also properties about expectations and long-run behaviour

- Is the expected length of a terminating computation below a threshold ℓ ?
- Will the stack height exceed a threshold h at most 20% of the time ?

Probability of termination

Fundamental problem

Many other problems reducible to it (even model-checking problems)

Equivalent to the **extinction probability** in multitype branching processes, with many applications in physics and biology.

For the probability of termination, LIFO, FIFO, or parallel execution is **irrelevant**.

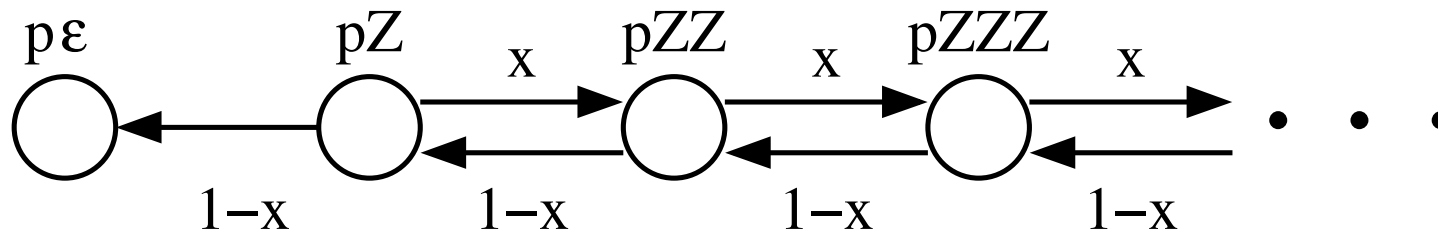
Qualitative termination: does the program terminate with probability 1?

Quantitative termination:

- Decision problem: does the program terminate with probability at least ρ ?
- Numerical problem: approximate the probability of termination within a given additive error ϵ .

A 1-state PPDS

$$\begin{aligned} pZ &\xrightarrow{x} pZZ \\ pZ &\xrightarrow{1-x} p\varepsilon \end{aligned}$$



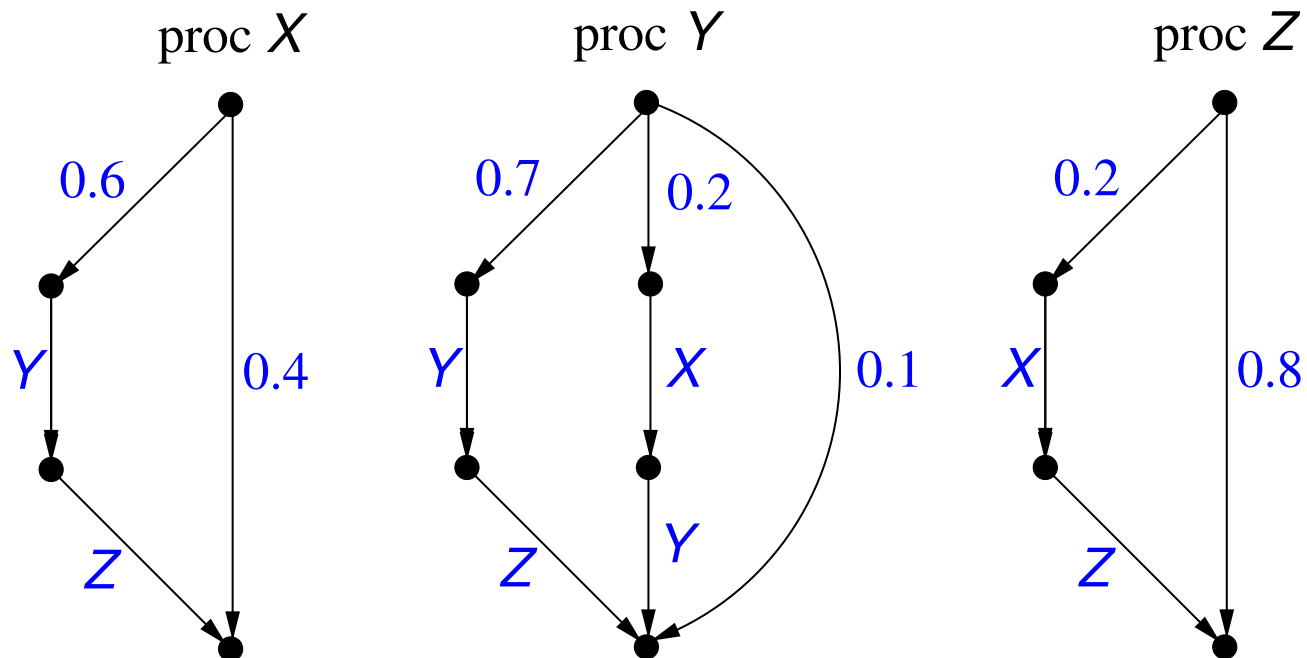
The probability of termination is 1 if $x \leq 1/2$, otherwise $\frac{1-x}{x}$.

Even qualitative termination depends on the actual values of the probabilities.

→ qualitative problems cannot be solved by graph-theoretical methods only.

Another 1-state PPDS

The PPDS for a program without global variables has 1 state.



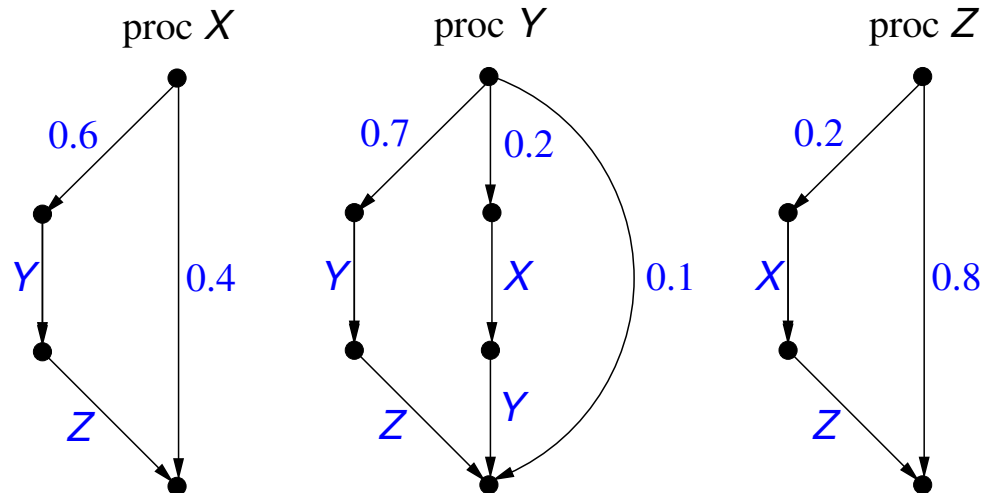
Equations for the probability of termination

Let x, y, z denote the probability of terminating (emptying the stack) from stack X, Y, Z .

$$x = 0.4 + 0.6yz$$

$$y = 0.1 + 0.2xy + 0.7yz$$

$$z = 0.8 + 0.2xz$$



Observe: $x = y = z = 1$ is solution, but not the least solution!

Observe: System of **polynomial** equations.

Prob. of termination: The one-state case

Assume: rules of the form $pX \xrightarrow{x} q\epsilon$ or $pX \xrightarrow{x} qYZ$.

Assume: the PPDS has one state p ; write $X \xrightarrow{x} \alpha$ for $pX \xrightarrow{x} p\alpha$.

We speak of the configuration (or stack) α , instead of $p\alpha$.

Define $[X]$ as the probability of, starting at stack X , eventually terminating (reaching the empty stack ϵ).

Theorem: The $[X]$'s are the least solution of the following system of equations:

$$\langle X \rangle = \sum_{X \xrightarrow{x} \epsilon} x + \sum_{X \xrightarrow{x} YZ} x \cdot \langle Y \rangle \cdot \langle Z \rangle$$

Prob. of termination: General case

Define $\langle pXq \rangle$ as the probability of, starting at the configuration pX , eventually reaching the configuration $q\epsilon$.

Theorem: The $\langle pXq \rangle$'s are the least solutions of the following system of equations:

$$\langle pXq \rangle = \sum_{pX \xrightarrow{x} q\epsilon} x + \sum_{pX \xrightarrow{x} rYZ} x \cdot \sum_{t \in P} \langle rYt \rangle \cdot \langle tZq \rangle$$

Computing the prob. of termination

The least solution of the equations can contain irrationals, or even algebraic numbers of arbitrary degree
—→ no closed-form solution.

The least solution of a system on n equations with coefficients 0 or $1/2$ can have components as small as $1/2^{2^{O(n)}}$ or as large as $1 - 1/2^{2^{O(n)}}$
—→ writing down the least solution requires exponential number of bits!

Computing the prob. of termination: Upper bounds

Theorem: The problem $[pXq] \stackrel{?}{\leq} \rho$ can be solved in PSPACE for every $0 \leq \rho \leq 1$

Reduction to the decision problem for the **existential first-order theory of the reals**

Theory of the reals: first-order theory of $(\mathbb{R}, <, +, *)$

Existential first-order theory of the reals: fragment of the form

$$\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$$

Expressing that a solution is the **least** solution requires quantifier alternation.

Avoidable by showing that another system of the same size, computable in poly-space, has only one solution, which is the least solution of the original system.

Some recent work on this area (e.g., [Jovanovic, De Moura 2013]), but only practical for problems with a few variables.

Computing the prob. of termination: Lower bounds

Theorem: The SQUARE-ROOT-SUM problem is polynomially reducible to the problem $[pXq] \stackrel{?}{=} 1$ (or even to the problem of approximating $[pXq]$ within any additive error $< 1/2$).

Given: $(d_1, \dots, d_n) \in \mathbb{N}^n$ and $k \in \mathbb{N}$

Decide whether: $\sum_{i=1}^n \sqrt{d_i} \leq k$

In PSPACE

Not known to be in NP.

It lies in the 4th level of the Counting Hierarchy [Allender et al. 06].

Computing the prob. of termination: Numerical methods

Our systems of equations are **monotone polynomial systems** (MPS).

We use from now on the notation $X = f(X)$ (where X vector).

Theorem: [Tarski '55, Kleene '52] The least solution of a MPS $X = f(X)$, denoted by μf , always exists and is equal to the supremum of the **Kleene approximants** $\{\kappa_i\}_{i \geq 0}$ given by

$$\begin{aligned}\kappa_0 &= f(0) \\ \kappa_{i+1} &= f(\kappa_i) .\end{aligned}$$

Basic algorithm for calculation of μf : Compute $\kappa_0, \kappa_1, \kappa_2, \dots$ until either $\kappa_i = \kappa_{i+1}$ or the approximation is considered adequate.

Kleene iteration may be (very) slow

$$x = 0.5 x^2 + 0.5 \quad \mu f = 1 = 0.99999 \dots$$

“Logarithmic convergence”: n iterations give $O(\log n)$ correct digits.

$$\kappa_n \leq 1 - \frac{1}{n+1} \quad \kappa_{2000} = 0.9990$$

Kleene iteration may be (very) slow

$$x = 0.5 x^2 + 0.5 \quad \mu f = 1 = 0.99999 \dots$$

“Logarithmic convergence”: n iterations give $O(\log n)$ correct digits.

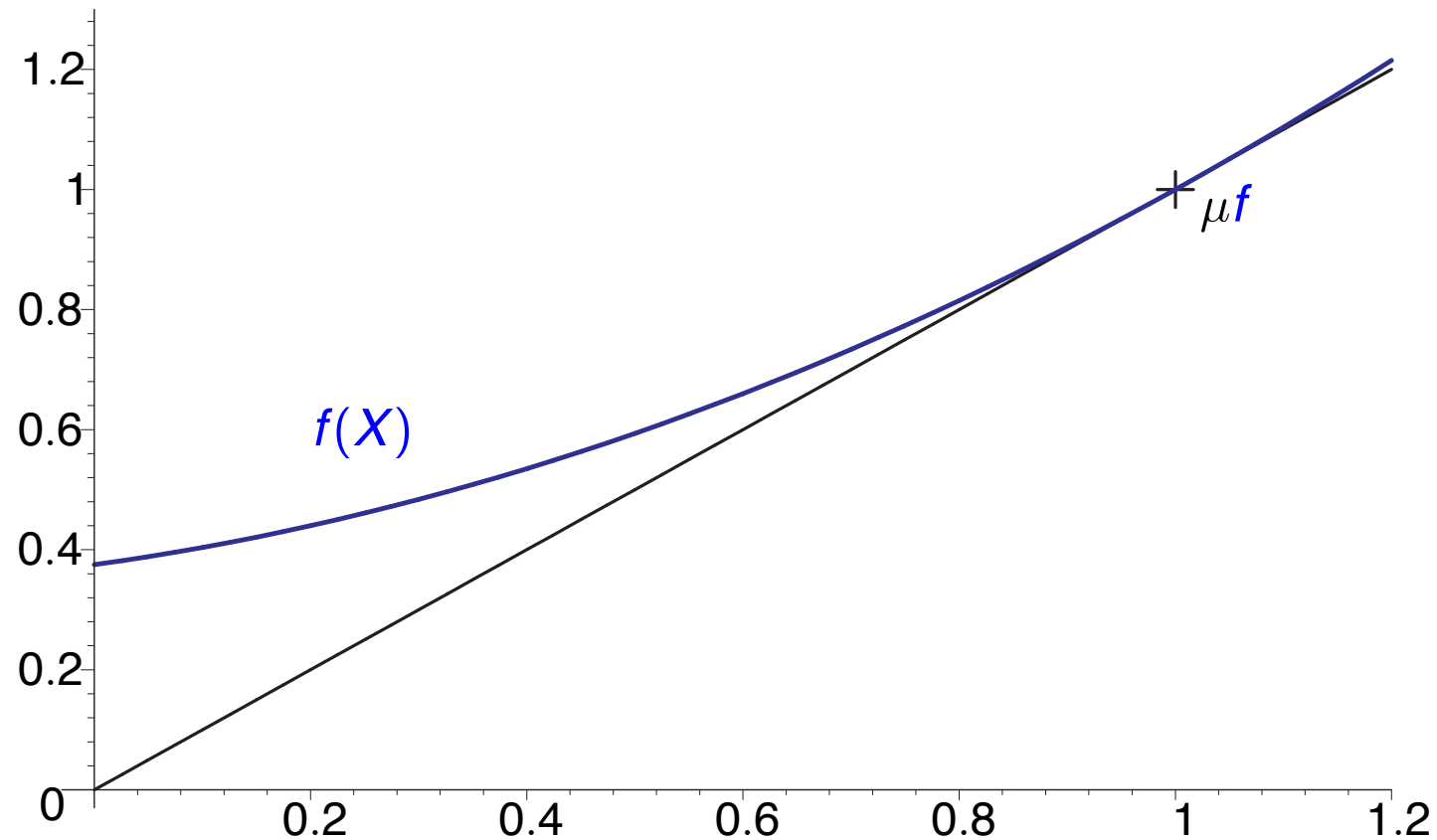
$$\kappa_n \leq 1 - \frac{1}{n+1} \quad \kappa_{2000} = 0.9990$$

Better idea:
Newton's method

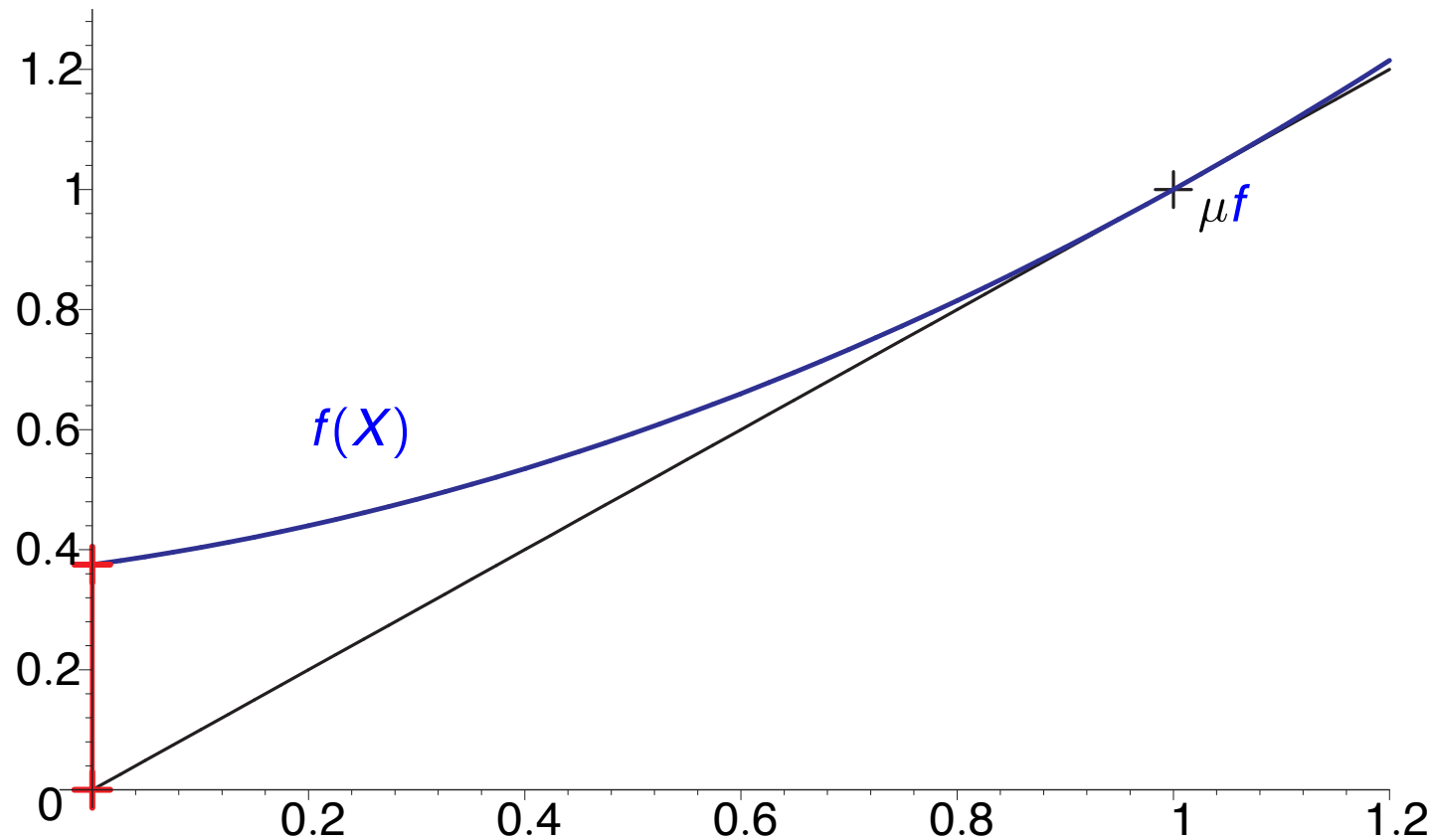


© Martiarena

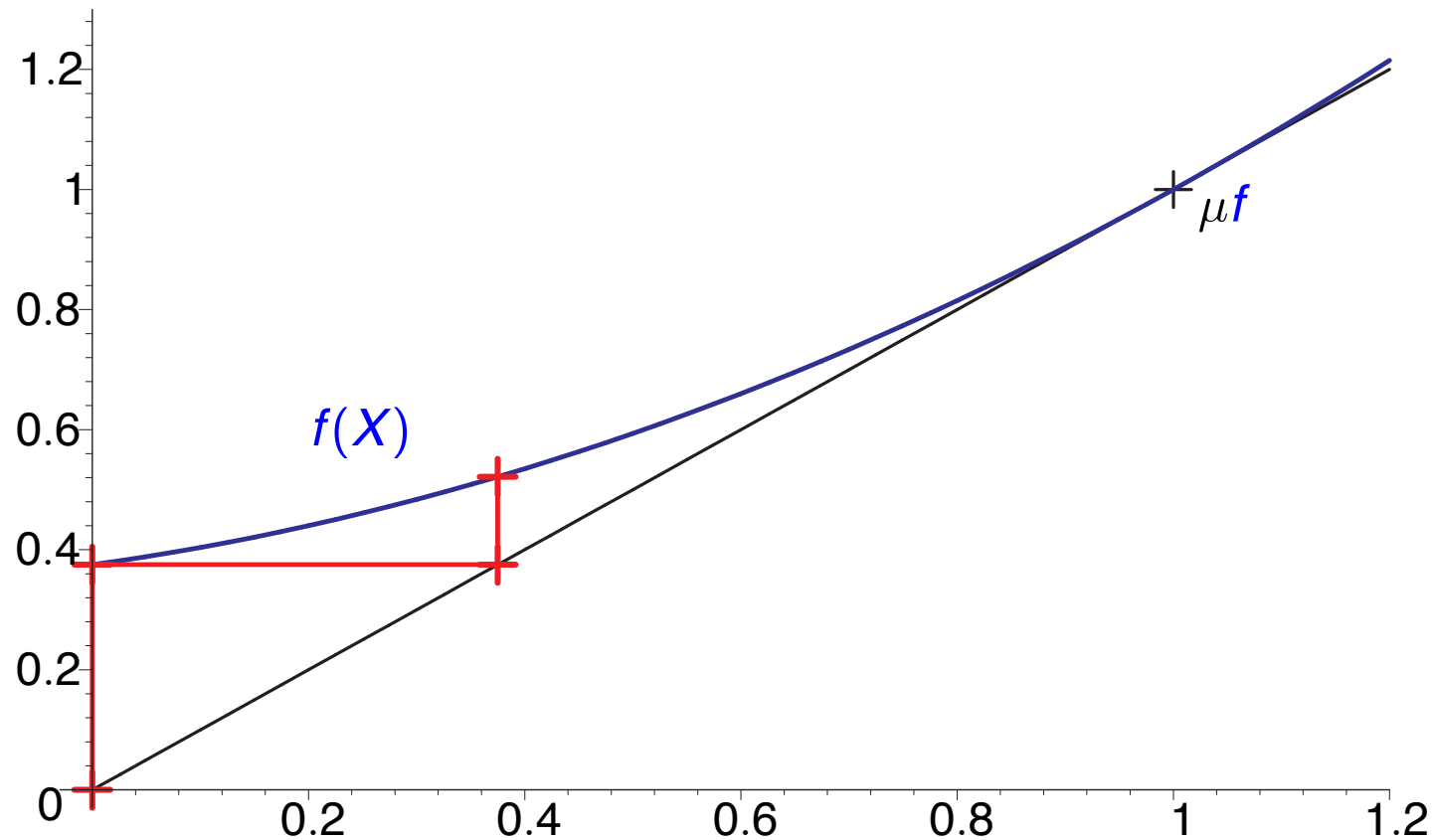
Kleene Iteration for $X = f(X)$ (univariate case)



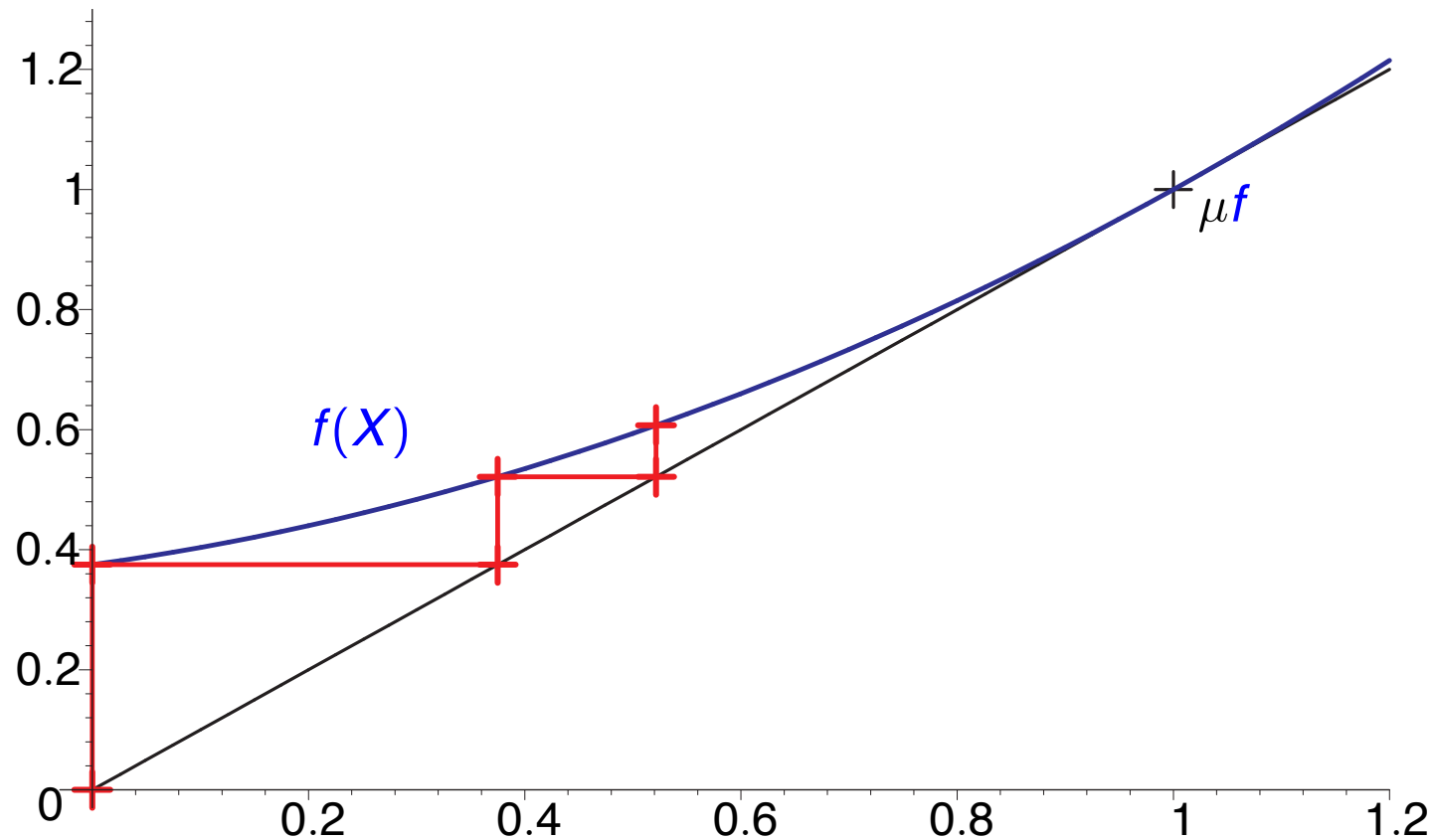
Kleene Iteration for $X = f(X)$ (univariate case)



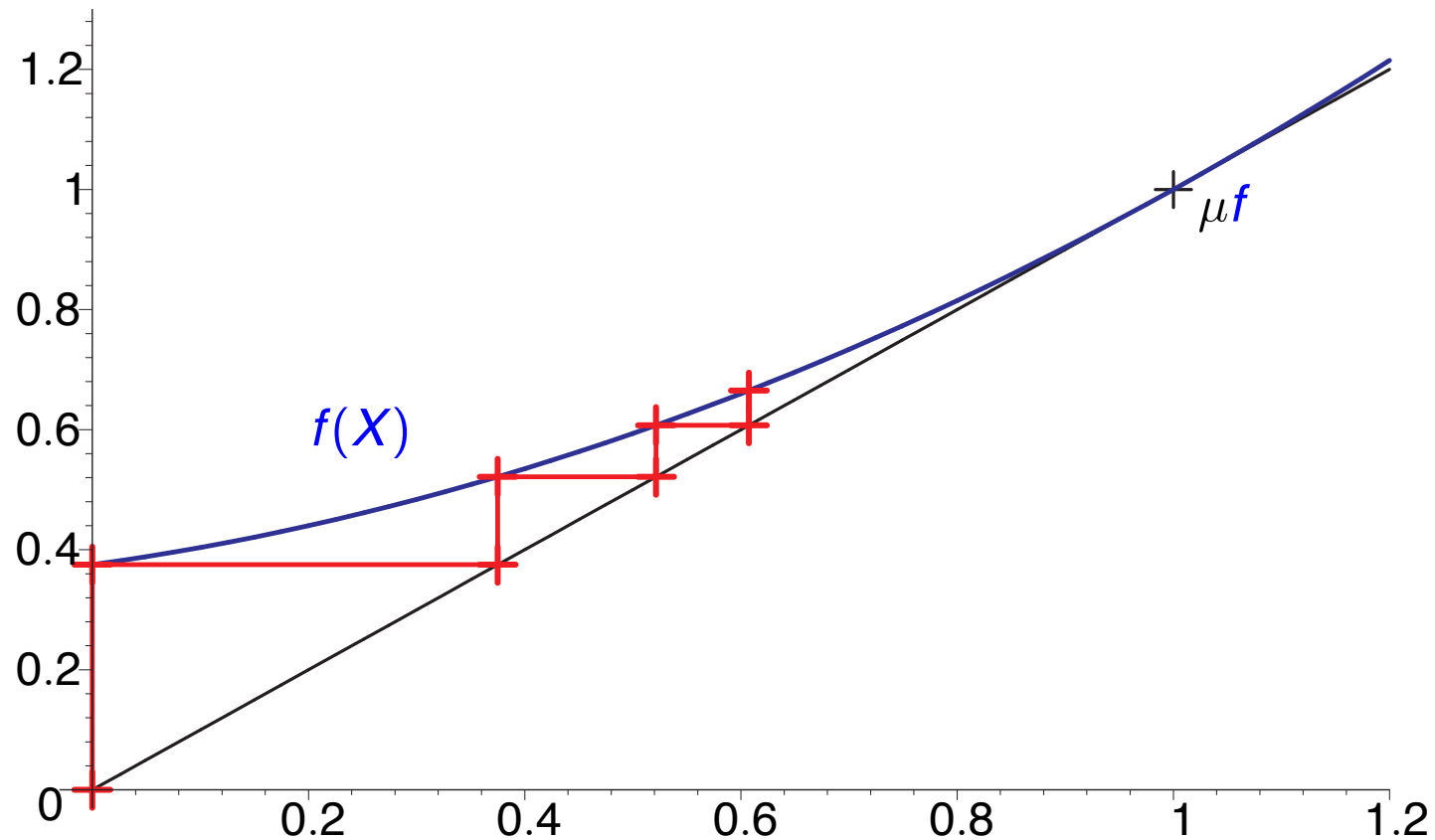
Kleene Iteration for $X = f(X)$ (univariate case)



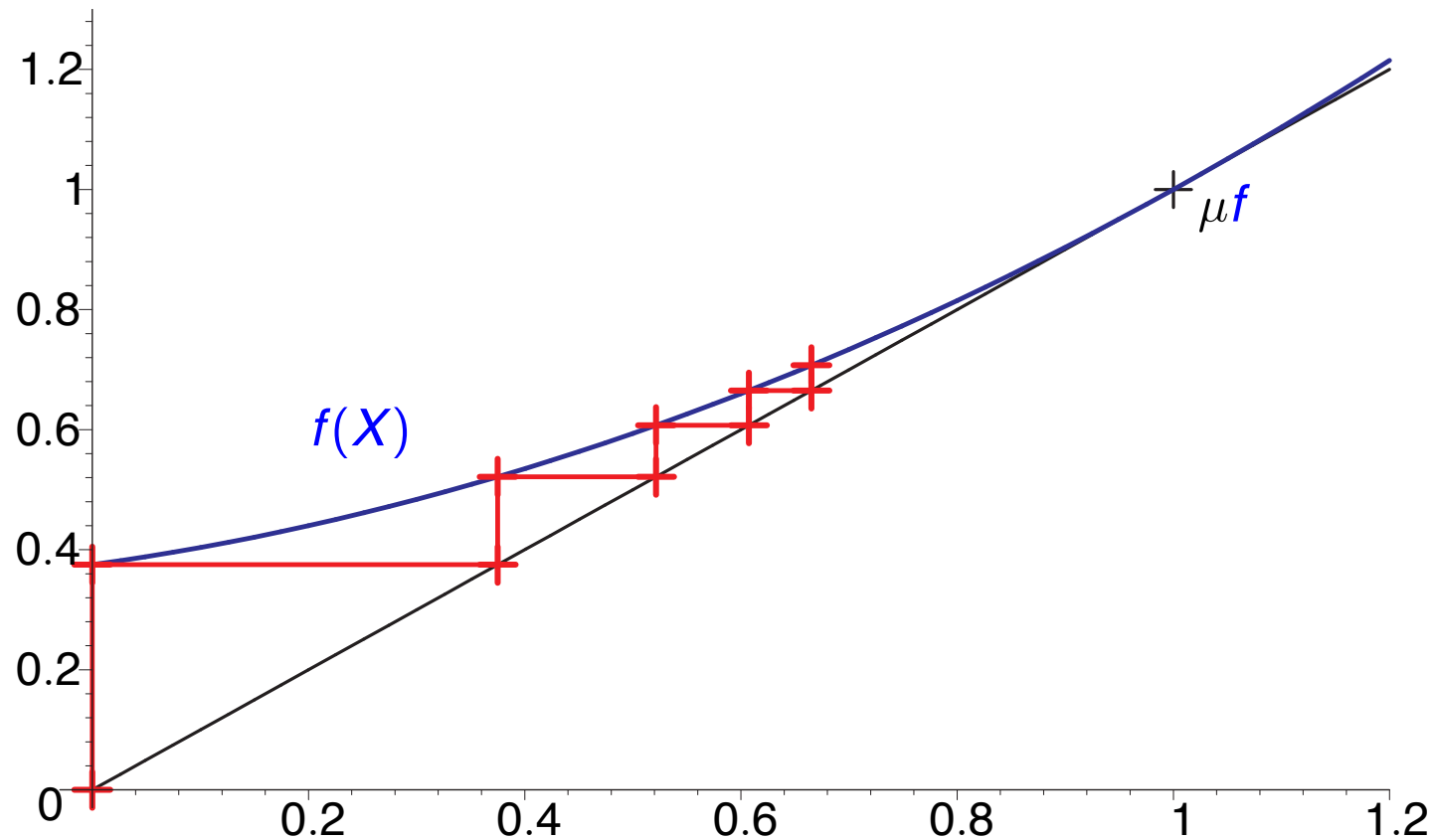
Kleene Iteration for $X = f(X)$ (univariate case)



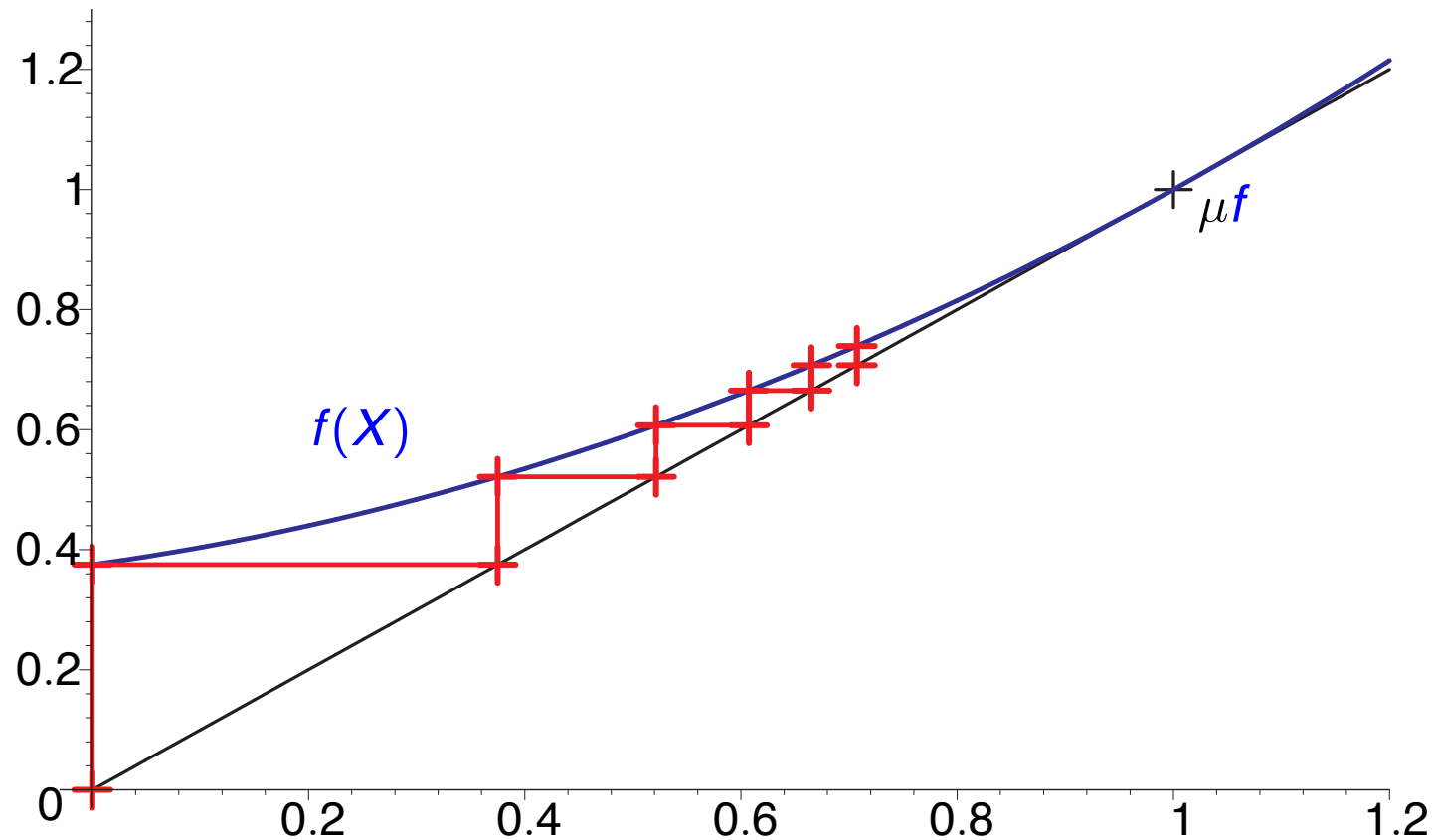
Kleene Iteration for $X = f(X)$ (univariate case)



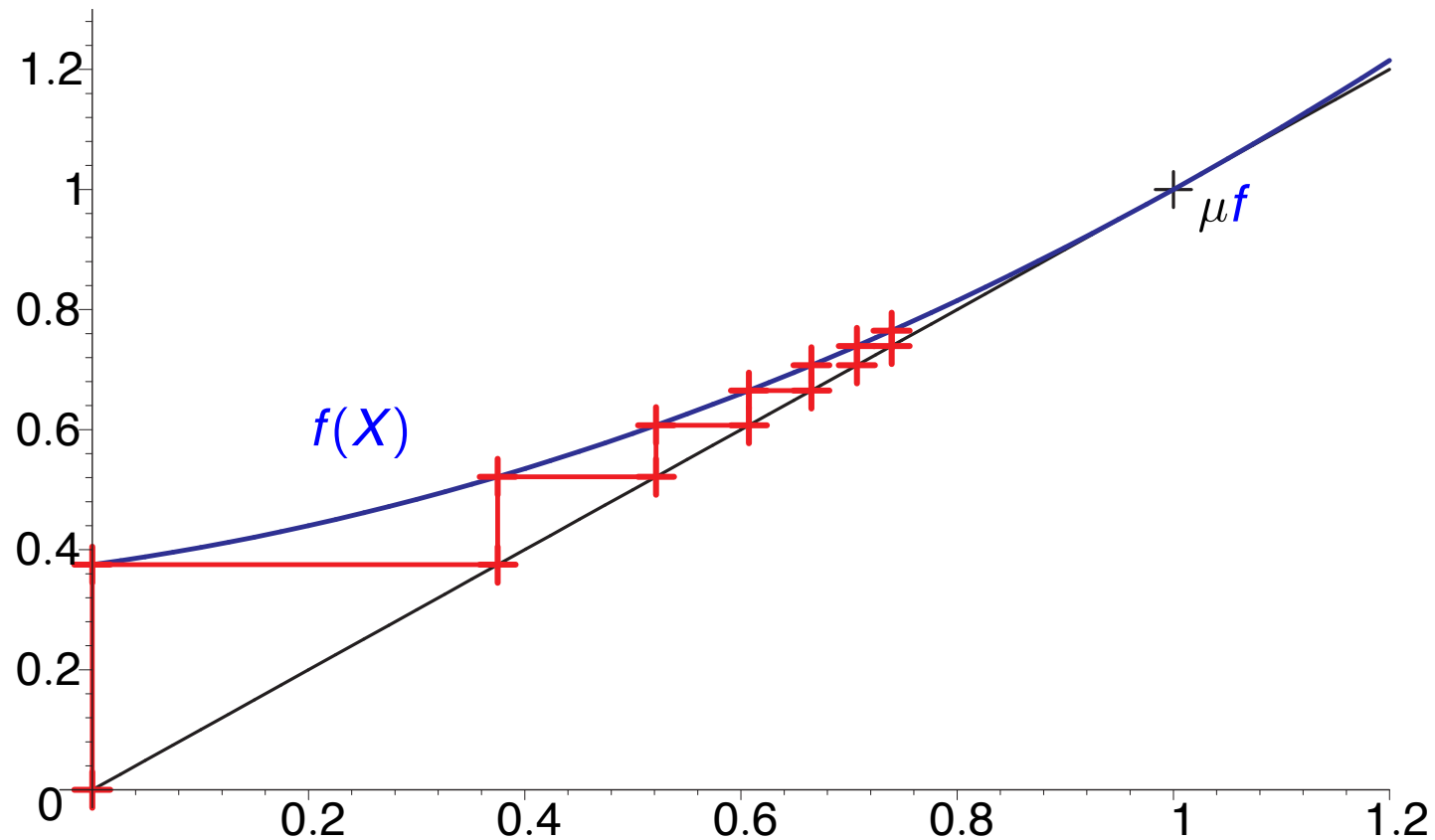
Kleene Iteration for $X = f(X)$ (univariate case)



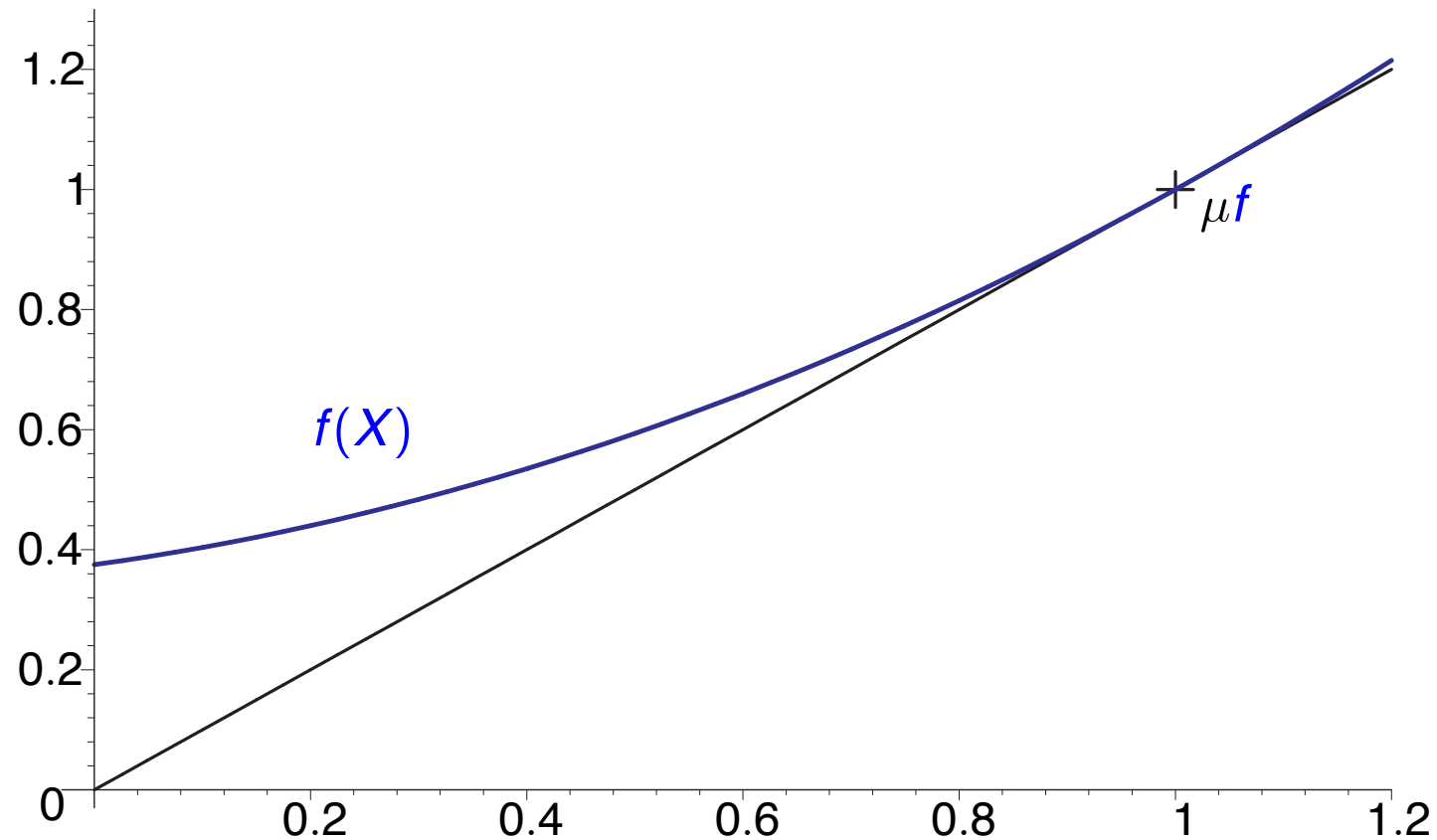
Kleene Iteration for $X = f(X)$ (univariate case)



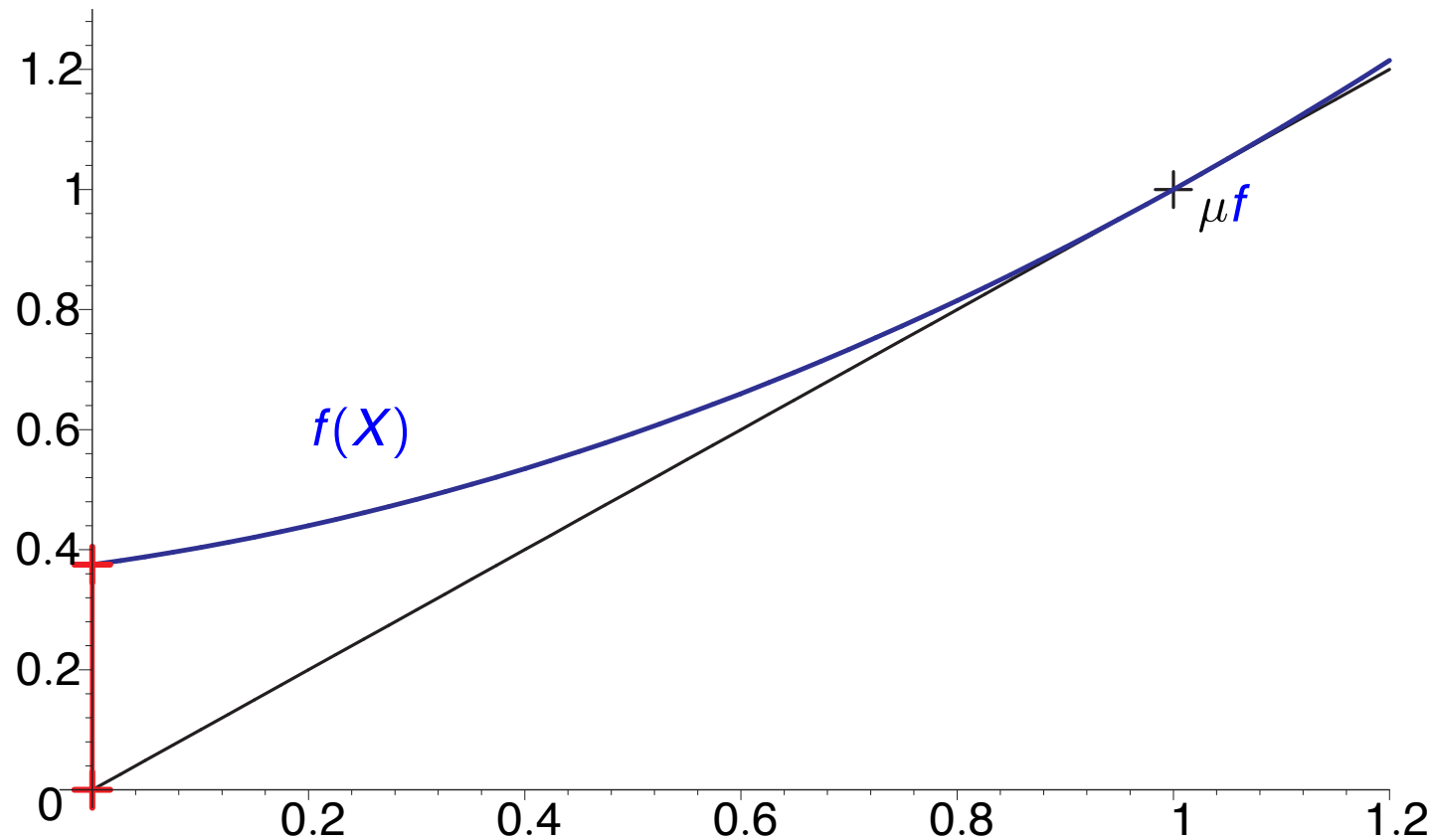
Kleene Iteration for $X = f(X)$ (univariate case)



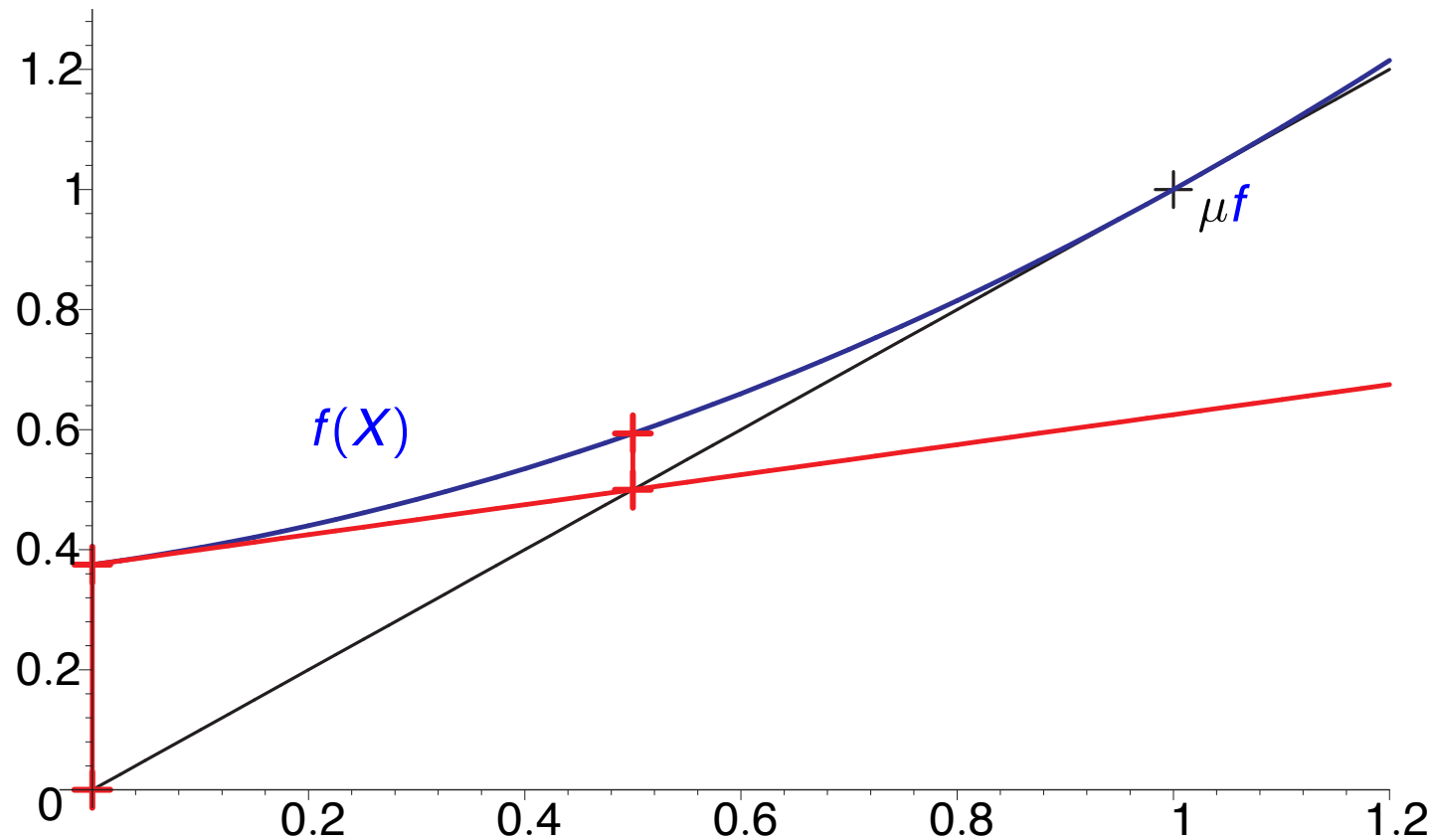
Newton's Method for $X = f(X)$ (univariate case)



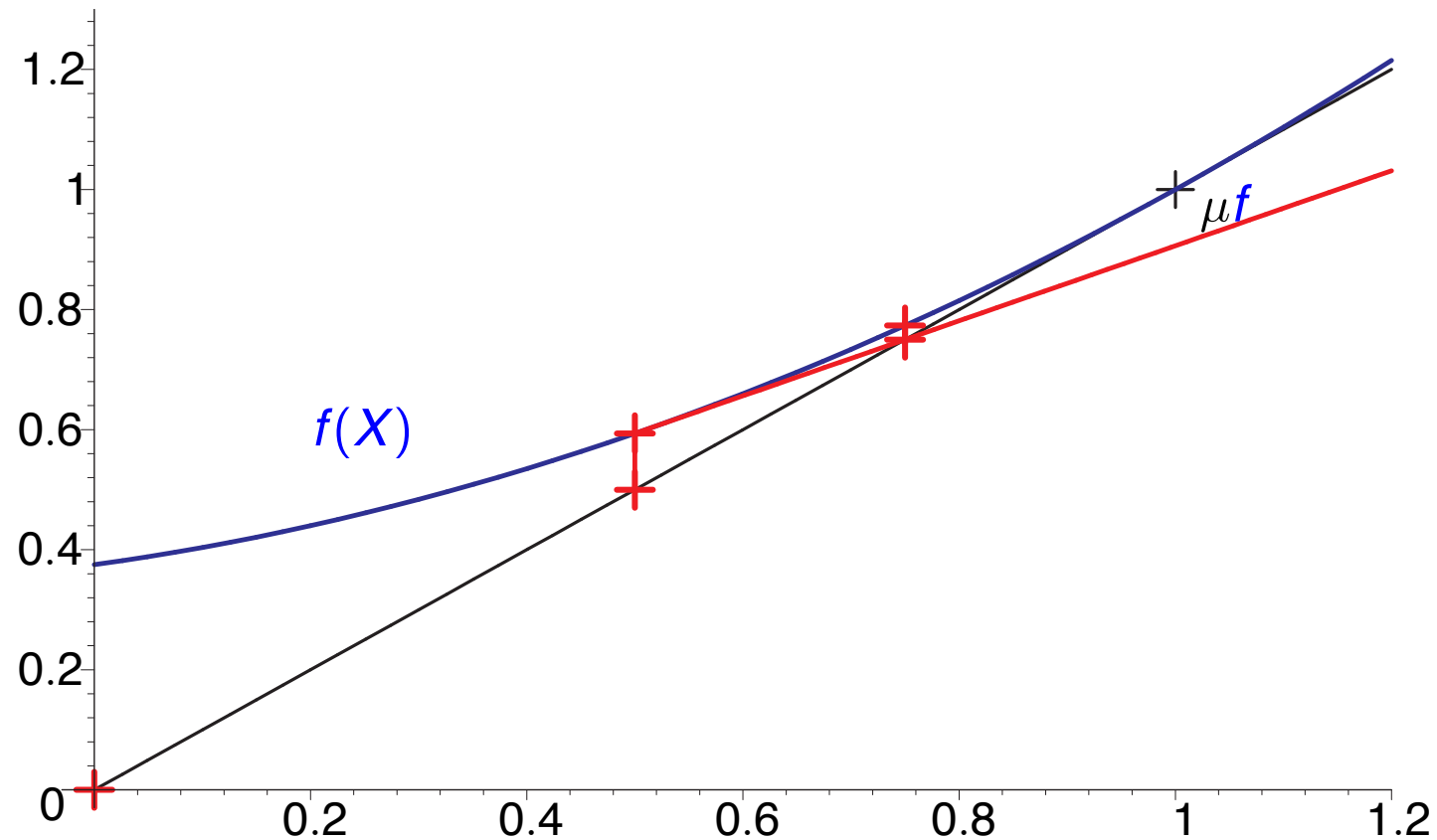
Newton's Method for $X = f(X)$ (univariate case)



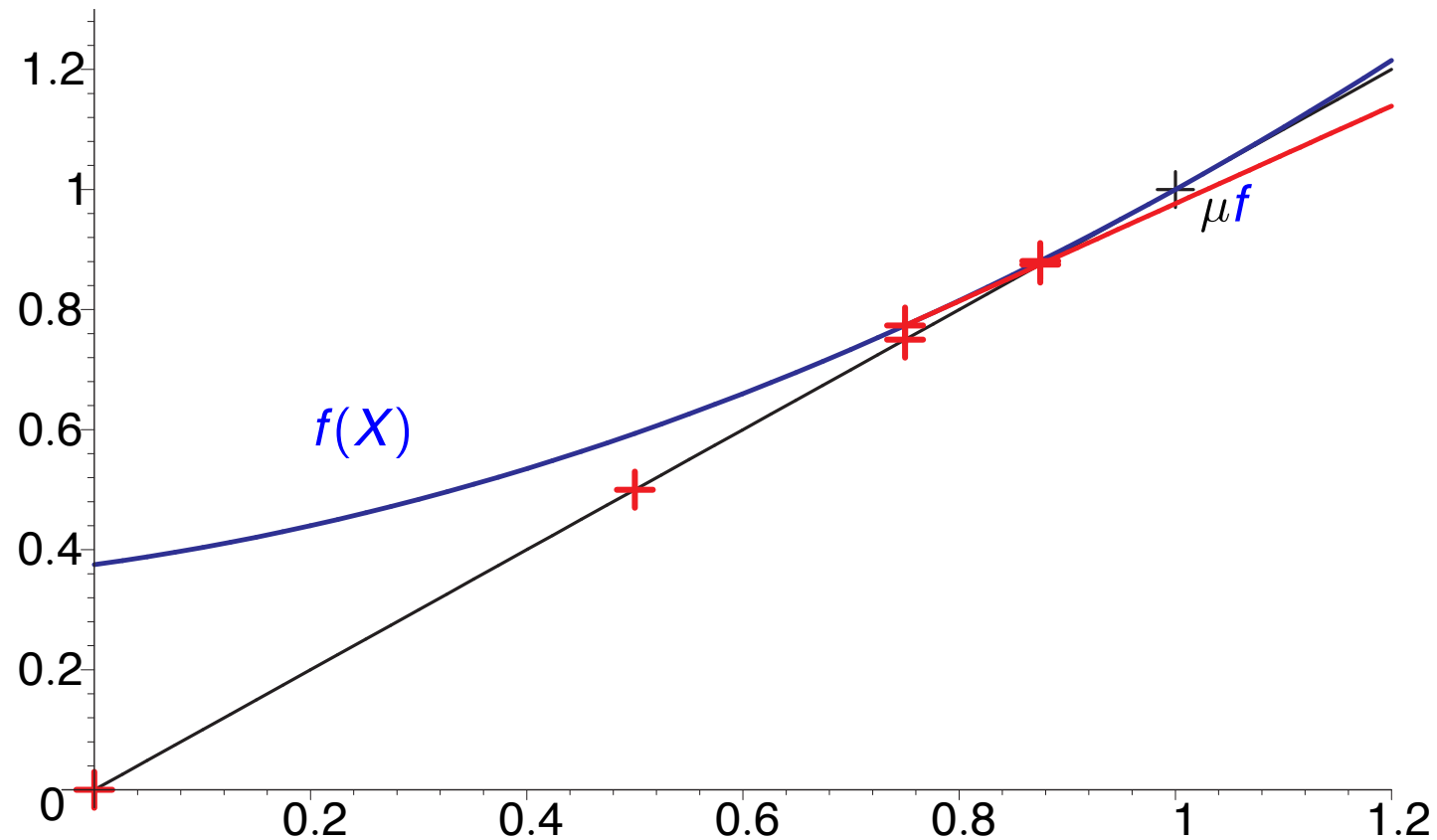
Newton's Method for $X = f(X)$ (univariate case)



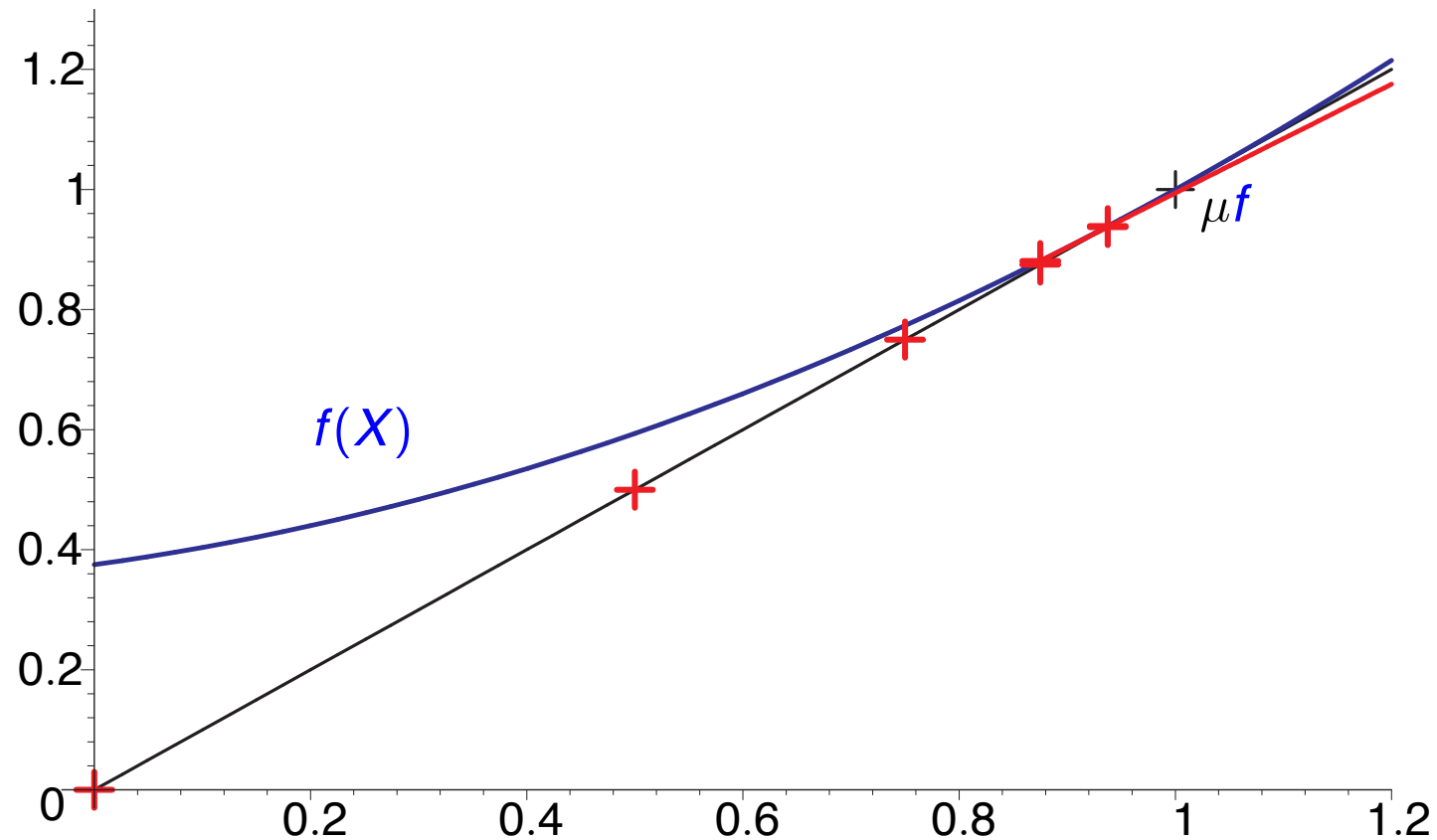
Newton's Method for $X = f(X)$ (univariate case)



Newton's Method for $X = f(X)$ (univariate case)



Newton's Method for $X = f(X)$ (univariate case)



Definition of Newton's method: Univariate case

Let $x = f(x)$ be a monotonic polynomial equation.

The sequence of **Newton approximants** is given by

$$\nu_0 := \text{seed}$$

$$\nu_{i+1} := \nu_i + \frac{f(\nu_i) - \nu_i}{1 - f'(\nu_i)}$$

Definition of Newton's method: Multivariate case

Let $X = f(X)$ be an MPS.

Let $J(X)$ be the Jacobi matrix

$$J_{ij} = \frac{\partial f_i}{\partial X_j}$$

The sequence of **Newton approximants** is given by

$$\begin{aligned}\nu_0 &:= \text{seed} \quad (\text{we shall use seed} = 0) \\ \nu_{i+1} &:= \nu_i + (Id - J)^{-1}(\nu_i) \cdot (f(\nu_i) - \nu_i)\end{aligned}$$

(compare with $\nu_{i+1} := \nu_i + (1 - f'(\nu_i))^{-1}(f(\nu_i) - \nu_i)$)

Newton's method: Runtime

$(Id - J)^{-1}(\nu_i)$ can be computed by solving a system of linear equations.
(Approximate solution is possible.)

For an MPS with n variables, computing ν_{i+1} from ν_i requires $O(n^3)$ arithmetic operations.

Unit-cost model: Count the number of operations (1 time unit per operation, independently of the size of the operands)

Turing model: Time for an operation depends on size of operands.

Polynomial time in the unit-cost model does not imply polynomial time in the Turing model !

Newton's method: Convergence speed

Newton's method "often" has **exponential convergence** (called **quadratic convergence** in math): The number of accurate bits grows exponentially with the number of iterations.

However, for general equations $g(X) = 0$, the method may

- not converge,
- converge only in a small ball around the zero, or
- converge slowly.

A hard case for Newton's method

$$\begin{aligned}X_1 &= \frac{1}{2}X_1^2 + \frac{1}{2} \\X_2 &= \frac{1}{4}X_1^2 + \frac{1}{2}X_1X_2 + \frac{1}{4}X_2^2 \\&\dots \\X_n &= \frac{1}{4}X_{n-1}^2 + \frac{1}{2}X_{n-1}X_n + \frac{1}{4}X_n^2\end{aligned}$$

We have $\mu f = 1$

After 2^{n-1} iterations of Newton's method we still have $\nu_{2^{i-1}}(X_n) < 1/2$!

So: faster algorithms require clever preprocessing, or special cases.

Convergence speed: The 1-state case

Programs without global variables

Theorem: [Etessami, Stewart, Yannakakis '12] (E., Kiefer, Luttenberger '10)

Let $X = f(X)$ be the MPS for a 1-state PPDS in "normal form".

For each variable X_i can decide in (strongly) polynomial time which of

$$\mu f(X_i) = 0 \quad 0 < \mu f(X_i) < 1 \quad \mu f(X_i) = 1$$

holds.

Assume $0 < \mu f < 1$. Let $\{\nu_i\}_{i \geq 0}$ be the Newton approximants with $\nu_0 := 0$, and let $K = 32|f| + 2$.

For every $i \in \mathbb{N}$ we have:

$$\| \mu f - \nu_{(K+2i)} \|_{\infty} \leq \frac{1}{2^{2i}}$$

Convergence speed: The 1-state case

Therefore: the first 2^i bits of the solution can be obtained by computing the first $K + 2i$ approximants.

It follows: polynomial time in the unit-cost model.

Rounded-down Newton's method with parameter h : After each iteration round keep only the first h bits of the fractional part.

Theorem: [Etessami, Stewart, Yannakakis '12]

Let $h = i + 2 + 4|f|$. For every $i \in \mathbb{N}$ we have:

$$\| \mu f - \nu_h \|_{\infty} \leq \frac{1}{2^i}$$

So we can compute i bits of μf in polynomial time in i and $|f|$ in the Turing model of computation.

Convergence speed: The 1-stack-symbol case

Programs with one single (possibly recursive) procedure.

Theorem: [Etessami, Wojtczak, Yannakakis '10]

Let $X = f(X)$ be the MPS for a 1-stack symbol PPDS.

We can compute i bits of μf in polynomial time in i and $|f|$ in the Turing model of computation.

The degree of the polynomial is considerably worse than in the 1-state case.

Convergence speed: The general case

Theorem: [Etessami, Stewart, Yannakakis '16]

Let $X = f(X)$ be a MPS for a general PPDS.

We can compute the first i bits of μf in a number of iterations polynomial in i but exponential in the height of the SCC graph of the MPS.

Using Rounded-down Newton's method we obtain the same result in the Turing model of computation.

Rounding down in practice

The rounding-down parameters obtained in theory are not realistic.

[E., Gaiser, Kiefer '10] presents an adaptive technique that adds precision only on demand.

Given an error bound $\epsilon > 0$, the technique computes an interval $[\text{lb}, \text{ub}]$ such that $\text{lb} \leq \mu_f \leq \text{ub}$.

Lower bound easy, upper bound difficult.

The algorithm uses instructions of the form

$$x := F(x, y) \text{ such that } P(x)$$

The operation $F(x, y)$ is repeated with increasing accuracy until $P(x)$ holds.

Rounding down in practice

Input: MPS for 1-state PPDS f in normal form, error bound $\epsilon > 0$

Output: vectors \mathbf{lb}, \mathbf{ub} such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \epsilon$

$\mathbf{lb} \leftarrow \text{computeStrictPrefix}(f); \mathbf{ub} \leftarrow \mathbf{1};$

while $\mathbf{ub} - \mathbf{lb} \not\leq \epsilon$ **do**

$\mathbf{x} \leftarrow \mathcal{N}(\mathcal{N}(\mathbf{lb}))$ such that $f(\mathbf{lb}) + f'(\mathbf{lb})(\mathbf{x} - \mathbf{lb}) \prec \mathbf{x} \prec f(\mathbf{x}) \prec \mathbf{1};$

$\mathbf{lb} \leftarrow \mathbf{x};$

$Z \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{ub}) = 1\}; P \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{ub}) < 1\};$

$\mathbf{y}_Z \leftarrow \mathbf{1};$

$\mathbf{y}_P \leftarrow f_P(f(\mathbf{ub}))$ such that $f_P(\mathbf{y}) \prec \mathbf{y}_P \prec f_P(\mathbf{ub});$

forall superlinear SCCs S of f with $\mathbf{y}_S = \mathbf{1}$ **do**

$\mathbf{t} \leftarrow (1 - \mathbf{lb}_S);$

if $f'_S(1)\mathbf{t} \succ \mathbf{t}$ **then**

$\mathbf{y}_S \leftarrow \left(1 - \min \left\{1, \frac{\min_{i \in S}(f'_S(1)\mathbf{t} - \mathbf{t})_i}{2 \cdot \max_{i \in S}(f_S(2))_i}\right\} \cdot \mathbf{t}\right)$ such that $f_S(\mathbf{y}) \prec \mathbf{y}_S \prec \mathbf{1};$

$\mathbf{ub} \leftarrow \mathbf{y}_S$

Nuclear chain reaction [Harris '63]

^{235}U ball of radius D , spontaneous fission.

Probability of a chain reaction is $(1 - p_0)$,

where p_α for $0 \leq \alpha \leq D$ is least solution of

$$p_\alpha = k_\alpha + \int_0^D R_{\alpha,\beta} f(p_\beta) d\beta$$

for constants k_α , $R_{\alpha,\beta}$ and polynomial f .

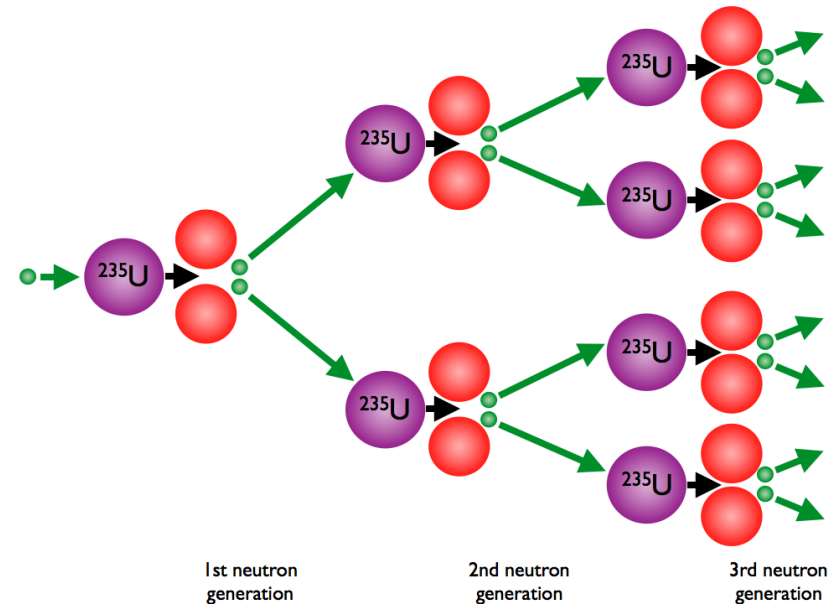
Discretizing $[0, D]$ we get the PPS

$$p_0 = k_0 + \sum_{j=1}^n r_{0,j} f(p_j)$$

...

$$p_n = k_n + \sum_{j=1}^n r_{n,j} f(p_j)$$

for constants k_i , $r_{i,j}$.



Nuclear chain reaction: Some experiments

Runtime in seconds on a standard laptop of various algorithms on different values of D and $n = 100$.

D	2	3	6	10
$p_0 = 1?$ (BOOM?)	n	y	y	y
Runtime of our alg.	2	2	2	2
Runtime of exact LP	258	124	168	222
Runtime of our alg. for $\epsilon = 10^{-3}$	4	32	21	17

Reachability

Compute the probability of reaching a configuration of $\mathcal{C} = \{p_0 X_0 w \mid w \in \Gamma^*\}$.

Define $[pX\bullet]$ as the probability of, starting at pX , reaching \mathcal{C} .

Define $[pXq]$ as the probability of, starting at pX , eventually reaching $q\epsilon$ without visiting \mathcal{C} .

Theorem: The $[pX\bullet]$'s and $[pXq]$'s are the least solution of the following system of equations:

$$\langle pXq \rangle = \sum_{pX \xrightarrow{x} q\epsilon} x + \sum_{pX \xrightarrow{x} rYZ} x \cdot \sum_{t \in P} \langle rYt \rangle \cdot \langle tZq \rangle$$

$$\langle pX\bullet \rangle = \sum_{pX \xrightarrow{x} rYZ} x \cdot (\langle rY\bullet \rangle + \sum_{t \in P} \langle rYt \rangle \cdot \langle tZ\bullet \rangle)$$

Computing expected rewards

Reward functions assign a reward to every configuration.

- **Simple** functions: reward depends only on control state and top stack symbol.
- **Linear** functions: reward is a linear function of the control state, the top stack symbol, and the number of occurrences of each symbol in the stack.

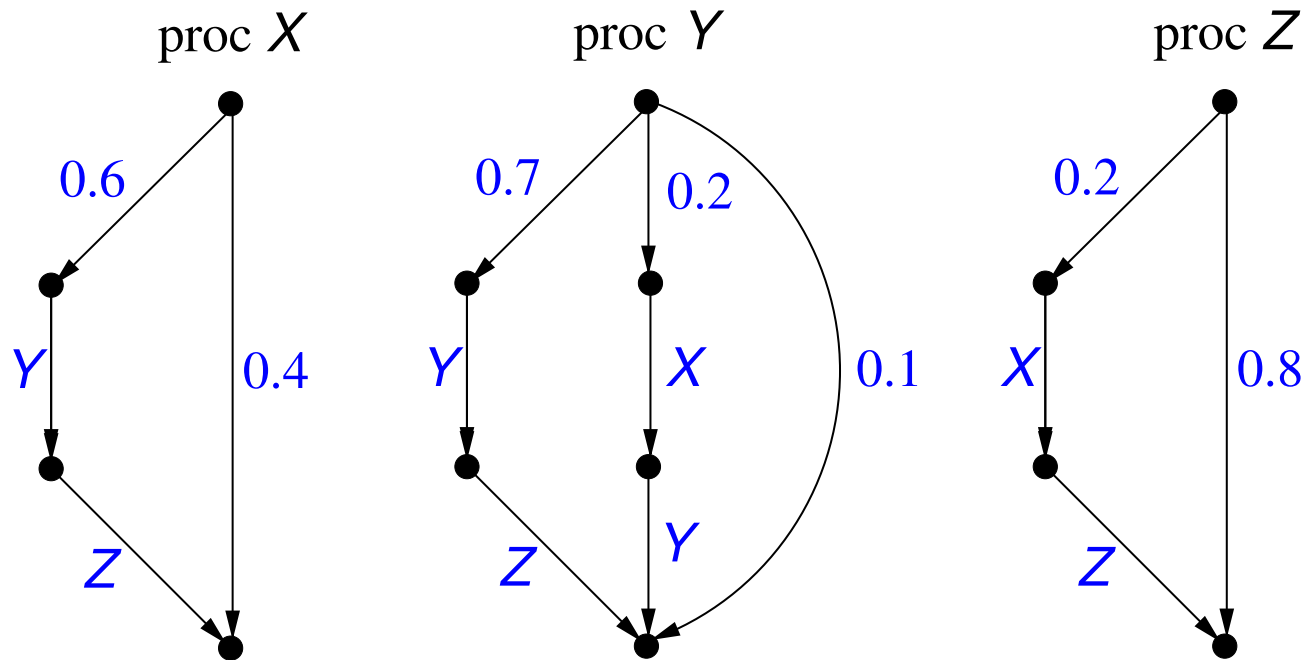
Total reward of an execution: sum of the rewards of the configurations it visits.

Mean payoff of an infinite execution: limit of the average reward of the prefixes.

Conditional expected termination time: expected reward for the simple function that assigns 1 to each configuration, subject to the condition that the execution terminates.

Average stack height: expected reward for the linear function that adds the number of occurrences of all stack symbols.

The 1-state PPDS again

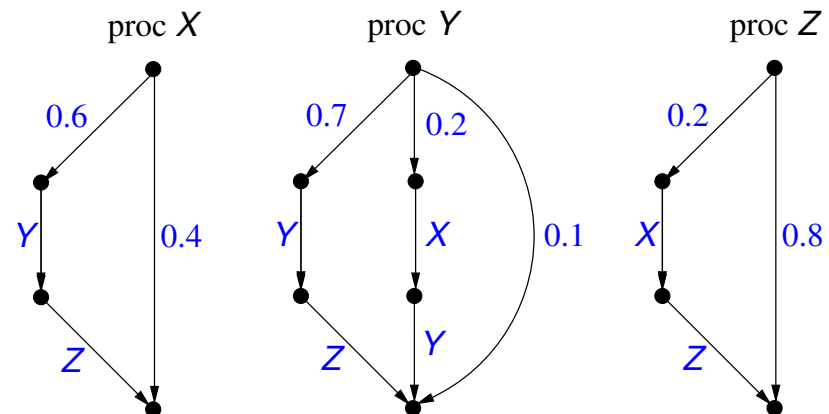


Equations for the expected time to termination

Let E_x, E_y, E_z denote the expected termination time from stack X, Y, Z .

Let t_x, t_y, t_z denote the termination probability from X, Y, Z .

$$\begin{aligned}E_x &= 0.4 + 0.6t_yt_z(1 + E_y + E_z) \\E_y &= 0.1 + 0.2t_xt_y + 0.7t_yt_z(1 + E_y + E_z) \\E_z &= 0.8 + 0.2t_xt_z(1 + E_x + E_z)\end{aligned}$$



Observe: The equations are **linear**.

Conditional expected time to termination: General case

Define $[pXq]$ as the probability of, starting at the configuration pX , eventually reaching the configuration $q\epsilon$.

Define E_{pXq} as the expected time of going from pX to $q\epsilon$.

Theorem: The $[pXq]$'s are the least solutions of the following system of equations:

$$\langle E_{pXq} \rangle = \sum_{pX \xrightarrow{x} q\epsilon} x + \sum_{pX \xrightarrow{x} rYZ} x \cdot \left(1 + \sum_{t \in P} \langle rYt \rangle \cdot \langle tZq \rangle (\langle E_{rYt} \rangle + \langle E_{tZq} \rangle) \right)$$

Checking Büchi specifications (qualitative)

Reducible to the following problem:

- Given: an initial configuration p_0X_0 , a control state p_r
- Decide whether: p_r is repeatedly reached w.p.1, i.e.
whether the runs that visit infinitely many configurations of the form $p_r\alpha$
have measure 1

We construct a **finite** Markov chain M with initial state s_0 s.t.

p_r is repeatedly reached from p_0X_0 w.p.1

\implies

certain states of M are repeatedly reached from s_0 w.p.1

which can be decided using graph-theoretical methods.

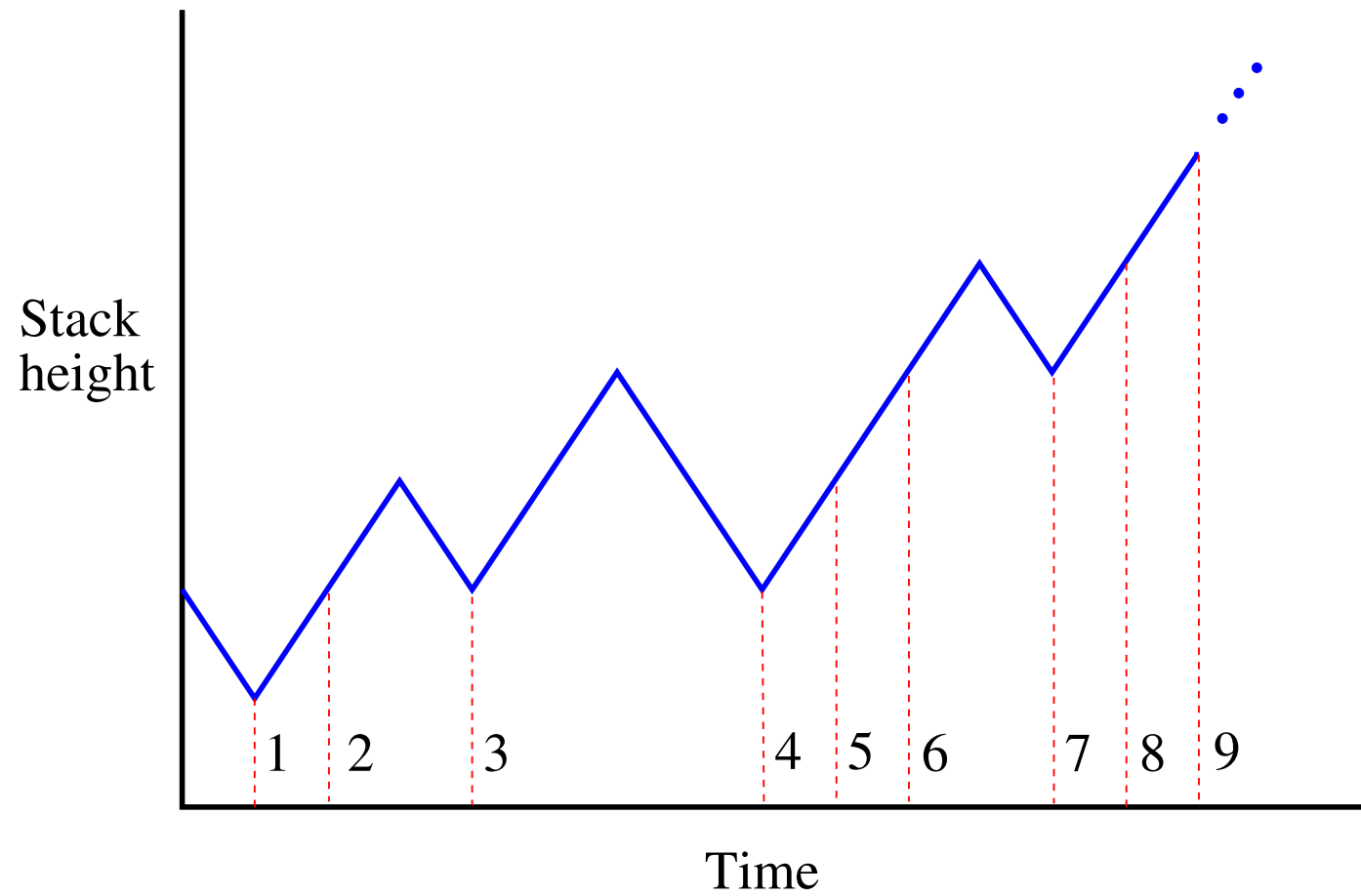
Minima of an infinite run

Let $w = p_0\alpha_0 p_1\alpha_1 p_2\alpha_2 \dots$ be an infinite run of a PPDS

$p_i\alpha_i$ is a minimum of w if $|\alpha_i| \geq |\alpha_j|$ for all $j \geq i$. (α_i “stays forever in the stack”)

Extract from w the subsequence $p_{m_1}\alpha_{m_1} p_{m_2}\alpha_{m_2} \dots$ of minima

The i -th minimum of w is the i -th configuration of the subsequence of minima.



The memoryless property

Given a configuration $c = pX\alpha$, let pX be the **head** and α the **tail** of c

Theorem [EKM04] (loosely formulated):

For every $i \geq 1$, the probability that the $i + 1$ -th minimum of a run has head pX depends only on the head of the i -th minimum (and is in particular independent of i).

We construct a Markov chain with

- the possible heads as states,
- transition probabilities given by:

$pX \xrightarrow{x} qY$ if x is the probability of, starting from a minimum with head pX , reaching the next minimum at a configuration with head qY .

Computing the transition probabilities

Let $[pX \Rightarrow qY]$ be the probability of $pX \longrightarrow qY$ in the new Markov chain.

Let $pX \uparrow$ be the probability of not terminating from pX .

We have:

$$[pX \Rightarrow qY] = \sum_{pX \xrightarrow{x} rZY} x \cdot [rZq] \cdot [qY \uparrow] + \sum_{pX \xrightarrow{x} rYZ} x \cdot [rY \uparrow]$$

Results on checking Büchi specifications

Theorem: [E., Kučera, Mayr '04]

The (qualitative and quantitative) repeated reachability problem can be solved in PSPACE.

Theorem: [Etessami, Yannakakis '05]

The qualitative (quantitative) model-checking problem for Büchi specifications can be solved in PSPACE in the size of the PPDS and in EXPTIME (EXPSPACE) in the size of the Büchi automaton.

Same “numerical complexity” as the reachability problem

Checking PCTL specifications

Syntax:

$$\varphi ::= \text{tt} \mid \mathbf{a} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}^{\geq \varrho}\varphi \mid \varphi_1 \mathbf{U}^{\geq \varrho}\varphi_2$$

where \mathbf{a} is an atomic proposition and ϱ is a probability

Semantics: Let $\llbracket \varphi \rrbracket$ denote the set of states satisfying φ

$$\begin{aligned}\llbracket \mathbf{X}^{\geq \varrho}\varphi \rrbracket &= \{p_\alpha \mid \mathcal{P}(\text{Run}(p_\alpha, \mathbf{X}\llbracket \varphi \rrbracket)) \geq \varrho \} \\ \llbracket \varphi_1 \mathbf{U}^{\geq \varrho}\varphi_2 \rrbracket &= \{p_\alpha \mid \mathcal{P}(\text{Run}(p_\alpha, \llbracket \varphi_1 \rrbracket \mathbf{U} \llbracket \varphi_2 \rrbracket)) \geq \varrho \} \end{aligned}$$

Qualitative fragment of PCTL: $\varrho = 1$

Model checking the qualitative fragment

In the finite-state case:

- Compute the set of states satisfying the subformulas of a formula
- Derive from them the set of states satisfying the formula

Problem: The set of states satisfying the subformulas can be **infinite**

Theorem: [E., Kučera, Mayr '04]

Let φ be a **qualitative** PCTL formula, and let ν be a regular valuation. The set of configurations that satisfy φ under the valuation ν is **effectively regular**.

Theorem: [E., Kučera, Mayr '04], [Bräzdiš, Kučera, Strazovský '05]

The qualitative model-checking problem for PCTL and pushdown systems is EXPTIME-hard and solvable in EXPSPACE. The quantitative model-checking problem for PCTL is undecidable.

Good approximation algorithms for “procedures that do not return values”.

Conclusions

Probabilistic verification is feasible for procedural programs.

Recursion requires to deal with polynomial (quadratic) instead of linear systems of fixpoint equations.

Decision algorithms by reduction to the theory of the reals.

Approximation algorithms use (carefully crafted) variants Newton's method.

Convergence rate of numerical algorithms very well understood.

Verification of Population Protocols

Javier Esparza

Technical University of Munich

Joint work with

Pierre Ganty, Jérôme Leroux, Rupak Majumdar,
and

Michael Blondin, Stefan Jaax, and Philipp Meyer

Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark



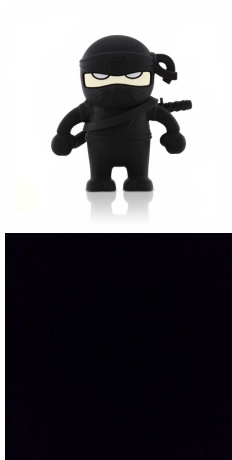
Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark
- They must decide **by majority** to attack or not (“don’t attack” if tie)



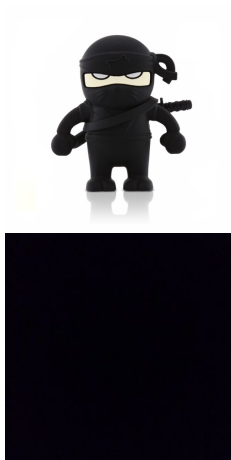
Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark
- They must decide **by majority** to attack or not (“don’t attack” if tie)



Deaf Black Ninjas in the Dark

- Deaf Black Ninjas meet at a Zen garden in the dark
- They must decide **by majority** to attack or not (“don’t attack” if tie)
- **How can they conduct the vote?**



Deaf Black Ninjas in the Dark

- Ninjas randomly wander around the garden, interacting when they bump into each other

Deaf Black Ninjas in the Dark

- Ninjas randomly wander around the garden, interacting when they bump into each other
- Each Ninja stores their current estimation of the final outcome of the vote (Yes or No). Additionally, it is Active or Passive.

Deaf Black Ninjas in the Dark

- Ninjas randomly wander around the garden, interacting when they bump into each other
- Each Ninja stores their current estimation of the final outcome of the vote (Yes or No). Additionally, it is Active or Passive.
- Initially all Ninjas are Active, and their initial estimation is their own vote

Deaf Black Ninjas in the Dark

- Ninjas **randomly** wander around the garden, interacting when they bump into each other
- Each Ninja stores their current estimation of the final outcome of the vote (**Y**es or **N**o). Additionally, it is **A**ctive or **P**assive.
- Initially all Ninjas are **A**ctive, and their initial estimation is their own vote
- Ninjas follow this protocol:

(**Y**A , **N**A) \rightarrow (**N**P , **N**P) (opposite votes “cancel”)

(**Y**A , **N**P) \rightarrow (**Y**A , **Y**P) (active “survivors” tell

(**N**A , **Y**P) \rightarrow (**N**A , **N**P) outcome to passive Ninjas)

(**N**P , **Y**P) \rightarrow (**N**P , **N**P) (to deal with ties)

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors
- “soups” of interacting molecules

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors
- “soups” of interacting molecules
- people in social networks

Population protocols (PP)

Theoretical model for distributed computation

Proposed in 2004 by Angluin *et al.*

Designed to model collections of

identical, finite-state, and mobile agents

like

- ad-hoc networks of mobile sensors
- “soups” of interacting molecules
- people in social networks
- ... and Ninjas

Syntax

A **PP-scheme** is a pair (Q, Δ) , where

- Q is a finite set of **states**, and
- $\Delta \subseteq (Q \times Q) \times (Q \times Q)$ is a set of **interactions**.

Syntax

A **PP-scheme** is a pair (Q, Δ) , where

- Q is a finite set of **states**, and
- $\Delta \subseteq (Q \times Q) \times (Q \times Q)$ is a set of **interactions**.

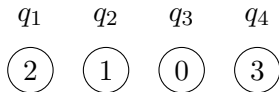
Intuition:

if $(q_1, q_2) \mapsto (q'_1, q'_2) \in \Delta$ **and**
two agents in states q_1 and q_2 “meet”,
then the agents can interact and
change their states to q'_1, q'_2 .

Assumption: at least one interaction for each pair of states
(possibly $(q_1, q_2) \mapsto (q_1, q_2)$)

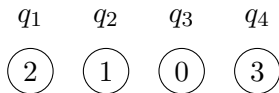
Semantics

Configuration: mapping $C: Q \rightarrow \mathbb{N}$, where $C(q)$ is the current number of agents in state q .



Semantics

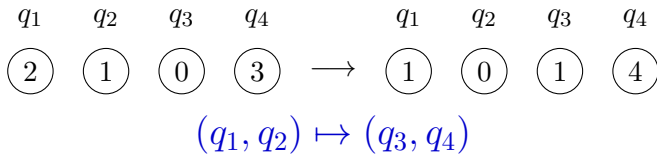
Configuration: mapping $C: Q \rightarrow \mathbb{N}$, where $C(q)$ is the current number of agents in state q .



$$(q_1, q_2) \mapsto (q_3, q_4)$$

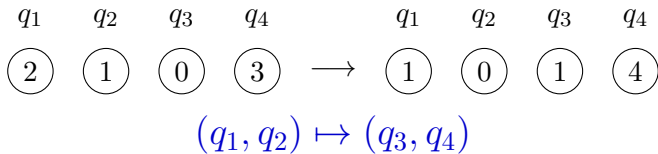
Semantics

Configuration: mapping $C: Q \rightarrow \mathbb{N}$, where $C(q)$ is the current number of agents in state q .



Semantics

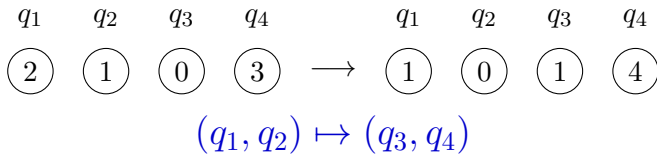
Configuration: mapping $C: Q \rightarrow \mathbb{N}$, where $C(q)$ is the current number of agents in state q .



If several steps are possible, a **random** scheduler chooses one (fixed nonzero prob. for each pair)

Semantics

Configuration: mapping $C: Q \rightarrow \mathbb{N}$, where $C(q)$ is the current number of agents in state q .



If several steps are possible, a **random** scheduler chooses one (fixed nonzero prob. for each pair)

Execution: infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \dots$ of steps

Population protocols (PPs)

A **population protocol** (PP) consists of

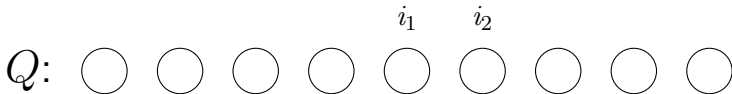
- A PP-scheme (Q, Δ)



Population protocols (PPs)

A **population protocol** (PP) consists of

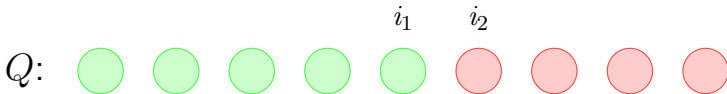
- A PP-scheme (Q, Δ)
- An ordered subset (i_1, \dots, i_k) of **input states**



Population protocols (PPs)

A **population protocol** (PP) consists of

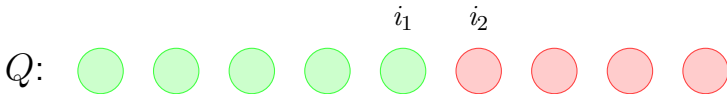
- A PP-scheme (Q, Δ)
- An ordered subset (i_1, \dots, i_k) of **input states**
- A partition of Q into 1-states (green) and 0-states (pink)



Population protocols (PPs)

A **population protocol** (PP) consists of

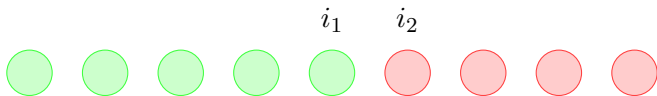
- A PP-scheme (Q, Δ)
- An ordered subset (i_1, \dots, i_k) of **input states**
- A partition of Q into 1-states (green) and 0-states (pink)



An execution **reaches consensus** $b \in \{0, 1\}$ if from some point on every agent stays within the b -states.

Computing with PPs

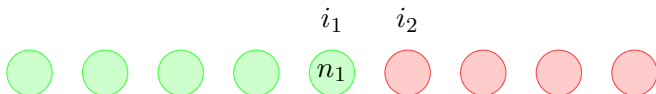
A PP computes the value b for input (n_1, n_2, \dots, n_k) if executions starting at the configuration



Computing with PPs

A PP computes the value b for input (n_1, n_2, \dots, n_k) if executions starting at the configuration

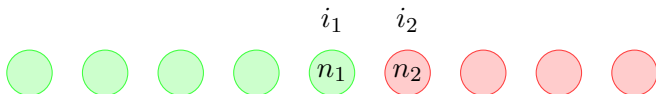
$$n_1 \cdot \mathbf{i}_1$$



Computing with PPs

A PP computes the value b for input (n_1, n_2, \dots, n_k) if executions starting at the configuration

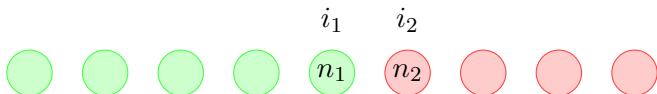
$$n_1 \cdot \mathbf{i}_1 + n_2 \cdot \mathbf{i}_2$$



Computing with PPs

A PP computes the value b for input (n_1, n_2, \dots, n_k) if executions starting at the configuration

$$n_1 \cdot \mathbf{i}_1 + n_2 \cdot \mathbf{i}_2 + \dots + n_k \cdot \mathbf{i}_k$$

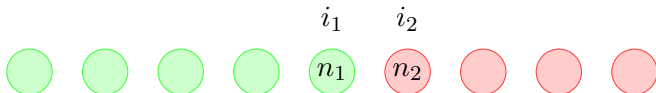


Computing with PPs

A PP computes the value b for input (n_1, n_2, \dots, n_k) if executions starting at the configuration

$$n_1 \cdot \mathbf{i}_1 + n_2 \cdot \mathbf{i}_2 + \dots + n_k \cdot \mathbf{i}_k$$

reach consensus b with probability 1.

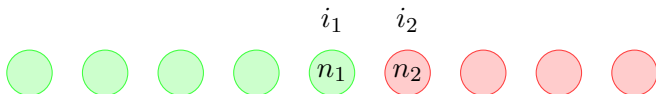


Computing with PPs

A PP computes the value b for input (n_1, n_2, \dots, n_k) if executions starting at the configuration

$$n_1 \cdot \mathbf{i}_1 + n_2 \cdot \mathbf{i}_2 + \dots + n_k \cdot \mathbf{i}_k$$

reach consensus b with probability 1.



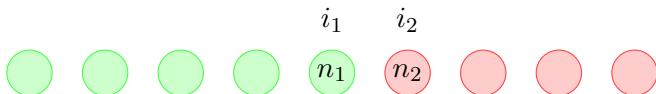
Equivalently: executions that do not reach consensus or reach consensus $1 - b$ have probability 0

Computing with PPs

A PP computes the value b for input (n_1, n_2, \dots, n_k) if executions starting at the configuration

$$n_1 \cdot \mathbf{i}_1 + n_2 \cdot \mathbf{i}_2 + \dots + n_k \cdot \mathbf{i}_k$$

reach consensus b with probability 1.



Equivalently: executions that do not reach consensus or reach consensus $1 - b$ have probability 0

A PP computes $P(x_1, \dots, x_n): \mathbb{N}^n \rightarrow \{0, 1\}$ if it computes $P(n_1, \dots, n_k)$ for every input (n_1, \dots, n_k)

Previous work

Expressive power thoroughly studied:

- PPs compute exactly the Presburger predicates
(Angluin *et al.* 2007)

Previous work

Expressive power thoroughly studied:

- PPs compute exactly the Presburger predicates (Angluin *et al.* 2007)
- Probabilistic PPs (Angluin *et al.* 2004-2006, Chatzigiannakis and Spirakis, 2008)
- Fault-tolerant PPs (Delporte-Gallet *et al.* 2006)
- Private computation in PPs (Delporte-Gallet *et al.* 2007)
- PPs with identifiers (Guerraoui *et al.* 2007)
- PPs with a leader (Angluin *et al.* 2008)
- Mediated PPs (Michail *et al.*, 2011)
- Trustful PPs (Bournez *et al.*, 2013)

Well-specified protocols

Q: And if the processes only reach consensus with probability < 1 ?

Well-specified protocols

Q: And if the processes only reach consensus with probability < 1 ?

A: *Then your protocol is not well-specified. Repair it!*

Well-specified protocols

Q: And if the processes only reach consensus with probability < 1 ?

A: *Then your protocol is not well-specified. Repair it!*

Q: And if the processes may reach consensus 0 and 1 for the same input, both with positive probability?

Well-specified protocols

Q: And if the processes only reach consensus with probability < 1 ?

A: *Then your protocol is not well-specified. Repair it!*

Q: And if the processes may reach consensus 0 and 1 for the same input, both with positive probability?

A: *Then your protocol is not well-specified. Repair it!*

Well-specified protocols

Q: And if the processes only reach consensus with probability < 1 ?

A: *Then your protocol is not well-specified. Repair it!*

Q: And if the processes may reach consensus 0 and 1 for the same input, both with positive probability?

A: *Then your protocol is not well-specified. Repair it!*

Q: And how do I know if my protocol is well-specified?

Well-specified protocols

Q: And if the processes only reach consensus with probability < 1 ?

A: *Then your protocol is not well-specified. Repair it!*

Q: And if the processes may reach consensus 0 and 1 for the same input, both with positive probability?

A: *Then your protocol is not well-specified. Repair it!*

Q: And how do I know if my protocol is well-specified?

A: *That's your problem . . .*

Well-specified protocols

Q: And if the processes only reach consensus with probability < 1 ?

A: *Then your protocol is not well-specified. Repair it!*

Q: And if the processes may reach consensus 0 and 1 for the same input, both with positive probability?

A: *Then your protocol is not well-specified. Repair it!*

Q: And how do I know if my protocol is well-specified?

A: *That's your problem . . .*

Well-specification problem: Given a protocol, decide if it is well-specified.

Correctness problem: Given a protocol and a Presburger predicate, decide if the protocol is well-specified and computes the predicate.

Verifying population protocols: Previous work

Verifying population protocols: Previous work

- For each input, the semantics of the protocol is a finite-state Markov chain
- The semantics for all inputs is an infinite collection of finite-state Markov chains

Verifying population protocols: Previous work

- For each input, the semantics of the protocol is a finite-state Markov chain
- The semantics for all inputs is an infinite collection of finite-state Markov chains
- Use model-checkers (SPIN, PRISM , ...) to verify correctness for some inputs

Pang *et al.*, 2008 ; Sun *et al.*, 2009

Chatzigiannakis *et al.*, 2010 ; Clément *et al.*, 2011

Verifying population protocols: Previous work

- For each input, the semantics of the protocol is a finite-state Markov chain
- The semantics for all inputs is an infinite collection of finite-state Markov chains
- Use model-checkers (SPIN, PRISM , ...) to verify correctness for some inputs
Pang et al., 2008 ; Sun et al., 2009
Chatzigiannakis et al., 2010 ; Clément et al., 2011
- Use interactive theorem provers (Coq) to prove correctness of a specific protocol
Deng et al., 2009 and 2011

Verifying population protocols: Previous work

- For each input, the semantics of the protocol is a finite-state Markov chain
- The semantics for all inputs is an infinite collection of finite-state Markov chains
- Use model-checkers (SPIN, PRISM , ...) to verify correctness for some inputs
Pang et al., 2008 ; Sun et al., 2009
Chatzigiannakis et al., 2010 ; Clément et al., 2011
- Use interactive theorem provers (Coq) to prove correctness of a specific protocol
Deng et al., 2009 and 2011

Not complete or not automatic.

Main results

Are the well-specification and correctness problems decidable?

Main results

Are the well-specification and correctness problems decidable?

Open for about 10 years.

Main results

Are the well-specification and correctness problems decidable?

Open for about 10 years.

Theorem: The well-specification and correctness problems can be reduced to the reachability problem for Petri nets, and are thus decidable.

Main results

Are the well-specification and correctness problems decidable?

Open for about 10 years.

Theorem: The well-specification and correctness problems can be reduced to the reachability problem for Petri nets, and are thus decidable.

Theorem: The reachability problem for Petri nets can be reduced to the well-specification and correctness problems for PPs.

From PPs to Petri nets

Population protocols Petri nets

State

Place

From PPs to Petri nets

Population protocols Petri nets

State

Place

Interaction

$(q_1, q_2) \mapsto (q'_1, q'_2)$

Transition with

input places q_1, q_2

output places q'_1, q'_2

From PPs to Petri nets

Population protocols Petri nets

State

Place

Interaction

$(q_1, q_2) \mapsto (q'_1, q'_2)$

Transition with

input places q_1, q_2

output places q'_1, q'_2

PP-scheme

Net **without marking**

From PPs to Petri nets

Population protocols Petri nets

State

Place

Interaction

$(q_1, q_2) \mapsto (q'_1, q'_2)$

Transition with

input places q_1, q_2

output places q'_1, q'_2

PP-scheme

Net **without marking**

Configuration

Marking

From PPs to Petri nets

Population protocols Petri nets

State

Place

Interaction

$(q_1, q_2) \mapsto (q'_1, q'_2)$

Transition with

input places q_1, q_2

output places q'_1, q'_2

PP-scheme

Net **without marking**

Configuration

Marking

Configuration graph

Reachability graph

From PPs to Petri nets

Population protocols Petri nets

State

Place

Interaction

$(q_1, q_2) \mapsto (q'_1, q'_2)$

Transition with

input places q_1, q_2

output places q'_1, q'_2

PP-scheme

Net **without marking**

Configuration

Marking

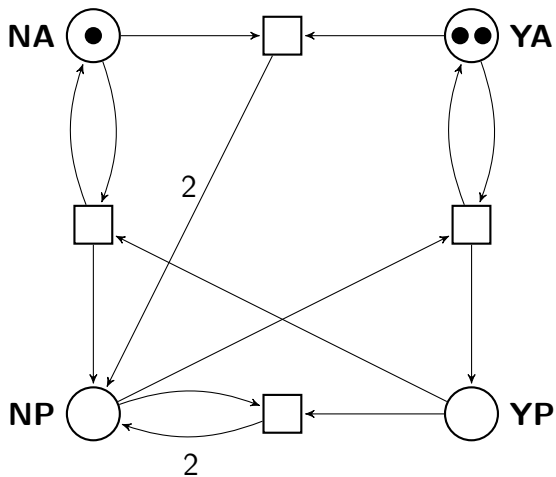
Configuration graph

Reachability graph

PP

Net + **infinite family of
initial markings**

Petri net of the majority protocol



Reducing well-specification to a reachability problem

Configuration graph of a PP:

- **Nodes:** all (infinitely many) possible configurations
- **Edges:** steps

Reducing well-specification to a reachability problem

Configuration graph of a PP:

- **Nodes:** all (infinitely many) possible configurations
- **Edges:** steps

Fact: Infinite, but every node has only finitely many successors.

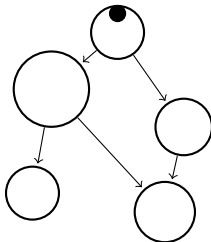
Reducing well-specification to a reachability problem

Configuration graph of a PP:

- **Nodes:** all (infinitely many) possible configurations
- **Edges:** steps

Fact: Infinite, but every node has only finitely many successors.

Fact: Every execution of a PP gets eventually trapped in a bottom SCC of its configuration graph w.p.1, and visits all configurations of the SCC infinitely often w.p.1.



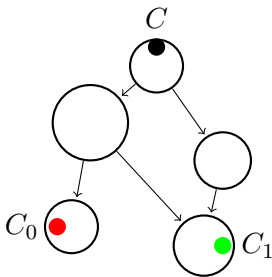
Reducing well-specification to a reachability problem

Bottom configuration: configuration of a bottom SCC.

Fact: A PP is ill-specified iff there is

- an initial configuration C , and
- two bottom configurations C_0 and C_1 , reachable from C

such that C_0 has at least one agent in a 0-state, and C_1 has at least one agent in a 1-state.



Well-specification is decidable

Theorem 1: Given two (possibly infinite!) Presburger sets of configurations C_1, C_2 of a Petri net, it is decidable if some configuration of C_2 is reachable from some configuration of C_1 .

Easy reduction to the reachability problem for Petri nets.

Well-specification is decidable

Theorem 1: Given two (possibly infinite!) Presburger sets of configurations C_1, C_2 of a Petri net, it is decidable if some configuration of C_2 is reachable from some configuration of C_1 .

Easy reduction to the reachability problem for Petri nets.

Theorem 2: The set of **all** configurations belonging to **all** bottom SCCs from **all** configurations is an **effectively Presburger** set.

Well-specification is decidable

Theorem 1: Given two (possibly infinite!) Presburger sets of configurations C_1, C_2 of a Petri net, it is decidable if some configuration of C_2 is reachable from some configuration of C_1 .

Easy reduction to the reachability problem for Petri nets.

Theorem 2: The set of **all** configurations belonging to **all** bottom SCCs from **all** configurations is an **effectively Presburger** set.

“Presburgerness” follows immediately from a classical result by Eilenberg and Schützenberger (1969) on rational sets in commutative monoids.

Well-specification is decidable

Theorem 1: Given two (possibly infinite!) Presburger sets of configurations C_1, C_2 of a Petri net, it is decidable if some configuration of C_2 is reachable from some configuration of C_1 .

Easy reduction to the reachability problem for Petri nets.

Theorem 2: The set of **all** configurations belonging to **all** bottom SCCs from **all** configurations is an **effectively Presburger** set.

“Presburgerness” follows immediately from a classical result by Eilenberg and Schützenberger (1969) on rational sets in commutative monoids.

Effectiveness follows from a bound on the size of the Presburger representation due to Leroux (2011).

Well-specification is decidable

- \mathcal{S} : protocol scheme
- \mathcal{I} : set of all initial configurations
- \mathcal{B}_b (where $b \in \{0, 1\}$): set of all bottom configurations with at least one agent in a b -state

Well-specification is decidable

- \mathcal{S} : protocol scheme
- \mathcal{I} : set of all initial configurations
- \mathcal{B}_b (where $b \in \{0, 1\}$): set of all bottom configurations with at least one agent in a b -state

Decision procedure:

Well-specification is decidable

- \mathcal{S} : protocol scheme
- \mathcal{I} : set of all initial configurations
- \mathcal{B}_b (where $b \in \{0, 1\}$): set of all bottom configurations with at least one agent in a b -state

Decision procedure:

- Construct the net $\mathcal{S} \parallel \mathcal{S}$ (“two copies of \mathcal{S} side by side”).

Well-specification is decidable

- \mathcal{S} : protocol scheme
- \mathcal{I} : set of all initial configurations
- \mathcal{B}_b (where $b \in \{0, 1\}$): set of all bottom configurations with at least one agent in a b -state

Decision procedure:

- Construct the net $\mathcal{S} \parallel \mathcal{S}$ (“two copies of \mathcal{S} side by side”).
- Construct the set $\mathcal{I}_2 = \{(C, C) \mid C \in \mathcal{I}\}$ of configurations of $\mathcal{S} \parallel \mathcal{S}$.

Presburger, because \mathcal{I} is Presburger.

Well-specification is decidable

- \mathcal{S} : protocol scheme
- \mathcal{I} : set of all initial configurations
- \mathcal{B}_b (where $b \in \{0, 1\}$): set of all bottom configurations with at least one agent in a b -state

Decision procedure:

- Construct the net $\mathcal{S} \parallel \mathcal{S}$ (“two copies of \mathcal{S} side by side”).
- Construct the set $\mathcal{I}_2 = \{(C, C) \mid C \in \mathcal{I}\}$ of configurations of $\mathcal{S} \parallel \mathcal{S}$.
Presburger, because \mathcal{I} is Presburger.
- Check if $\mathcal{B}_0 \times \mathcal{B}_1$ is reachable from \mathcal{I}_2
 $\mathcal{B}_0 \times \mathcal{B}_1$ is effectively Presburger by Theorem 2, the check is effective by Theorem 1.

More decidable problems

Given: A PP P , a Presburger predicate Π

Decide: Does P compute Π ?

Given: A well-specified PP P

Compute: A Presburger formula (or semilinear representation of) the predicate computed by P .

Fighting complexity I: The class WS^2

Search for a subclass of the class WS of **well-specified protocols** that

- Has a membership problem of reasonable complexity.
- Can compute all Presburger predicates.

Fighting complexity I: The class WS^2

Search for a subclass of the class WS of **well-specified protocols** that

- Has a membership problem of reasonable complexity.
- Can compute all Presburger predicates.

Many protocols from the literature are **silent**: Executions end w.p.1 in **terminal configurations** that enable no transitions.

Equivalent definition: bottom SCCs contain only one configuration.

Fighting complexity I: The class WS^2

Search for a subclass of the class WS of **well-specified protocols** that

- Has a membership problem of reasonable complexity.
- Can compute all Presburger predicates.

Many protocols from the literature are **silent**: Executions end w.p.1 in **terminal configurations** that enable no transitions.

Equivalent definition: bottom SCCs contain only one configuration.

Proposition: WS^2 protocols (well specified and silent) can compute all Presburger predicates.

Fighting complexity I: The class WS^2

Search for a subclass of the class WS of **well-specified protocols** that

- Has a membership problem of reasonable complexity.
- Can compute all Presburger predicates.

Many protocols from the literature are **silent**: Executions end w.p.1 in **terminal configurations** that enable no transitions.

Equivalent definition: bottom SCCs contain only one configuration.

Proposition: WS^2 protocols (well specified and silent) can compute all Presburger predicates.

Proposition : Petri net reachability is reducible to the membership problem for WS^2 .

Fighting complexity II: The class WS^3

WS^2 : Well-sp. silent

Termination

For every reachable configuration C there exists an execution leading from C to a terminal conf. C_{\perp}

Consensus

All terminal configurations reachable from a given initial configuration form the same consensus.

Fighting complexity II: The class WS^3

WS^2 : Well-sp. silent

Termination

For every reachable configuration C there exists an execution leading from C to a terminal conf. C_{\perp}

Consensus

All terminal configurations reachable from a given initial configuration form the same consensus.

WS^3 : Well-sp. strongly silent

Layered Termination

For every configuration C there exists a **layered execution** leading from C to a terminal configuration C_{\perp}

Strong Consensus

All terminal configurations **potentially reachable** from a given initial configuration form the same consensus.

Layered Termination

A protocol is **layered** if there is a partition of the set T of transitions into **layers** T_1, \dots, T_n s.t. for every configuration C (reachable or not):

- all executions from C containing only transitions of a single layer are finite.
- if all transitions of T_i are disabled at C , then they cannot be re-enabled by any sequence of transitions of T_{i+1}, \dots, T_n .

An execution is **layered** if it “respects the layers”, i.e., if it belongs to $T_1^* T_2^* \dots T_n^*$.

Layered Termination

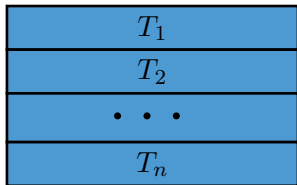
A protocol is **layered** if there is a partition of the set T of transitions into **layers** T_1, \dots, T_n s.t. for every configuration C (reachable or not):

- all executions from C containing only transitions of a single layer are finite.
- if all transitions of T_i are disabled at C , then they cannot be re-enabled by any sequence of transitions of T_{i+1}, \dots, T_n .

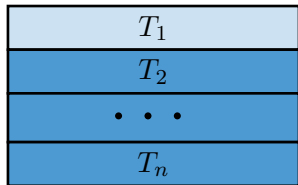
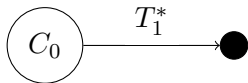
An execution is **layered** if it “respects the layers”, i.e., if it belongs to $T_1^* T_2^* \dots T_n^*$.

Fact: For every configuration C (**reachable or not**) there exists a layered execution leading from C to a terminal configuration C_\perp .

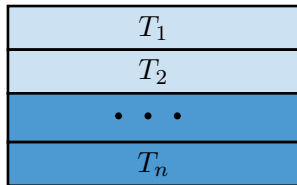
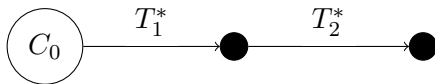
Layered Termination



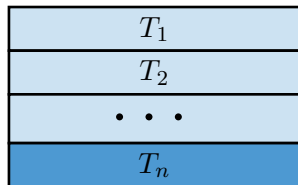
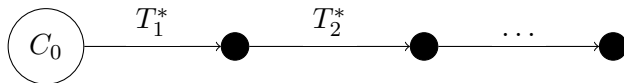
Layered Termination



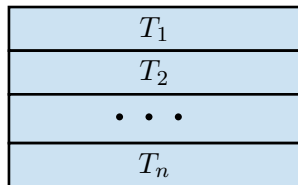
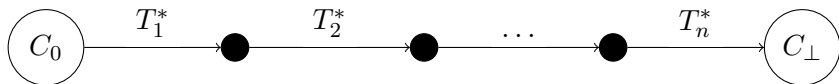
Layered Termination



Layered Termination



Layered Termination



Complexity of checking Layered Termination

Lemma: Deciding Layered Termination is in NP.

Complexity of checking Layered Termination

Lemma: Deciding Layered Termination is in NP.

Proof sketch:

- Guess layers.
- Test that each individual layer terminates.
Reducible to a Linear Programming Problem
- Test that lower layers cannot re-enable higher layers.
Simple syntactic check.

Strong Consensus

Replace reachability by *cruder* relation called *potential reachability*:

$$\begin{aligned}\text{Reachability} &\implies \text{Potential Reachability} \\ \text{Potential Reachability} &\not\implies \text{Reachability}\end{aligned}$$

Potential reachability is defined in terms of a class of linear invariants derivable from the Petri net of the protocol by syntactic means (place invariants, siphons, traps).

A configuration C' is **potentially reachable** from C if both C and C' satisfy the same invariants of the class.

Lemma: Deciding Strong Consensus is in co-NP.

Completeness

Lemma: All well-specified population protocols can be represented by an equivalent population protocol satisfying **Layered Termination** and **Strong Consensus**.

By quantifier elimination, all predicates computable by Population Protocols can be defined as boolean combinations of

- **Threshold:** Is the weighted sum of the input values larger than a given threshold?

$$\sum_i \alpha_i x_i > c$$

- **Remainder:** Is the sum of the input values modulo a given m equal to a given c ?

$$\sum_i \alpha_i x_i \bmod m = c$$

- Give WS^3 protocols for Threshold and Remainder predicates
- Prove that WS^3 protocols are closed under conjunction and negation.

Implementation

On top of the SMT-solver **Z3**.

Our tool reads a protocol and constructs two sets of constraints:

- The first is **satisfiable** iff. **Layered Termination** holds.
- The second is **unsatisfiable** iff. **Strong Consensus** holds.

Protocols from the literature for Majority, Threshold, Remainder, etc. belong to WS^3 .

Experimental Results

Experiments were performed on a machine equipped with an Intel Core i7-4810MQ CPU and 16 GB of RAM.

Threshold			
ℓ_{\max}	$ Q $	$ T $	Time[s]
3	28	288	8.0
4	36	478	26.5
5	44	716	97.6
6	52	1002	243.4
7	60	1336	565.0
8	68	1718	1019.7
9	76	2148	2375.9
10	84	2626	timeout

Remainder			
m	$ Q $	$ T $	Time[s]
10	12	65	0.4
20	22	230	2.8
30	32	495	15.9
40	42	860	79.3
50	52	1325	440.3
60	62	1890	3055.4
70	72	2555	3176.5
80	82	3320	timeout

Experimental Results

Flock of birds[1]			
c	$ Q $	$ T $	Time[s]
20	21	210	1.5
25	26	325	3.3
30	31	465	7.7
35	36	630	20.8
40	41	820	106.9
45	46	1035	295.6
50	51	1275	181.6
55	56	1540	timeout

[1] Chatzigiannakis et al., 2010

Flock of birds[2]			
c	$ Q $	$ T $	Time[s]
50	51	99	11.8
100	101	199	44.8
150	151	299	369.1
200	201	399	778.8
250	251	499	1554.2
300	301	599	2782.5
325	326	649	3470.8
350	351	699	timeout

[2] Clement et al., 2011

Conclusions

- The natural verification problems for population protocols are decidable.
- Efficient verification algorithms for the class WS^3 .
- Implementation on top of SMT-solvers.

Conclusions

- The natural verification problems for population protocols are decidable.
- Efficient verification algorithms for the class WS^3 .
- Implementation on top of SMT-solvers.
- Many open questions:
 - ▶ Complexity for immediate observation and immediate transmission protocols
 - ▶ Continuous Petri nets as abstractions
 - ▶ Expressive power of PP in non-uniform computational models
 - ▶ Applications to theoretical chemistry and systems biology
 - ▶ Correctness problem and convergence speed for WS^3 protocols.
 - ▶ Fault localization and repair.
 - ▶ Synthesis of WS^3 protocols.



Thank You