Computing Least Fixed Points of Probabilistic Systems of Polynomials

Javier Esparza, Andreas Gaiser and Stefan Kiefer
{ esparza, gaiser, kiefer } @in.tum.de
Institut für Informatik
Technische Universität München, Germany

December 21, 2009

Abstract

We study systems of equations of the form $X_1 = f_1(X_1, \ldots, X_n), \ldots, X_n = f_n(X_1, \ldots, X_n)$ where each f_i is a polynomial with nonnegative coefficients that add up to 1. The least nonnegative solution, say μ, of such equation systems is central to problems from various areas, like physics, biology, computational linguistics and probabilistic program verification. We give a simple and strongly polynomial algorithm to decide whether $\mu = (1, \ldots, 1)$ holds. Furthermore, we present an algorithm that computes reliable sequences of lower and upper bounds on μ, converging linearly to μ. Our algorithm has these features despite using inexact arithmetic for efficiency. We report on experiments that show the performance of our algorithms.

1 Introduction

We study how to efficiently compute the least nonnegative solution of an equation system of the form

$$X_1 = f_1(X_1, \ldots, X_n) \quad \ldots \quad X_n = f_n(X_1, \ldots, X_n) \ ,$$

where, for every $i \in \{1, \ldots, n\}$, f_i is a polynomial over $X_1, \ldots, X_n$ with positive rational coefficients that *add up to 1*.[1] The solutions are the fixed points of the function $f : \mathbb{R}^n \to \mathbb{R}^n$ with $f = (f_1, \ldots, f_n)$. We call f a *probabilistic system of polynomials* (short: *PSP*). For example, the PSP

$$f(X_1, X_2) = \left(\frac{1}{2} X_1 X_2 + \frac{1}{2} \ , \ \frac{1}{4} X_2 X_2 + \frac{1}{4} X_1 + \frac{1}{2} \right)$$

induces the equation system

$$X_1 = \tfrac{1}{2} X_1 X_2 + \tfrac{1}{2} \qquad X_2 = \tfrac{1}{4} X_2 X_2 + \tfrac{1}{4} X_1 + \tfrac{1}{2} \ .$$

Obviously, $\overline{1} = (1, \ldots, 1)$ is a fixed point of every PSP. By Kleene's theorem, every PSP has a least nonnegative fixed point (called just least fixed point in what follows), given by the limit of the sequence $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$

[1]Later, we allow that the coefficients add up to *at most* 1.

PSPs are important in different areas of the theory of stochastic processes and computational models. A fundamental result of the theory of branching processes, with numerous applications in physics, chemistry and biology (see e.g. [8, 2]), states that extinction probabilities of species are equal to the least fixed point of a PSP. The same result has been recently shown for the probability of termination of certain probabilistic recursive programs [6, 5]. The consistency of stochastic context-free grammars, a problem of interest in statistical natural language processing, also reduces to checking whether the least fixed point of a PSP equals $\overline{1}$ (see e.g. [10]).

Given a PSP f with least fixed point μ_f, we study how to efficiently solve the following two problems: (1) decide whether $\mu_f = \overline{1}$, and (2) given a rational number $\epsilon > 0$, compute $\mathbf{lb}, \mathbf{ub} \in \mathbb{Q}^n$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$ (where $\mathbf{u} \leq \mathbf{v}$ for vectors $\mathbf{u}, \mathbf{v}$ means $\leq$ in all components). While the motivation for Problem (2) is clear (compute the probability of extinction with a given accuracy), the motivation for Problem (1) requires perhaps some explanation. In the case study of Section 4.3 we consider a family of PSPs, taken from [8], modelling the neutron branching process in a ball of radioactive material of radius D (the family is parameterized by D). The least fixed point is the probability that a neutron produced through spontaneous fission *does not* generate an infinite "progeny" through successive collisions with atoms of the ball; loosely speaking, this is the probability that the neutron *does not* generate a chain reaction and the ball *does not* explode. Since the number of atoms in the ball is very large, spontaneous fission produces many neutrons per second, and so even if the probability that a given neutron produces a chain reaction is very small, the ball will explode with large probability in a very short time. It is therefore important to determine the largest radius D at which the probability of no chain reaction is still 1 (usually called the *critical radius*). An algorithm for Problem (1) allows to compute the critical radius using binary search. A similar situation appears in the analysis of parameterized probabilistic programs. In [6, 5] it is shown that the question whether a probabilistic program almost surely terminates can be reduced to Problem (1). Using binary search one can find the "critical" value of the parameter for which the program may not terminate any more.

Etessami and Yannakakis show in [6] that Problem (1) can be solved in polynomial time by a reduction to (exact) Linear Programming (LP), which is not known to be strongly polynomial. Our first result reduces Problem (1) to solving a system of linear equations, resulting in a strongly polynomial algorithm for Problem (1). The Maple library offers exact arithmetic solvers for LP and systems of linear equations, which we use to test the performance of our new algorithm. In the neutron branching process discussed above we obtain speed-ups of about one order of magnitude with respect to LP.

The second result of the paper is, to the best of our knowledge, the first practical algorithm for Problem (2). Lower bounds for μ_f can be computed using Newton's method for approximating a root of the function $f(\overline{X}) - \overline{X}$. This has recently been investigated in detail [6, 9, 4]. However, Newton's method faces considerable numerical problems. Experiments show that naive use of exact arithmetic is inefficient, while floating-point computation leads to false results even for very small systems. For instance, the PReMo tool [11], which implements Newton's method with floating-point arithmetic for efficiency, reports $\mu_f \geq 1$ for a PSP with only 7 variables and small coefficients, although

$\mu_f < 1$ is the case (see Section 3.1).

Our algorithm produces a sequence of guaranteed lower and upper bounds, both of which converge linearly to μ_f. Linear convergence means that, loosely speaking, the number of accurate bits of the bound is a linear function of the position of the bound in the sequence. The algorithm is based on the following idea. Newton's method is an iterative procedure that, given a current lower bound $\mathbf{lb}$ on μ_f, applies a certain operator $\mathcal{N}$ to it, yielding a new, more precise lower bound $\mathcal{N}(\mathbf{lb})$. Instead of computing $\mathcal{N}(\mathbf{lb})$ using exact arithmetic, our algorithm computes *two* consecutive Newton steps, i.e., $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$, using *inexact* arithmetic. Then it checks if the result satisfies a carefully chosen condition. If so, the result is taken as the next lower bound. If not, then the precision is increased, and the computation redone. The condition is eventually satisfied, assuming the results of computing with increased precision converge to the exact result. Usually, the repeated inexact computation is much faster than the exact one. At the same time, a careful (and rather delicate) analysis shows that the sequence of lower bounds converges linearly to μ_f.

Computing *upper* bounds is harder, and seemingly has not been considered in the literature before. Similarly to the case of lower bounds, we apply f twice to $\mathbf{ub}$, i.e., we compute $f(f(\mathbf{ub}))$ with increasing precision until a condition holds. The sequence so obtained may not even converge to μ_f. So we need to introduce a further operation, after which we can then prove linear convergence.

We test our algorithm on the neutron branching process. The time needed to obtain lower and upper bounds on the probability of no explosion with $\epsilon = 0.0001$ lies below the time needed to check, using exact LP, whether this probability is 1 or smaller than one. That is, in this case study our algorithm is faster, and provides more information.

The rest of the paper is structured as follows. We give preliminary definitions and facts in Section 2. Sections 3 and 4 present our algorithms for solving Problems (1) and (2), and report on their performance on some case studies. Section 5 contains our conclusions. A shorter version of this paper will appear in *27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010).*

2 Preliminaries

Vectors and matrices. We use bold letters for designating (column) vectors, e.g. $\mathbf{v} \in \mathbb{R}^n$. We write $\overline{s}$ with $s \in \mathbb{R}$ for the vector $(s, \ldots, s)^\top \in \mathbb{R}^n$ (where $^\top$ indicates transpose), if the dimension n is clear from the context. The i-th component of $\mathbf{v} \in \mathbb{R}^n$ will be denoted by $\mathbf{v}_i$. We write $\mathbf{x} = \mathbf{y}$ (resp. $\mathbf{x} \le \mathbf{y}$ resp. $\mathbf{x} \prec \mathbf{y}$) if $\mathbf{x}_i = \mathbf{y}_i$ (resp. $\mathbf{x}_i \le \mathbf{y}_i$ resp. $\mathbf{x}_i < \mathbf{y}_i$) holds for all $i \in \{1, \ldots, n\}$. By $\mathbf{x} < \mathbf{y}$ we mean $\mathbf{x} \le \mathbf{y}$ and $\mathbf{x} \ne \mathbf{y}$.

By $\mathbb{R}^{m \times n}$ we denote the set of real matrices with m rows and n columns. We write *Id* for the identity matrix. For a square matrix A, we denote by $\rho(A)$ the *spectral radius* of A, i.e., the maximum of the absolute values of the eigenvalues. A matrix is *nonnegative* if all its entries are nonnegative. A nonnegative matrix $A \in \mathbb{R}^{n \times n}$ is *irreducible* if for every $k, l \in \{1, \ldots, n\}$ there exists an $i \in \mathbb{N}$ so that $(A^i)_{kl} \ne 0$.

Probabilistic Systems of Polynomials. We investigate equation systems of the form

$$X_1 = f_1(X_1, \ldots, X_n) \quad \ldots \quad X_n = f_n(X_1, \ldots, X_n),$$

where the f_i are polynomials in the variables $X_1, \ldots, X_n$ with positive real coefficients, and for every polynomial f_i the sum of its coefficients is *at most* 1. The vector $f := (f_1, \ldots, f_n)^\top$ is called a *probabilistic system of polynomials* (PSP for short) and is identified with its induced function $f : \mathbb{R}^n \to \mathbb{R}^n$. If $X_1, \ldots, X_n$ are the formal variables of f, we define $\overline{X} := (X_1, \ldots, X_n)^\top$ and $\mathrm{Var}(f) := \{X_1, \ldots, X_n\}$. We assume that f is represented as a list of polynomials, and each polynomial is a list of its monomials. If $S \subseteq \{X_1, \ldots, X_n\}$, then f_S denotes the result of removing the polynomial $f_i(X_1, \ldots, X_n)$ from f for every $x_i \notin S$; further, given $\mathbf{x} \in \mathbb{R}^n$ and $B \in \mathbb{R}^{n \times n}$, we denote by $\mathbf{x}_S$ and B_{SS} the vector and the matrix obtained from $\mathbf{x}$ and B by removing the entries with indices i such that $X_i \notin S$. The coefficients are represented as fractions of positive integers. The *size* of f is the size of that representation. The *degree* of f is the maximum of the degrees of $f_1, \ldots, f_n$. PSPs of degree 0 (resp. 1 resp. >1) are called *constant* (resp. *linear* resp. *superlinear*). PSPs f where the degree of each f_i is at least 2 are called *purely superlinear*. We write f' for the *Jacobian* of f, i.e., the matrix of first partial derivatives of f.

Given a PSP f, a variable X_i *depends directly* on a variable X_j if X_j "occurs" in f_i, more formally if $\frac{\partial f_i}{\partial X_j}$ is not the constant 0. A variable X_i *depends* on X_j if X_i depends directly on X_j or there is a variable X_k such that X_i depends directly on X_k and X_k depends on X_j. We often consider the *strongly connected components* (or SCCs for short) of the dependence relation. The SCCs of a PSP can be computed in linear time using e.g. Tarjan's algorithm. An SCC S of a PSP f is *constant* resp. *linear* resp. *superlinear* resp. *purely superlinear* if the PSP $\tilde{f}$ has the respective property, where $\tilde{f}$ is obtained by restricting f to the S-components and replacing all variables not in S by the constant 1. A PSP is an *scPSP* if it is not constant and consists of only one SCC. Notice that a PSP f is an scPSP if and only if $f'(\overline{1})$ is irreducible.

A fixed point of a PSP f is a vector $\mathbf{x} \geq \overline{0}$ with $f(\mathbf{x}) = \mathbf{x}$. By Kleene's theorem, there exists a least fixed point μ_f of f, i.e., $\mu_f \leq \mathbf{x}$ holds for every fixed point $\mathbf{x}$. Moreover, the sequence $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$ converges to μ_f. Vectors $\mathbf{x}$ with $\mathbf{x} \leq f(\mathbf{x})$ (resp. $\mathbf{x} \geq f(\mathbf{x})$) are called *pre-fixed* (resp. *post-fixed*) points. Notice that the vector $\overline{1}$ is always a post-fixed point of a PSP f, due to our assumption on the coefficients of a PSP. By Knaster-Tarski's theorem, μ_f is the least post-fixed point, so we always have $\overline{0} \leq \mu_f \leq \overline{1}$. It is easy to detect and remove all components i with $(\mu_f)_i = 0$ by a simple round-robin method (see e.g. [4]), which needs linear time in the size of f. We therefore assume in the following that $\mu_f \succ \overline{0}$.

3 An algorithm for consistency of PSPs

Recall that for applications like the neutron branching process it is crucial to know exactly whether $\mu_f = \overline{1}$ holds. We say a PSP f is *consistent* if $\mu_f = \overline{1}$; otherwise it is *inconsistent*. Similarly, we call a component i consistent if $(\mu_f)_i = 1$. We present a new algorithm for the consistency problem, i.e., the problem to check a PSP for consistency.

It was proved in [6] that consistency is checkable in polynomial time by reduction to Linear Programming (LP). We first observe that consistency of general PSPs can be reduced to consistency of scPSPs by computing the DAG of SCCs, and checking consistency SCC-wise [6]: Take any bottom SCC S, and check the consistency of f_S. (Notice that f_S is either constant or an scPSP; if constant, f_S is consistent iff $f_S = 1$, if an scPSP, we can check its consistency by assumption.) If f_S is inconsistent, then so is f, and we are done. If f_S is consistent, then we remove every f_i from f such that $x_i \in S$, replace all variables of S in the remaining polynomials by the constant 1, and iterate (choose a new bottom SCC, etc.). Note that this algorithm processes each polynomial at most once, as every variable belongs to exactly one SCC.

It remains to reduce the consistency problem for scPSPs to LP. The first step is:

Proposition 3.1. *[8, 6] An scPSP f is consistent iff $\rho(f'(\overline{1})) \leq 1$ (i.e., iff the spectral radius of the Jacobi matrix f' evaluated at the vector $\overline{1}$ is at most 1).*

The second step consists of observing that the matrix $f'(\overline{1})$ of an scPSP f is irreducible and nonnegative. It is shown in [6] that $\rho(A) \leq 1$ holds for an irreducible and nonnegative matrix A iff the system of inequalities

$$A\mathbf{x} \geq \mathbf{x} + \overline{1} \ , \ \mathbf{x} \geq \overline{0} \tag{3.1}$$

is infeasible. However, no strongly polynomial algorithm for LP is known, and we are not aware that (3.1) falls within any subclass solvable in strongly polynomial time [7].

We provide a very simple, strongly polynomial time algorithm to check whether $\rho(f'(\overline{1})) \leq 1$ holds. We need some results from Perron-Frobenius theory (see e.g. [3]).

Lemma 3.2. *Let $A \in \mathbb{R}^{n\times n}$ be nonnegative and irreducible.*

(1) $\rho(A)$ is a simple *eigenvalue of A.*

(2) There exists an eigenvector $\mathbf{v} \succ \overline{0}$ with $\rho(A)$ as eigenvalue.

(3) Every eigenvector $\mathbf{v} \succ \overline{0}$ has $\rho(A)$ as eigenvalue.

(4) For all $\alpha, \beta \in \mathbb{R} \setminus \{0\}$ and $\mathbf{v} > \overline{0}$: if $\alpha\mathbf{v} < A\mathbf{v} < \beta\mathbf{v}$, then $\alpha < \rho(A) < \beta$.

The following lemma is the key to the algorithm:

Lemma 3.3. *Let $A \in \mathbb{R}^{n\times n}$ be nonnegative and irreducible.*

(a) Assume there is $\mathbf{v} \in \mathbb{R}^n \setminus \{\overline{0}\}$ such that $(Id - A)\mathbf{v} = \overline{0}$. Then $\rho(A) \leq 1$ iff $\mathbf{v} \succ \overline{0}$ or $\mathbf{v} \prec \overline{0}$.

(b) Assume $\mathbf{v} = \overline{0}$ is the only solution of $(Id - A)\mathbf{v} = \overline{0}$. Then there exists a unique $\mathbf{x} \in \mathbb{R}^n$ such that $(Id - A)\mathbf{x} = \overline{1}$, and $\rho(A) \leq 1$ iff $\mathbf{x} \geq \overline{1}$ and $A\mathbf{x} < \mathbf{x}$.

Proof.

(a) From $(Id - A)\mathbf{v} = \overline{0}$ it follows $A\mathbf{v} = \mathbf{v}$. We see that $\mathbf{v}$ is an eigenvector of A with eigenvalue 1. So $\rho(A) \geq 1$.

($\Leftarrow$): As both $\mathbf{v}$ and $-\mathbf{v}$ are eigenvectors of A with eigenvalue 1, we can assume w.l.o.g. that $\mathbf{v} \succ \overline{0}$. By Lemma 3.2(3), $\rho(A)$ is the eigenvalue of $\mathbf{v}$, and so $\rho(A) = 1$.

($\Rightarrow$): Since $\rho(A) \leq 1$ and $\rho(A) \geq 1$, it follows that $\rho(A) = 1$. By Lemma 3.2(1) and (2), the eigenspace of the eigenvalue 1 is one-dimensional and contains a vector $\mathbf{x} \succ \overline{0}$. So $\mathbf{v} = \alpha \cdot \mathbf{x}$ for some $\alpha \in \mathbb{R}, \alpha \neq 0$. If $\alpha > 0$, we have $\mathbf{v} \succ \overline{0}$, otherwise $\mathbf{v} \prec \overline{0}$.

(b) With the assumption and basic facts from linear algebra it follows that $(Id - A)$ has full rank and therefore $(Id - A)\mathbf{x} = \overline{1}$ has a unique solution $\mathbf{x}$. We still have to prove the second part of the conjunction:

($\Leftarrow$): Follows directly from Lemma 3.2(4).

($\Rightarrow$): Let $\rho(A) \leq 1$. Assume for a contradiction that $\rho(A) = 1$. Then, by Lemma 3.2(1), the matrix A would have an eigenvector $\mathbf{v} \neq \overline{0}$ with eigenvalue 1, so $(Id - A)\mathbf{v} = \overline{0}$, contradicting the assumption. So we have, in fact, $\rho(A) < 1$. By standard matrix facts (see e.g. [3]), this implies that $(Id - A)^{-1} = A^* = \sum_{i=0}^{\infty} A^i$ exists, and so we have $\mathbf{x} = (Id - A)^{-1}\overline{1} = A^*\overline{1} \geq \overline{1}$. Furthermore, $A\mathbf{x} = \sum_{i=1}^{\infty} A^i\overline{1} < \sum_{i=0}^{\infty} A^i\overline{1} = \mathbf{x}$. $\square$

In order to check whether $\rho(A) \leq 1$, we first solve the system $(Id - A)\mathbf{v} = \overline{0}$ using Gaussian elimination. If we find a vector $\mathbf{v} \neq \overline{0}$ such that $(Id - A)\mathbf{v} = \overline{0}$, we apply Lemma 3.3(a). If $\mathbf{v} = \overline{0}$ is the only solution of $(Id - A)\mathbf{v} = \overline{0}$, we solve $(Id - A)\mathbf{v} = \overline{1}$ using Gaussian elimination again, and apply Lemma 3.3(b). Since Gaussian elimination of a rational n-dimensional linear equation system can be carried out in strongly polynomial time using $O(n^3)$ arithmetic operations (see e.g. [7]), we obtain:

Proposition 3.4. *Given a nonnegative irreducible matrix $A \in \mathbb{R}^{n \times n}$, one can decide in strongly polynomial time, using $O(n^3)$ arithmetic operations, whether $\rho(A) \leq 1$.*

Combining Propositions 3.1 and 3.4 directly yields an algorithm for checking the consistency of scPSPs. Extending it to multiple SCCs as above, we get:

Theorem 3.5. *Let $f(X_1, \ldots, X_n)$ be a PSP. There is a strongly polynomial time algorithm that uses $O(n^3)$ arithmetic operations and determines the consistency of f.*

3.1 Case study: A family of "almost consistent" PSPs

In this section, we illustrate some issues faced by algorithms that solve the consistency problem. Consider the following family $h^{(n)}$ of scPSPs, $n \geq 2$:

$$h^{(n)} = \left(0.5X_1^2 + 0.1X_n^2 + 0.4 \text{ , } 0.01X_1^2 + 0.5X_2 + 0.49 \text{ , } \ldots \text{ ,} 0.01X_{n-1}^2 + 0.5X_n + 0.49 \right)^\top .$$

It is not hard to show that $h^{(n)}(\mathbf{p}) \prec \mathbf{p}$ holds for $\mathbf{p} = (1 - 0.02^n, \ldots, 1 - 0.02^{2n-1})^\top$, so we have $\mu_{h^{(n)}} \prec \overline{1}$ by Proposition 4.4, i.e., the $h^{(n)}$ are inconsistent.

	$n = 25$	$n = 100$	$n = 200$	$n = 400$	$n = 600$	$n = 1000$
Exact LP	< 1 sec	2 sec	8 sec	67 sec	208 sec	> 2h
Our algorithm	< 1 sec	< 1 sec	1 sec	4 sec	10 sec	29 sec

Table 1: Consistency checks for $h^{(n)}$-systems: Runtimes of different approaches.

The tool PReMo [11] relies on Java's floating-point arithmetic to compute approximations of the least fixed point of a PSP. We invoked PReMo for computing approximants of $\mu_{h^{(n)}}$ for different values of n between 5 and 100. Due to its fixed precision, PReMo's approximations for $\mu_{h^{(n)}}$ are greater than or equal to 1 in all components if $n \geq 7$. This might lead to the wrong conclusion that $h^{(n)}$ is consistent.

Recall that the consistency problem can be solved by checking the feasibility of the system (3.1) with $A = f'(\overline{1})$. We checked it with lp_solve, a well-known LP tool using hardware floating-point arithmetic. The tool wrongly states that (3.1) has no solution for $h^{(n)}$-systems with $n > 10$. This is due to the fact that the solutions cannot be represented adequately using machine number precision.[2] Finally, we also checked feasibility with Maple's Simplex package, which uses exact arithmetic, and compared its performance with the implementation, also in Maple, of our consistency algorithm. Table 1 shows the results. Our algorithm clearly outperforms the LP approach. For more experiments see Section 4.3.

4 Approximating μ_f with inexact arithmetic

It is shown in [6] that μ_f may not be representable by roots, so one can only approximate μ_f. In this section we present an algorithm that computes two sequences, $(\mathbf{lb}^{(i)})_i$ and $(\mathbf{ub}^{(i)})_i$, such that $\mathbf{lb}^{(i)} \leq \mu_f \leq \mathbf{ub}^{(i)}$ and $\lim_{i\to\infty} \mathbf{ub}^{(i)} - \mathbf{lb}^{(i)} = \overline{0}$. In words: $\mathbf{lb}^{(i)}$ and $\mathbf{ub}^{(i)}$ are lower and upper bounds on μ_f, respectively, and the sequences converge to μ_f. Moreover, they converge linearly, meaning that the *number of accurate bits* of $\mathbf{lb}^{(i)}$ and $\mathbf{ub}^{(i)}$ are linear functions of i. (The number of accurate bits of a vector $\mathbf{x}$ is defined as the greatest number k such that $|(\mu_f - \mathbf{x})_j|/|(\mu_f)_j| \leq 2^{-k}$ holds for all $j \in \{1, \ldots, n\}$.) These properties are guaranteed even though our algorithm uses inexact arithmetic: Our algorithm detects numerical problems due to rounding errors, recovers from them, and increases the precision of the arithmetic as needed. Increasing the precision dynamically is, e.g., supported by the GMP library [1].

Let us make precise what we mean by increasing the precision. Consider an elementary operation g, like multiplication, subtraction, etc., that operates on two input numbers x and y. We can *compute $g(x, y)$ with increasing precision* if there is a procedure that on input x, y outputs a sequence $g^{(1)}(x, y), g^{(2)}(x, y), \ldots$ that converges to $g(x, y)$. Note that there are no requirements on the convergence speed of this procedure — in particular, we do not require that there is an i with $g^{(i)}(x, y) = g(x, y)$. This procedure, which we assume exists, allows

[2]The mentioned problems of PReMo and lp_solve are not due to the fact that the coefficients of $h^{(n)}$ cannot be properly represented using basis 2: The problems persist if one replaces the coefficients of $h^{(n)}$ by similar numbers exactly representable by machine numbers.

to implement *floating assignments* of the form

$$z \leftsquigarrow g(x,y) \textbf{ such that } \phi(z)$$

with the following semantics: z is assigned the value $g^{(i)}(x,y)$, where $i \geq 1$ is the smallest index such that $\phi(g^{(i)}(x,y))$ holds. We say that the assignment is *valid* if $\phi(g(x,y))$ holds and ϕ involves only continuous functions and strict inequalities. Our assumption on the arithmetic guarantees that (the computation underlying) a valid floating assignment terminates. As "syntactic sugar", more complex operations (e.g., linear equation solving) are also allowed in floating assignments, because they can be decomposed into elementary operations.

We feel that any implementation of arbitrary precision arithmetic should satisfy our requirement that the computed values converge to the exact result. For instance, the documentation of the GMP library [1] states: "Each function is defined to calculate with 'infinite precision' followed by a truncation to the destination precision, but of course the work done is only what's needed to determine a result under that definition."

To approximate the least fixed point of a PSP, we first transform it into a certain normal form. A purely superlinear PSP f is called *perfectly superlinear* if every variable depends directly on itself and every superlinear SCC is purely superlinear. The following proposition states that any PSP f can be made perfectly superlinear.

Proposition 4.1. *Let f be a PSP of size s. We can compute in time $O(n \cdot s)$ a perfectly superlinear PSP $\tilde{f}$ with $Var(\tilde{f}) = Var(f) \cup \{\tilde{X}\}$ of size $O(n \cdot s)$ such that $\mu_f = (\mu_{\tilde{f}})_{Var(f)}$.*

4.1 The algorithm

The algorithm receives as input a perfectly superlinear PSP f and an error bound $\epsilon > 0$, and returns vectors $\mathbf{lb}, \mathbf{ub}$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$. A first initialization step requires to compute a vector $\mathbf{x}$ with $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$, i.e., a "strict" pre-fixed point. This is done in Section 4.1.1. The algorithm itself is described in Section 4.1.2.

4.1.1 Computing a strict pre-fixed point

Algorithm 1 computes a strict pre-fixed point:

Algorithm 1: Procedure `computeStrictPrefix`

```
Input: perfectly superlinear PSP $f$
Output: $\mathbf{x}$ with $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$
$\mathbf{x} \leftarrow \overline{0}$;
while $\overline{0} \not\prec \mathbf{x}$ do
    $Z \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{x}) = 0\}$;
    $P \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{x}) > 0\}$;
    $\mathbf{y}_Z \leftarrow \overline{0}$;
    $\mathbf{y}_P \leftsquigarrow f_P(\mathbf{x})$ such that $\overline{0} \prec \mathbf{y}_P \prec f_P(\mathbf{y}) \prec \overline{1}$;
    $\mathbf{x} \leftarrow \mathbf{y}$;
```

Proposition 4.2. *Algorithm 1 is correct and terminates after at most n iterations.*

The reader may wonder why Algorithm 1 uses a floating assignment $\mathbf{y}_P \leftsquigarrow f_P(\mathbf{x})$, given that it must also perform exact comparisons to obtain the sets Z and P and to decide exactly whether $\mathbf{y}_P \prec f_P(\mathbf{y})$ holds in the **such that** clause of the floating assignment. The reason is that, while we perform such operations exactly, we do not want to use the *result* of exact computations as input for other computations, as this easily leads to an explosion in the required precision. For instance, the size of the exact result of $f_P(\mathbf{y})$ may be larger than the size of $\mathbf{y}$, while an approximation of smaller size may already satisfy the **such that** clause. In order to emphasize this, we *never* store the result of an exact numerical computation in a variable.

4.1.2 Computing lower and upper bounds

Algorithm 1 uses Kleene iteration $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$ to compute a strict pre-fixed point. One could, in principle, use the same scheme to compute lower bounds of μ_f, as this sequence converges to μ_f from below by Kleene's theorem. However, convergence of Kleene iteration is generally slow. It is shown in [6] that for the 1-dimensional PSP f with $f(X) = 0.5X^2 + 0.5$ we have $\mu_f = 1$, and the i-th Kleene approximant $\boldsymbol{\kappa}^{(i)}$ satisfies $\boldsymbol{\kappa}^{(i)} \leq 1 - \frac{1}{i}$. Hence, Kleene iteration may converge only logarithmically, i.e., the number of accurate bits is a logarithmic function of the number of iterations.

In [6] it was suggested to use Newton's method for faster convergence. In order to see how Newton's method can be used, observe that instead of computing μ_f, one can equivalently compute the least nonnegative zero of $f(\overline{X}) - \overline{X}$. Given an approximant $\mathbf{x}$ of μ_f, Newton's method first computes $g^{(\mathbf{x})}(\overline{X})$, the first-order linearization of f at the point $\mathbf{x}$:

$$g^{(\mathbf{x})}(\overline{X}) = f(\mathbf{x}) + f'(\mathbf{x})(\overline{X} - \mathbf{x})$$

The next Newton approximant $\mathbf{y}$ is obtained by solving $\overline{X} = g^{(\mathbf{x})}(\overline{X})$, i.e.,

$$\mathbf{y} = \mathbf{x} + (Id - f'(\mathbf{x}))^{-1}(f(\mathbf{x}) - \mathbf{x}) \, .$$

We write $\mathcal{N}_f(\mathbf{x}) := \mathbf{x} + (Id - f'(\mathbf{x}))^{-1}(f(\mathbf{x}) - \mathbf{x})$, and usually drop the subscript of $\mathcal{N}_f$. If $\boldsymbol{\nu}^{(0)} \leq \mu_f$ is any pre-fixed point of f, for instance $\boldsymbol{\nu}^{(0)} = \overline{0}$, we can define a *Newton sequence* $(\boldsymbol{\nu}^{(i)})_i$ by setting $\boldsymbol{\nu}^{(i+1)} = \mathcal{N}(\boldsymbol{\nu}^{(i)})$ for $i \geq 0$. It has been shown in [6, 9, 4] that Newton sequences converge at least linearly to μ_f. Moreover, we have $\overline{0} \leq \boldsymbol{\nu}^{(i)} \leq f(\boldsymbol{\nu}^{(i)}) \leq \mu_f$ for all i.

These facts were shown only for Newton sequences that are computed exactly, i.e., without rounding errors. Unfortunately, Newton approximants are hard to compute exactly: Since each iteration requires to solve a linear equation system whose coefficients depend on the results of the previous iteration, the size of the Newton approximants easily explodes. Therefore, we wish to use inexact arithmetic, but without losing the good properties of Newton's method (reliable lower bounds, linear convergence).

Algorithm 2 accomplishes these goals, and additionally computes post-fixed points $\mathbf{ub}$ of f, which are upper bounds on μ_f. Let us describe the algorithm in some detail. The lower bounds are stored in the variable $\mathbf{lb}$. The first

Algorithm 2: Procedure `calcBounds`

```
   Input: perfectly superlinear PSP $f$, error bound $\epsilon > 0$
   Output: vectors $\mathbf{lb}, \mathbf{ub}$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$
1  $\mathbf{lb} \leftarrow$ computeStrictPrefix($f$);
2  $\mathbf{ub} \leftarrow \overline{1}$;
3  while $\mathbf{ub} - \mathbf{lb} \not\leq \overline{\epsilon}$ do
4      $\mathbf{x} \leftsquigarrow \mathcal{N}(\mathcal{N}(\mathbf{lb}))$ such that $f(\mathbf{lb}) + f'(\mathbf{lb})(\mathbf{x} - \mathbf{lb}) \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$;
5      $\mathbf{lb} \leftarrow \mathbf{x}$;
6      $Z \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{ub}) = 1\}$;
7      $P \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{ub}) < 1\}$;
8      $\mathbf{y}_Z \leftarrow \overline{1}$;
9      $\mathbf{y}_P \leftsquigarrow f_P(f(\mathbf{ub}))$ such that $f_P(\mathbf{y}) \prec \mathbf{y}_P \prec f_P(\mathbf{ub})$;
10     forall superlinear SCCs $S$ of $f$ with $\mathbf{y}_S = \overline{1}$ do
11         $\mathbf{t} \leftarrow \overline{1} - \mathbf{lb}_S$;
12         if $f'_{SS}(\overline{1})\mathbf{t} \succ \mathbf{t}$ then
13             $\mathbf{y}_S \leftsquigarrow \overline{1} - \min\left\{1, \frac{\min_{i \in S}(f'_{SS}(\overline{1})\mathbf{t} - \mathbf{t})_i}{2 \cdot \max_{i \in S}(f_S(\overline{2}))_i}\right\} \cdot \mathbf{t}$ such that
               $f_S(\mathbf{y}) \prec \mathbf{y}_S \prec \overline{1}$;
14     $\mathbf{ub} \leftarrow \mathbf{y}$;
```

value of $\mathbf{lb}$ is not simply $\overline{0}$, but is computed by `computeStrictPrefix`(f), in order to guarantee the validity of the following floating assignments. We use Newton's method for improving the lower bounds because it converges fast (at least linearly) when performed exactly. In each iteration of the algorithm, *two* Newton steps are performed using inexact arithmetic. The intention is that two inexact Newton steps should improve the lower bound at least as much as one exact Newton step. While this may sound like a vague hope for small rounding errors, it can be rigorously proved thanks to the **such that** clause of the floating assignment in line 4. The proof involves two steps. The first step is to prove that $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$ is a (strict) post-fixed point of the function $g(\overline{X}) = f(\mathbf{lb}) + f'(\mathbf{lb})(\overline{X} - \mathbf{lb})$, i.e., $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$ satisfies the first inequality in the **such that** clause. For the second step, recall that $\mathcal{N}(\mathbf{lb})$ is the least fixed point of g. By Knaster-Tarski's theorem, $\mathcal{N}(\mathbf{lb})$ is actually the least post-fixed point of g. So, our value $\mathbf{x}$, the inexact version of $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$, satisfies $\mathbf{x} \geq \mathcal{N}(\mathbf{lb})$, and hence two inexact Newton steps are in fact at least as "fast" as one exact Newton step. Thus, the $\mathbf{lb}$ converge linearly to μ_f.

The upper bounds $\mathbf{ub}$ are post-fixed points, i.e., $f(\mathbf{ub}) \leq \mathbf{ub}$ is an invariant of the algorithm. The algorithm computes the sets Z and P so that inexact arithmetic is only applied to the components i with $f_i(\mathbf{ub}) < 1$. In the P-components, the function f is applied to $\mathbf{ub}$ in order to improve the upper bound. In fact, f is applied twice in line 9, analogously to applying $\mathcal{N}$ twice in line 4. Here, the **such that** clause makes sure that the progress towards μ_f is at least as fast as the progress of one exact application of f would be. One can show that this leads to linear convergence to μ_f.

The rest of the algorithm (lines 10-13) deals with the problem that, given a post-fixed $\mathbf{ub}$, the sequence $\mathbf{ub}, f(\mathbf{ub}), f(f(\mathbf{ub})), \ldots$ does not necessarily converge to μ_f. For instance, if $f(X) = 0.75X^2 + 0.25$, then $\mu_f = 1/3$, but $1 = f(1) = f(f(1)) = \cdots$. Therefore, the if-statement of Algorithm 2 allows to improve the upper bound from $\overline{1}$ to a post-fixed point less than $\overline{1}$, by exploiting the lower bounds **lb**. This is illustrated in Figure 1 for a 2-dimensional scPSP f. The dotted lines indicate the curve of the points (X_1, X_2) satis-

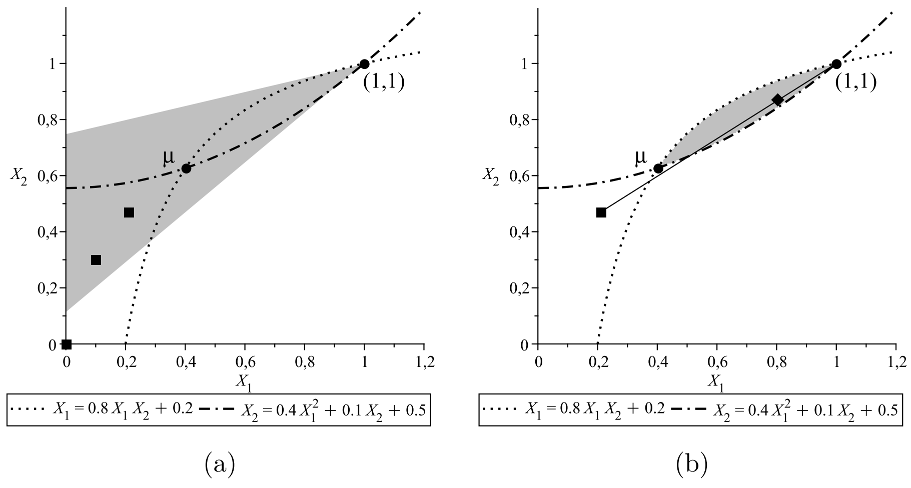

Figure 1: Computation of a post-fixed point less than $\overline{1}$

fying $X_1 = 0.8X_1X_2 + 0.2$ and $X_2 = 0.4X_1^2 + 0.1X_2 + 0.5$. Notice that $\mu_f \prec \overline{1} = f(\overline{1})$. In Figure 1 (a) the shaded area consists of those points **lb** where $f'(\overline{1})(\overline{1} - \mathbf{lb}) \succ \overline{1} - \mathbf{lb}$ holds, i.e., the condition of line 12. One can show that μ_f must lie in the shaded area, so by continuity, any sequence converging to μ_f, in particular the sequence of lower bounds **lb**, finally reaches the shaded area. In Figure 1 (a) this is indicated by the points with the square shape. Figure 1 (b) shows how to exploit such a point **lb** to compute a post-fixed point $\mathbf{ub} \prec \overline{1}$ (post-fixed points are shaded in Figure 1 (b)): The post-fixed point **ub** (diamond shape) is obtained by starting at $\overline{1}$ and moving a little bit along the straight line between $\overline{1}$ and **lb**, cf. line 13. The sequence $\mathbf{ub}, f(\mathbf{ub}), f(f(\mathbf{ub})), \ldots$ now converges linearly to μ_f.

Theorem 4.3. *Algorithm 2 terminates and computes vectors* $\mathbf{lb}, \mathbf{ub}$ *such that* $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ *and* $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$. *Moreover, the sequences of lower and upper bounds computed by the algorithm both converge linearly to* μ_f.

Notice that Theorem 4.3 is about the convergence speed of the approximants, not about the time needed to compute them. To analyse the computation time, one would need stronger requirements on how floating assignments are performed.

The lower and upper bounds computed by Algorithm 2 have a special feature: they satisfy $\mathbf{lb} \prec f(\mathbf{lb})$ and $\mathbf{ub} \geq f(\mathbf{ub})$. The following proposition guarantees that such points are in fact lower and upper bounds.

Proposition 4.4. *Let f be a perfectly superlinear PSP. Let* $\overline{0} \leq \mathbf{x} \leq \overline{1}$. *If* $\mathbf{x} \prec f(\mathbf{x})$, *then* $\mathbf{x} \prec \mu_f$. *If* $\mathbf{x} \geq f(\mathbf{x})$, *then* $\mathbf{x} \geq \mu_f$.

So a user of Algorithm 2 can immediately verify that the computed bounds are correct. To summarize, Algorithm 2 computes provably and even verifiably correct lower and upper bounds, although exact computation is restricted to detecting numerical problems. See Section 4.3 for experiments.

4.2 Proving consistency using the inexact algorithm

In Section 3 we presented a simple and efficient algorithm to check the consistency of a PSP. Algorithm 2 is aimed at approximating μ_f, but note that it can also prove the inconsistency of a PSP: when the algorithm sets $\mathbf{ub}_i < 1$, we know $(\mu_f)_i < 1$. This raises the question whether Algorithm 2 can also be used for proving consistency. The answer is yes, and the procedure is based on the following proposition.

Proposition 4.5. *Let f be an scPSP. Let $\mathbf{t} \succ \overline{0}$ be a vector with $f'(\overline{1})\mathbf{t} \leq \mathbf{t}$. Then f is consistent.*

Proposition 4.5 can be used to identify consistent components.

Use Algorithm 2 with some (small) ϵ to compute $\mathbf{ub}$ and $\mathbf{lb}$. Take any bottom SCC S.

- If $f'(\overline{1})(\overline{1} - \mathbf{lb}_S) \leq \overline{1} - \mathbf{lb}_S$, mark all variables in S as consistent and remove the S-components from f. In the remaining components, replace all variables in S with 1.
- Otherwise, remove S and all other variables that depend on S from f.

Repeat with the new bottom SCC until all SCCs are processed.
There is no guarantee that this method detects all i with $(\mu_f)_i = 1$.

4.3 Case study: A neutron branching process

One of the main applications of the theory of branching processes is the modelling of cascade creation of particles in physics. We study a problem described by Harris in [8]. Consider a ball of fissionable radioactive material of radius D. Spontaneous fission of an atom can liberate a neutron, whose collision with another atom can produce further neutrons etc. If D is very small, most neutrons leave the ball without colliding. If D is very large, then nearly all neutrons eventually collide, and the probability that the neutron's progeny never dies is large. A well-known result shows that, loosely speaking, the population of a process that does not go extinct grows exponentially over time with large probability. Therefore, the neutron's progeny never dying out actually means that after a (very) short time all the material is fissioned, which amounts to a nuclear explosion. The task is to compute the largest value of D for which the probability of extinction of a neutron born at the centre of the ball is still 1 (if the probability is 1 at the centre, then it is 1 everywhere). This is often called the critical radius. Notice that, since the number of atoms that undergo spontaneous fission is large (some hundreds per second for the critical radius of plutonium), if the probability of extinction lies only slightly below 1, there is already a large probability of a chain reaction. Assume that a neutron born at distance ξ from the centre leaves the ball without colliding with probability $l(\xi)$, and collides with an atom at distance η from the centre with probability density

D	2			3			6			10		
n	20	50	100	20	50	100	20	50	100	20	50	100
inconsistent (yes/no)	n	n	n	y	y	y	y	y	y	y	y	y
Cons. check (Alg. Sec. 3)	< 1	< 1	2	< 1	< 1	2	< 1	< 1	2	< 1	< 1	2
Cons. check (exact LP)	< 1	20	258	< 1	22	124	< 1	16	168	< 1	37	222
Approx. Q_D ($\epsilon = 10^{-3}$)	< 1	< 1	4	2	8	32	1	5	21	1	4	17
Approx. Q_D ($\epsilon = 10^{-4}$)	< 1	< 1	4	2	8	34	2	7	28	1	6	23

Table 2: Runtime in seconds of various algorithms on different values of D and n.

$R(\xi, \eta)$. Let further $f(x) = \sum_{i \geq 0} p_i x^i$, where p_i is the probability that a collision generates i neutrons. For a neutron's progeny to go extinct, the neutron must either leave the ball without colliding, or collide at some distance η from the centre, but in such a way that the progeny of all generated neutrons goes extinct. So the extinction probability $Q_D(\xi)$ of a neutron born at distance ξ from the centre is given by [8], p. 86:

$$Q_D(\xi) = l(\xi) + \int_0^D R(\xi, \eta) f(Q_D(\eta))\, d\eta$$

Harris takes $f(x) = 0.025 + 0.830x + 0.07x^2 + 0.05x^3 + 0.025x^4$, and gives expressions for both $l(\xi)$ and $R(\xi, \eta)$. By discretizing the interval $[0, D]$ into n segments and replacing the integral by a finite sum we obtain a PSP of dimension $n + 1$ over the variables $\{Q_D(jD/n) \mid 0 \leq j \leq n\}$. Notice that $Q_D(0)$ is the probability that a neutron born in the centre does not cause an explosion.

Results For our experiments we used three different discretizations $n = 20, 50, 100$. We applied our consistency algorithm from Section 3 and Maple's Simplex to check inconsistency, i.e., to check whether an explosion occurs. The results are given in the first 3 rows of Table 2: Again our algorithm dominates the LP approach, although the polynomials are much denser than in the $h^{(n)}$-systems.

We also implemented Algorithm 2 using Maple for computing lower and upper bounds on $Q_D(0)$ with two different values of the error bound ϵ. The runtime is given in the last two rows. By setting the *Digits* variable in Maple we controlled the precision of Maple's software floating-point numbers for the floating assignments. In all cases starting with the standard value of 10, Algorithm 2 increased *Digits* at most twice by 5, resulting in a maximal *Digits* value of 20. The numerical results, plotted in Figure 2, fit in well with the approximations given in [8].

As a side note we mention that Algorithm 2 computed an upper bound $\prec \overline{1}$, and thus proved inconsistency, after the first few iterations in all investigated cases, almost as fast as the consistency algorithm from Section 3.

Computing approximations for the critical radius. From the data displayed in Figure 2 one can suspect that the critical radius, i.e., the smallest

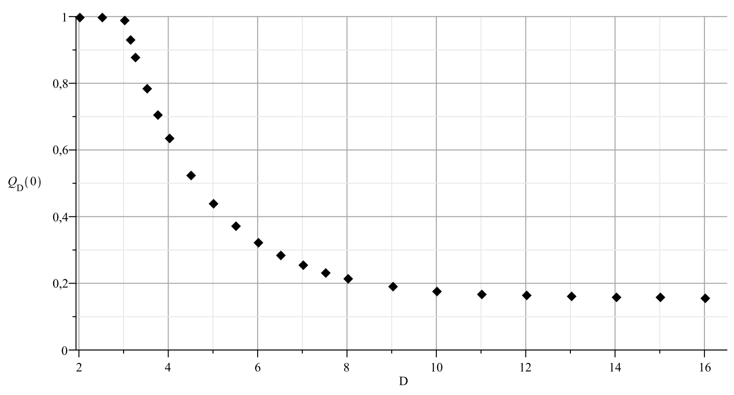

Figure 2: $Q_D(0)$ for different values of D, $n = 100$.

value of D for which $Q_D(0) = 1$, lies somewhere between 2.7 and 3. We combined binary search with the consistency algorithm from Section 3 to determine the critical radius up to an error of 0.01. During the binary search, the algorithm from Section 3 has to analyze PSPs that come closer and closer to the verge of (in)consistency. For the last (and most expensive) binary search step that decreases the interval to 0.01, our algorithm took <1, 1, 3, 8 seconds for $n = 20, 50, 100, 150$, respectively. For $n = 150$, we found the critical radius to be in the interval $[2.981, 2.991]$. Harris [8] estimates 2.9.

5 Conclusions

We have presented a new, simple, and efficient algorithm for checking the consistency of PSPs, which outperforms the previously existing LP-based method. We have also described the first algorithm that computes reliable lower and upper bounds on μ_f. The sequence of bounds converges linearly to μ_f. To achieve these properties without sacrificing efficiency, we use a novel combination of exact and inexact (floating-point) arithmetic. Experiments on PSPs from concrete branching processes confirm the practicality of our approach. The results raise the question whether our combination of exact and inexact arithmetic could be transferred to other computational problems.

Acknowledgments

We thank several anonymous referees for pointing out inaccuracies and helping us clarify certain aspects of the paper. The second author was supported by the DFG Graduiertenkolleg 1480 (PUMA).

References

[1] GMP library. http://gmplib.org.

[2] K. B. Athreya and P. E. Ney. *Branching Processes.* Springer, 1972.

[3] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences.* SIAM, 1994.

[4] J. Esparza, S. Kiefer, and M. Luttenberger. Convergence thresholds of Newton's method for monotone polynomial equations. In *Proceedings of STACS*, pages 289–300, 2008.

[5] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS 2004*, pages 12–21. IEEE Computer Society, 2004.

[6] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009.

[7] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer, 1993.

[8] T. E. Harris. *The theory of branching processes.* Springer, Berlin, 1963.

[9] S. Kiefer, M. Luttenberger, and J. Esparza. On the convergence of Newton's method for monotone systems of polynomial equations. In *Proceedings of STOC*, pages 217–226. ACM, 2007.

[10] C. D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing.* MIT Press, June 1999.

[11] D. Wojtczak and K. Etessami. PReMo: an analyzer for probabilistic recursive models. In *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 66–71. Springer, 2007.

A Proofs of Section 4

Here is a restatement of Proposition 4.1.

Proposition 4.1. *Let f be a PSP of size s. We can compute in time $O(n \cdot s)$ a perfectly superlinear PSP $\tilde{f}$ with $Var(\tilde{f}) = Var(f) \cup \{\tilde{X}\}$ of size $O(n \cdot s)$ such that $\mu_f = (\mu_{\tilde{f}})_{Var(f)}$.*

Proof. In a first step, we add to the equation system $\overline{X} = f(\overline{X})$ an $(n+1)$-st equation $\tilde{X} = \frac{1}{3}\tilde{X}^2 + \frac{2}{3}$. It is easy to see that the least solution of this equation is $\tilde{X} = 1$. In order to make f purely superlinear, we take all components f_i that are not yet superlinear and multiply a monomial of f_i by $\tilde{X}$. For instance, if $f_i = \frac{1}{4}X_j + \frac{1}{3}X_k$, then we replace f_i with $\frac{1}{4}X_j\tilde{X} + \frac{1}{3}X_k$. This transformation does not change the least fixed point in the non-$\tilde{X}$-components. Call the resulting PSP again f for simplicity. Notice that f is now purely superlinear.

In a second step we make sure that all superlinear SCCs are purely superlinear. For this, we repeatedly apply a certain operation: Let X_j be a variable that occurs in a monomial m of a component f_i, i.e., there is a monomial $\tilde{m}$ with $m = X_j \cdot \tilde{m}$. The operation that replaces the monomial m in f_i with $0.5 \cdot m + 0.5 \cdot f_j \cdot \tilde{m}$ is called *substituting* (an occurrence of) X_j. It is easy to see that applying this operation to a PSP yields a PSP with the same set of fixed points. Notice that substituting does not change the dependency relation between the variables.

In order to make all superlinear SCCs purely superlinear, we apply a sequence of substituting operations. Take a superlinear SCC S which is not purely superlinear and let $g(S)$ denote the PSP obtained by restricting f to the S-components and replacing all variables which are not in S by the constant 1. Since S is superlinear and not purely superlinear, the PSP $g(S)$ is, by definition, superlinear and not purely superlinear. So there exist variables $X_i, X_j \in S$ such that X_i directly depends on X_j in $g(S)$, and $g(S)_i$ is linear, and $g(S)_j$ is superlinear. Substitute the corresponding occurrence of X_j in f_i. This makes $g(S)_i$ superlinear. By proceeding this way, at most n substituting operations suffice to make all superlinear SCCs purely superlinear.

To make f perfectly superlinear, it remains to make each variable directly depend on itself. We achieve that by replacing, for all variables X, the polynomial f_X with $0.5 f_X + 0.5X$. It is easy to see that f has the same least fixed point, the sum of the coefficients is still at most 1 in all components, and no new variable dependencies are created by this operation except that every variable now depends directly on itself. So, this operation makes f perfectly superlinear.

Clearly, the bottleneck of this whole procedure consists of the substituting operations. Notice that computing the DAG of SCCs can be done in time $O(s)$ with Tarjan's algorithm. The size of each single polynomial at the end of the substituting procedure is $O(s)$, so the total size of the resulting PSP is $O(n \cdot s)$. ☐

Here is a restatement of Proposition 4.2.

Proposition 4.2. *Algorithm 1 is correct and terminates after at most n iterations.*

Proof. We will prove the following invariant of the algorithm:

(a) $\overline{0} \leq \mathbf{x} \leq f(\mathbf{x})$;

(b) for all components j with $(f^i(\overline{0}))_j > 0$ we have $0 < \mathbf{x}_j < f_j(\mathbf{x})$.

The invariant implies that the loop terminates after at most n iterations, because $f^n(\overline{0}) \succ \overline{0}$ holds as $\mu_f \succ \overline{0}$. The invariant also implies that we have $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$ after the loop terminates.

So it remains to show the invariant. Part (a) clearly holds throughout the loop because for all components j either $0 = \mathbf{x}_j$ holds or $0 < \mathbf{x}_j < f_j(\mathbf{x})$ is guaranteed by the floating assignment. Assume inductively that the invariant holds after $i \geq 0$ iterations. It suffices to prove that (b) holds after $i+1$ iterations. Let $\mathbf{x}^{(i)}$ denotes the value of $\mathbf{x}$ after i iterations. Let $(f^{i+1}(\overline{0}))_j > 0$. Then $f_j(\overline{0}) > 0$ or there is a monomial in f_j which consists only of variables X_k with $(f^i(\overline{0}))_k > 0$. In the second case we have, by induction hypothesis part (b), that $\mathbf{x}_k^{(i)} > 0$ holds for those variables X_k. In both cases it follows $f_j(\mathbf{x}^{(i)}) > 0$. Furthermore, we have by induction hypothesis part (a) that $f_j(\mathbf{x}^{(i)}) \leq f_j(f(\mathbf{x}^{(i)}))$ where, in fact, the inequality is strict because X_j depends on itself (as f is perfectly superlinear). We conclude that $0 < f_j(\mathbf{x}^{(i)}) < f_j(f(\mathbf{x}^{(i)}))$, and hence the floating assignment guarantees $0 < \mathbf{x}_j^{(i+1)} < f_j(\mathbf{x}^{(i+1)})$. So the invariant holds after $i+1$ iterations. $\square$

A.1 Proof of Proposition 4.4

For the proof of Proposition 4.4 we need some auxiliary lemmas that will also used later on for Theorem 4.3. We start with the following simple lemma on linear PSPs.

Lemma A.1. *Let f be a linear PSP. Then μ_f is the unique fixed point of f.*

Proof. Recall that $\mu_f \succ \overline{0}$. Assume that there is a fixed point $\mathbf{x}$ of f different from μ_f. Then, by the linearity of f, all points on the straight line through $\mathbf{x}$ and μ_f are fixed points of f. So there is a point $\mathbf{y} \geq \overline{0}$ on this straight line with $\mathbf{y}_i = 0$ for some $i \in \{1, \ldots, n\}$. This contradicts the fact that μ_f is the *least* fixed point of f. $\square$

The following lemma shows how to compute a post-fixed point that satisfies certain properties and is arbitrarily close to $\overline{1}$.

Lemma A.2. *Let f be a perfectly superlinear PSP. Let $r \in \mathbb{R}$ with $0 < r < 1$. Let $\mathbf{x} = \mu_f + r(\overline{1} - \mu_f)$. Then $f(\mathbf{x}) \leq \mathbf{x}$. Furthermore, let $\mathbf{p} = f^n(\mathbf{x})$. Then $f(\mathbf{p}) \leq \mathbf{p}$ and $f_i(\mathbf{p}) < \mathbf{p}_i$ holds for all $i \in \{1, \ldots, n\}$ with $(\mu_f)_i < 1$.*

Proof. Letting $\mathbf{u}, \mathbf{v}$ be any vectors, we write $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. Then we have

$$\mu_f + f'(\mu_f)(\overline{1} - \mu_f) + R(\mu_f, \overline{1} - \mu_f) = f(\mu_f + (\overline{1} - \mu_f)) = f(\overline{1}) \leq \overline{1} = \mu_f + (\overline{1} - \mu_f) ,$$

so it follows $f'(\mu_f)\mathbf{d} + R(\mu_f, \mathbf{d}) \leq \mathbf{d}$ where $\mathbf{d} := \overline{1} - \mu_f$. Moreover, we have

$$\begin{aligned}
f(\mathbf{x}) &= f(\mu_f + r\mathbf{d}) = f(\mu_f) + f'(\mu_f)r\mathbf{d} + R(\mu_f, r\mathbf{d}) \\
&= \mu_f + rf'(\mu_f)\mathbf{d} + R(\mu_f, r\mathbf{d}) \\
&\leq \mu_f + rf'(\mu_f)\mathbf{d} + rR(\mu_f, \mathbf{d}) && \text{(since } R(\mu_f, \cdot) \text{ is superlinear)} \\
&= \mu_f + r(f'(\mu_f)\mathbf{d} + R(\mu_f, \mathbf{d})) \\
&\leq \mu_f + r\mathbf{d} && \text{(from above)} \\
&= \mathbf{x}\ ,
\end{aligned}$$

i.e., $\mathbf{x}$ is a post-fixed point, hence $\mu_f \leq \mathbf{x} \leq \overline{1}$. Consider the sequence $\mathbf{x} \geq f(\mathbf{x}) \geq f(f(\mathbf{x})) \geq \cdots$. Since every component depends directly on itself, we have for all components i that once $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$ holds for some j, we have $(f^k(\mathbf{x}))_i > (f^{k+1}(\mathbf{x}))_i$ for all $k \geq j$. On the other hand, it is easy to see that if $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$ for some j, then $(f^k(\mathbf{x}))_i > (f^{k+1}(\mathbf{x}))_i$ for some $k \leq n$. It follows that $f_i(\mathbf{p}) < \mathbf{p}_i$ holds for all components i for which there exists j with $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$. It remains to show that for all components i with $(\mu_f)_i < 1$ there exists j with $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$. Assume for a contradiction that this does not hold. Choose a variable X_i that violates the property (i.e., $(\mu_f)_i < 1$ and $\mathbf{x}_i = (f^j(\mathbf{x}))_i$ for all j) so that all components in lower SCCs satisfy the property. Then for all X_j on which X_i depends we have $(\mu_f)_j = 1$, and so $\mathbf{x}_j = 1$. Furthermore, we have $(\mu_f)_S \prec \mathbf{x}_S \prec \overline{1}$. Let S denote the SCC of X_i and let g be the PSP obtained by restricting f to the S-components and replacing all variables in lower SCCs by the constant 1. Notice that Lemma A.1 guarantees that g is not linear, since both $(\mu_f)_S$ and $\mathbf{x}_S$ are fixed points of g. Hence, g is superlinear. For any vectors $\mathbf{u}, \mathbf{v}$ we write $g(\mathbf{u}+\mathbf{v}) = g(\mathbf{u}) + g'(\mathbf{u})\mathbf{v} + T(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of g at $\mathbf{u}$. We have:

$$\begin{aligned}
r\mathbf{d}_S &= \mathbf{x}_S - (\mu_f)_S \\
&= g(\mathbf{x}_S) - (\mu_f)_S \\
&= g((\mu_f)_S + r\mathbf{d}_S) - (\mu_f)_S \\
&= (\mu_f)_S + g'((\mu_f)_S)r\mathbf{d}_S + T((\mu_f)_S, r\mathbf{d}_S) - (\mu_f)_S \\
&= g'((\mu_f)_S)r\mathbf{d}_S + T((\mu_f)_S, r\mathbf{d}_S)
\end{aligned}$$

Moreover, as $\mathbf{d}_S \succ \overline{0}$ and g is superlinear, the following inequality is strict in at least one component:

$$\begin{aligned}
g(\overline{1}) &= g\left((\mu_f)_S + \frac{1}{r}r\mathbf{d}\right) \\
&= g((\mu_f)_S) + g'((\mu_f)_S)\frac{1}{r}r\mathbf{d}_S + T\left((\mu_f)_S, \frac{1}{r}r\mathbf{d}\right) \\
&\geq (\mu_f)_S + \frac{1}{r}g'((\mu_f)_S)r\mathbf{d}_S + \frac{1}{r^2}T\left((\mu_f)_S, r\mathbf{d}\right) \\
&> (\mu_f)_S + \frac{1}{r}\left(g'((\mu_f)_S)r\mathbf{d}_S + T\left((\mu_f)_S, r\mathbf{d}\right)\right) \\
&= (\mu_f)_S + \frac{1}{r}r\mathbf{d}_S && \text{(as computed above)} \\
&= (\mu_f)_S + \mathbf{d}_S = \overline{1}
\end{aligned}$$

This is the desired contradiction as $g(\overline{1}) \leq \overline{1}$ should hold since g is a PSP. $\square$

The following lemma is used for the proof of Proposition 4.4, but will also be essential to prove the convergence statement of Theorem 4.3.

Lemma A.3. *Let f be a perfectly superlinear PSP. Let $f(\mathbf{p}) \le \mathbf{p} \le \overline{1}$ and $f_i(\mathbf{p}) < \mathbf{p}_i$ for all $i \in \{1, \ldots, n\}$ with $(\mu_f)_i < 1$. Then the sequence $(\mathbf{p}^{(i)})_{i\in\mathbb{N}}$ defined by*

$$\mathbf{p}^{(0)} := \mathbf{p} \text{ and } \mathbf{p}^{(i+1)} := f(\mathbf{p}^{(i)}) \text{ for } i \ge 0$$

converges linearly to μ_f.

Proof. If $(\mu_f)_i = 1$ then $(\mu_f)_j = 1$ has to hold for every component j on which i depends. As μ_f is the least post-fixed point by Knaster-Tarski's theorem, we have $f_i^k(\mathbf{p}) = 1$ for every $k \in \mathbb{N}$. Hence we can ignore the 1-components in our convergence proof and assume w.l.o.g. that $\mu_f \prec \overline{1}$ and with the assumptions $f(\mathbf{p}) \prec \mathbf{p}$. By the monotonicity of f and as every variable depends on itself, we get by a simple induction that $\mathbf{p}^{(i)} \succ \mathbf{p}^{(i+1)} \succ \mu_f$ for all $i \in \mathbb{N}$. This already shows that $(\mathbf{p}^{(i)})$ converges to some limit point.

For every $\mathbf{u} \succ \mu_f$ with $\mathbf{u} \succ f(\mathbf{u})$ we write $\mathbf{u} = \mu_f + \mathbf{\Delta_u}$ and get:

$$\begin{aligned} f(\mathbf{u}) - \mu_f &= f(\mu_f + \mathbf{\Delta_u}) - \mu_f \\ &= f(\mu_f) + f'(\mu_f)\mathbf{\Delta_u} + R(\mu_f, \mathbf{\Delta_u}) - \mu_f \qquad \text{(Taylor expansion)} \end{aligned}$$

Since $R(\mu_f, \mathbf{\Delta_u})$ depends at least quadratically on $\mathbf{\Delta_u}$, one can write $R(\mu_f, \mathbf{\Delta_u}) = \tilde{R}(\mu_f, \mathbf{\Delta_u}) \cdot \mathbf{\Delta_u}$ for a nonnegative matrix $\tilde{R}(\mu_f, \mathbf{\Delta_u})$. Continuing the above equaliy, we obtain:

$$\begin{aligned} &= (f'(\mu_f) + \tilde{R}(\mu_f, \mathbf{\Delta_u}))\mathbf{\Delta_u} \\ &\prec \mathbf{\Delta_u} \qquad \text{(as } \mathbf{u} \succ f(\mathbf{u}).\text{)} \end{aligned}$$

Define $A(\mathbf{u}) := f'(\mu_f) + \tilde{R}(\mu_f, \mathbf{\Delta_u})$, so that we obtain

$$f(\mathbf{u}) - \mu_f = A(\mathbf{u}) \cdot \mathbf{\Delta_u} \prec \mathbf{\Delta_u}. \tag{A.1}$$

This holds especially for $\mathbf{u} = \mathbf{p}$. From the $\prec$-inequality in (A.1) follows that there exists $0 < \delta < 1$ such that

$$f(\mathbf{p}) - \mu_f = A(\mathbf{p})\mathbf{\Delta_p} \le \delta\mathbf{\Delta_p}. \tag{A.2}$$

We now show for every $i \ge 0$ that $\mathbf{p}^{(i)} - \mu_f = \mathbf{\Delta}_{\mathbf{p}^{(\mathbf{i})}} \le \delta^i \mathbf{\Delta_p}$ by induction over i. This implies the linear convergence of $(\mathbf{p}^{(i)})_{i\in\mathbb{N}}$. The base case $i = 1$ is proved by (A.2). For $i > 1$ note that if $\overline{0} \le \mathbf{u} \le \mathbf{u}'$ and $\overline{0} \le \mathbf{v} \le \mathbf{v}'$, we have $A(\mathbf{u})\mathbf{v} \le A(\mathbf{u}')\mathbf{v}'$, since $A(\mathbf{u})$ is nonnegative if $\mathbf{u}$ is nonnegative.

$$\begin{aligned} f^i(\mathbf{p}) - \mu_f &= f(\mathbf{p}^{(i-1)}) - \mu_f \\ &= A(\mathbf{p}^{(i-1)})(\mathbf{p}^{(i-1)} - \mu_f) && \text{(by (A.1))} \\ &\le A(\mathbf{p})(\delta^{i-1}\mathbf{\Delta_p}) && \text{(induction hypothesis)} \\ &= \delta^{i-1} A(\mathbf{p})\mathbf{\Delta_p} \\ &\le \delta^i \mathbf{\Delta_p}. && \text{(A.2)} \end{aligned}$$

□

Now we can prove Proposition 4.4 which we restate here.

Proposition 4.4. *Let f be a perfectly superlinear PSP. Let $\overline{0} \leq \mathbf{x} \leq \overline{1}$. If $\mathbf{x} \prec f(\mathbf{x})$, then $\mathbf{x} \prec \mu_f$. If $\mathbf{x} \geq f(\mathbf{x})$, then $\mathbf{x} \geq \mu_f$.*

Proof. By Knaster-Tarski's theorem, μ_f is the least post-fixed point; the final statement of the proposition follows. It remains to show the first statement. By choosing the number r from Lemma A.2 large enough we can find a post-fixed point $\mathbf{y}$ with $\mathbf{x} \prec \mathbf{y} \leq \overline{1}$. By Lemma A.2 and Lemma A.3 the sequence $\mathbf{y}, f(\mathbf{y}), f(f(\mathbf{y})), \ldots$ converges to μ_f. On the other hand, by repeatedly applying f to both sides of the inequality $\mathbf{x} \prec \mathbf{y}$ we obtain that $\mathbf{x} \succ f(\mathbf{x}) \leq f^i(\mathbf{x}) \leq f^i(\mathbf{y})$ holds for all $i \geq 0$. Since $(f^i(\mathbf{y}))_i$ converges to μ_f, we have $\mathbf{x} \prec \mu_f$. □

A.2 Proof of Theorem 4.3

In order to prove Theorem 4.3 we can reuse some results of the previous subsection, but we need some additional lemmas. The following lemma was essentially proved in [6, 4].

Lemma A.4. *Let f be a perfectly superlinear PSP and let $\mathbf{x} \prec f(\mathbf{x})$. Then $\mathcal{N}(\mathbf{x}) = \mathbf{x} + (f'(\mathbf{x}))^*(f(\mathbf{x}) - \mathbf{x})$, where for any square matrix A, we define the matrix star $A^* = \sum_{i=0}^{\infty} A^i = Id + \sum_{i=1}^{\infty} A^i$.*

Proof. By Proposition 4.4 we have $\mathbf{x} \prec \mu_f$. For such points $\mathbf{x}$ it was shown in [6, 4] that $\rho(f'(\mathbf{x})) < 1$. By standard matrix facts [3], the matrix star A^* exists if and only if $\rho(A) < 1$. Furthermore, if A^* exists, it is equal to $(\mathrm{Id} - A)^{-1}$. Hence, $(Id - f'(\mathbf{x}))^{-1} = f'(\mathbf{x})^*$, and the statement follows. □

So, Lemma A.4 allows to replace the matrix inverse $(Id - f'(\mathbf{x}))^{-1}$ with the matrix star $f'(\mathbf{x})^*$ as long as $\mathbf{x} \prec f(\mathbf{x})$ holds, which will be true in this paper whenever we compute $\mathcal{N}(\mathbf{x})$.

The following two lemmas are used to show the validity of the floating assignment in line 4 of Algorithm 2.

Lemma A.5. *Let f be perfectly superlinear. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$ and $\mathbf{y} = \mathcal{N}(\mathbf{x})$. Then $f(\mathbf{x}) \prec \mathbf{y} \prec f(\mathbf{y}) \prec \overline{1}$.*

Proof. By Lemma A.4 we have $\mathcal{N}(\mathbf{x}) = \mathbf{x} + f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$. Write $\mathbf{\Delta} = f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$, i.e., $\mathbf{y} = \mathbf{x} + \mathbf{\Delta}$. As every variable depends directly on itself, we have $f'(\mathbf{x})(f(\mathbf{x}) - \mathbf{x}) \succ \overline{0}$. Consequently,

$$\begin{aligned}
f(\mathbf{x}) &\prec \mathbf{x} + (f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})(f(\mathbf{x}) - \mathbf{x}) \\
&= \mathbf{x} + \sum_{i=0,1} f'(\mathbf{x})^i (f(\mathbf{x}) - \mathbf{x}) \\
&\leq \mathbf{x} + \sum_{i=0}^{\infty} f'(\mathbf{x})^i (f(\mathbf{x}) - \mathbf{x}) \\
&= \mathbf{x} + \mathbf{\Delta} \\
&= \mathbf{y} \, .
\end{aligned}$$

Letting $\mathbf{u}, \mathbf{v}$ be any vectors, we write $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. Notice that $R(\mathbf{x}, \mathbf{\Delta}) \succ \overline{0}$, because $\mathbf{x} \succ \overline{0}$ and f is purely superlinear. Hence we have

$$\begin{aligned}
\mathbf{y} &= \mathbf{x} + \mathbf{\Delta} \\
&\prec \mathbf{x} + \mathbf{\Delta} + R(\mathbf{x}, \mathbf{\Delta}) \\
&= \mathbf{x} + f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) + R(\mathbf{x}, \mathbf{\Delta}) \\
&= \mathbf{x} + (f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) + R(\mathbf{x}, \mathbf{\Delta}) \\
&= f(\mathbf{x}) + f'(\mathbf{x})\mathbf{\Delta} + R(\mathbf{x}, \mathbf{\Delta}) \\
&= f(\mathbf{x} + \mathbf{\Delta}) \\
&= f(\mathbf{y}) \; .
\end{aligned}$$

By [6, 9] we have $\mathbf{y} \leq \mu_f$. By the monotonicity of f it follows that $f(\mathbf{y}) \leq f(\mu_f) = \mu_f$. Using the monotonicity of f once more and the fact that every variable depends directly on itself, we obtain $\mathbf{y} \prec f(\mathbf{y}) \prec f(f(\mathbf{y})) \leq f(\mu_f) = \mu_f$. As $\mu_f \leq \overline{1}$, it follows $f(\mathbf{y}) \prec \overline{1}$. □

The following lemma will also be used to show the validity of the floating assignment in line 4 of Algorithm 2. It says that $\mathcal{N}(\mathcal{N}(\mathbf{x}))$ is a post-fixed point of the linearization of f at $\mathbf{x}$.

Lemma A.6. *Let f be perfectly superlinear. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$. Let $\mathbf{z} = \mathcal{N}(\mathcal{N}(\mathbf{x}))$. Then $f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{z} - \mathbf{x}) \prec \mathbf{z}$.*

Proof. Letting $\mathbf{u}, \mathbf{v}$ be any vectors, we write $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. We write $\mathbf{y} = \mathcal{N}(\mathbf{x})$ and $\mathbf{\Delta} = f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$. Notice that $\mathbf{y} = \mathbf{x} + \mathbf{\Delta}$. We have

$$\begin{aligned}
\mathbf{z} &= \mathbf{y} + f'(\mathbf{y})^*(f(\mathbf{y}) - \mathbf{y}) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})^*(f(\mathbf{x} + \mathbf{\Delta}) - \mathbf{x} - \mathbf{\Delta}) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})^*(f(\mathbf{x}) + f'(\mathbf{x})\mathbf{\Delta} + R(\mathbf{x}, \mathbf{\Delta}) - \mathbf{x} - \mathbf{\Delta}) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})^*((f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) - \mathbf{\Delta} + R(\mathbf{x}, \mathbf{\Delta})) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})^*(\mathbf{\Delta} - \mathbf{\Delta} + R(\mathbf{x}, \mathbf{\Delta})) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})^* R(\mathbf{x}, \mathbf{\Delta}) \; .
\end{aligned}$$

It follows

$$\begin{aligned}
f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{z} - \mathbf{x}) &= f(\mathbf{x}) + f'(\mathbf{x})\left(\mathbf{\Delta} + f'(\mathbf{y})^* R(\mathbf{x}, \mathbf{\Delta})\right) \\
&= \mathbf{x} + (f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})f'(\mathbf{y})^* R(\mathbf{x}, \mathbf{\Delta}) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{x})f'(\mathbf{y})^* R(\mathbf{x}, \mathbf{\Delta}) \\
&\leq \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})f'(\mathbf{y})^* R(\mathbf{x}, \mathbf{\Delta}) \\
&\prec \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})f'(\mathbf{y})^* R(\mathbf{x}, \mathbf{\Delta}) + R(\mathbf{x}, \mathbf{\Delta}) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})^* R(\mathbf{x}, \mathbf{\Delta}) \\
&= \mathbf{z} \; .
\end{aligned}$$

For the $\prec$-inequality in this inequality chain, notice that, since $\mathbf{x} \prec f(\mathbf{x})$, we have $\mathbf{\Delta} \succ \overline{0}$, and since f is purely superlinear, we have $R(\mathbf{x}, \mathbf{\Delta}) \succ \overline{0}$. □

The following lemma states that $\mathcal{N}(\mathbf{x})$ is the least post-fixed point of the linearization of f at $\mathbf{x}$.

Lemma A.7. *Let f be a PSP. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$. Let $f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{z} - \mathbf{x}) \le \mathbf{z}$. Then $\mathcal{N}(\mathbf{x}) \le \mathbf{z}$.*

Proof. We write $\mathbf{y} = \mathcal{N}(\mathbf{x})$ and $\mathbf{\Delta} = f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$. Notice that $\mathbf{y} = \mathbf{x} + \mathbf{\Delta}$. We have

$$\begin{aligned}
(Id - f'(\mathbf{x}))\,(\mathbf{z} - \mathbf{y}) &= \mathbf{z} - \mathbf{x} - \mathbf{\Delta} + f'(\mathbf{x})(\mathbf{x} + \mathbf{\Delta} - \mathbf{z}) \\
&= \mathbf{z} - f(\mathbf{x}) + (f(\mathbf{x}) - \mathbf{x}) - f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) \\
&\quad\; + f'(\mathbf{x})(\mathbf{x} - \mathbf{z}) \\
&= \mathbf{z} - f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{x} - \mathbf{z}) \\
&\ge \overline{0} \quad \text{(by assumption)}\ .
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathbf{z} - \mathbf{y} &= (Id - f'(\mathbf{x}))^{-1}\,(Id - f'(\mathbf{x}))\,(\mathbf{z} - \mathbf{y}) \\
&= f'(\mathbf{x})^*\,(Id - f'(\mathbf{x}))\,(\mathbf{z} - \mathbf{y}) \ge f'(\mathbf{x})^*\overline{0} = \overline{0}\ ,
\end{aligned}$$

i.e., $\mathcal{N}(\mathbf{x}) = \mathbf{y} \le \mathbf{z}$. □

The following lemma shows the validity of the floating assignment in line 13 of Algorithm 2.

Lemma A.8. *Let f be a purely superlinear PSP. Let $\overline{0} \prec \mathbf{t} \prec \overline{1}$ such that $f'(1)\mathbf{t} \succ \mathbf{t}$. Let*

$$\mathbf{y} = \overline{1} - \min\left\{1, \frac{\min_{i\in\{1,\dots,n\}}(f'(\overline{1})\mathbf{t} - \mathbf{t})_i}{2\cdot \max_{i\in\{1,\dots,n\}}(f(\overline{2}))_i}\right\} \cdot \mathbf{t}\ .$$

Then $f(\mathbf{y}) \prec \mathbf{y} \prec \overline{1}$.

Proof. Letting $\mathbf{u}, \mathbf{v}$ be any vectors, we write $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. Let $r = \min\left\{1, \frac{\min_{i\in\{1,\dots,n\}}(f'(\overline{1})\mathbf{t} - \mathbf{t})_i}{2\cdot \max_{i\in\{1,\dots,n\}}(f(\overline{2}))_i}\right\}$. Then we have:

$$\begin{aligned}
R(\overline{1}, -r\mathbf{t}) &\le R(\overline{1}, r\mathbf{t}) \\
&\le r^2 R(\overline{1}, \mathbf{t}) && \text{(degree of } R(\overline{1}, \cdot) \text{ at least 2)} \\
&\le r^2 R(\overline{1}, \overline{1}) && (\mathbf{t} \le \overline{1}) \\
&\le r^2 f(\overline{2}) && (f(\overline{1} + \overline{1}) = f(\overline{1}) + f'(\overline{1})\overline{1} + R(\overline{1}, \overline{1})) \\
&\le r^2 \max_{i\in\{1,\dots,n\}}(f(\overline{2}))_i \cdot \overline{1} \\
&\le r \cdot \frac{\min_{i\in\{1,\dots,n\}}(f'(\overline{1})\mathbf{t} - \mathbf{t})_i}{2} \cdot \overline{1} && \text{(definition of } r) \\
&\le \frac{r}{2} \cdot (f'(\overline{1})\mathbf{t} - \mathbf{t}) \\
&\prec r \cdot (f'(\overline{1})\mathbf{t} - \mathbf{t}) && (f'(\overline{1})\mathbf{t} \succ \mathbf{t})
\end{aligned}$$

Using this inequality we obtain

$$\begin{aligned} f(\overline{1}-r\mathbf{t}) &= f(\overline{1}) + f'(\overline{1})\cdot(-r\mathbf{t}) + R(\overline{1},-r\mathbf{t}) \\ &\leq 1 - rf'(\overline{1})\mathbf{t} + R(\overline{1},-r\mathbf{t}) && (f(\overline{1}) \leq \overline{1}) \\ &\prec 1 - rf'(\overline{1})\mathbf{t} + r\cdot(f'(\overline{1})\mathbf{t} - \mathbf{t}) && \text{(see above)} \\ &= 1 - r\mathbf{t} \end{aligned}$$

□

The following lemma states a monotonicity property.

Lemma A.9. *Let f be perfectly superlinear. Let $\mathbf{y} = f(\mathbf{x}) \leq \mathbf{x}$ and $f_i(\mathbf{x}) < \mathbf{x}_i$ for some $i \in \{1,\ldots,n\}$. Then $f(\mathbf{y}) \leq \mathbf{y}$ and $f_i(\mathbf{y}) < \mathbf{y}_i$.*

Proof. We have $f(\mathbf{y}) = f(f(\mathbf{x})) \leq f(\mathbf{x}) = f(\mathbf{y})$ by the monotonicity of f. Moreover, since each component depends on itself, the strict inequality $f_i(\mathbf{x}) < \mathbf{x}_i$ implies the strict inequality $f_i(f(\mathbf{x})) < f_i(f(\mathbf{x}))$. □

The following lemma will be used in the proof of Theorem 4.3 to show that $\mathbf{ub}_i < 1$ eventually holds in the components i with $(\mu_f)_i < 1$.

Lemma A.10. *Let f be a purely superlinear PSP and $\mathbf{x} \geq \overline{0}, \mathbf{u} \succ \overline{0}$. Then*

$$f'(\mathbf{x}+\mathbf{u})\mathbf{u} \succ f(\mathbf{x}+\mathbf{u}) - f(\mathbf{x}).$$

Proof. It suffices to show $f_i(\mathbf{x}) - f_i(\mathbf{x}+\mathbf{u}) + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} > 0$ for every component i of f. We can write $f_i(\mathbf{x}+\mathbf{u})$ as $f_i(\mathbf{x}) + \int_0^1 f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u}\,ds$. Hence

$$\begin{aligned} & f_i(\mathbf{x}) - f_i(\mathbf{x}+\mathbf{u}) + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} \\ &= f_i(\mathbf{x}) - f_i(\mathbf{x}) - \int_0^1 f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u}\,ds + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} \\ &= -\int_0^1 f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u}\,ds + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} \\ &= -\int_0^{1/2} f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u}\,ds - \int_{1/2}^1 f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u}\,ds + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} \end{aligned}$$

For $0 \leq s \leq 1$, we have $f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u} \leq f_i'(\mathbf{x}+\mathbf{u})\mathbf{u}$, and for $0 \leq s \leq 1/2$, the inequality is strict, because $\mathbf{u} \succ \overline{0}$ and f is purely superlinear. Hence

$$\begin{aligned} & -\int_0^{1/2} f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u}\,ds - \int_{1/2}^1 f_i'(\mathbf{x}+s\mathbf{u})\mathbf{u}\,ds + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} \\ & > -\int_0^{\frac{1}{2}} f_i'(\mathbf{x}+\mathbf{u})\mathbf{u}\,ds - \int_{\frac{1}{2}}^1 f_i'(\mathbf{x}+\mathbf{u})\mathbf{u}\,ds + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} \\ & = -f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} + f_i'(\mathbf{x}+\mathbf{u})\mathbf{u} = 0. \end{aligned}$$

□

Now we can prove Theorem 4.3 which is restated here:

Theorem 4.3. *Algorithm 2 terminates and computes vectors $\mathbf{lb}, \mathbf{ub}$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$. Moreover, the sequences of lower and upper bounds computed by the algorithm both converge linearly to μ_f.*

Proof. The validity of the floating assignment in line 4 follows from Lemma A.5 and Lemma A.6. Next we show the convergence of the lower bounds. Let $(\mathbf{lb}^{(k)})_k$ be the sequence of the lower bounds $\mathbf{lb}$ in the algorithm, where $\mathbf{lb}^{(0)}$ is the result of `computeStrictPrefix`(f). Moreover, define an "exact" Newton sequence $\boldsymbol{\nu}^{(0)} = \mathbf{lb}^{(0)}$ and $\boldsymbol{\nu}^{(k+1)} = \mathcal{N}(\boldsymbol{\nu}^{(k)})$. We prove by induction that $\boldsymbol{\nu}^{(k)} \le \mathbf{lb}^{(k)}$. The induction base ($k = 0$) is trivial. Let $k \ge 0$. Notice that the floating assignment in line 4 guarantees $f(\mathbf{lb}^{(k)}) + f'(\mathbf{lb}^{(k)})\left(\mathbf{lb}^{(k+1)} - \mathbf{lb}^{(k)}\right) \le \mathbf{lb}^{(k+1)}$. Therefore, Lemma A.7 assures $\mathcal{N}(\mathbf{lb}^{(k)}) \le \mathbf{lb}^{(k+1)}$. Hence we have

$$\begin{aligned}
\boldsymbol{\nu}^{(k+1)} &= \mathcal{N}(\boldsymbol{\nu}^{(k)}) \\
&\le \mathcal{N}(\mathbf{lb}^{(k)}) \quad \text{(induction hypothesis, monotonicity of } \mathcal{N} \text{ as shown in [4])} \\
&\le \mathbf{lb}^{(k+1)} \quad \text{(as argued above)} .
\end{aligned}$$

So we have $\boldsymbol{\nu}^{(k)} \le \mathbf{lb}^{(k)}$ for all k. By the floating assignment in line 4, we have $\mathbf{lb}^{(k)} \prec f(\mathbf{lb}^{(k)})$, so $\mathbf{lb}^{(k)} \prec \mu_f$ by Proposition 4.4. As $(\boldsymbol{\nu}^{(k)})_k$ converges to μ_f, the sequence $(\mathbf{lb}^{(k)})_k$ converges to μ_f as well. In addition, it was shown in [9, 4] that $(\boldsymbol{\nu}^{(k)})_k$ converges linearly to μ_f. As $\boldsymbol{\nu}^{(k)} \le \mathbf{lb}^{(k)}$, the same holds for $(\mathbf{lb}^{(k)})_k$.

Now we turn the upper bounds $\mathbf{ub}$. We prove the following invariants of the algorithm:

(a) $f(\mathbf{ub}) \le \mathbf{ub} \le \overline{1}$;

(b) for all components j with $\mathbf{ub}_i < 1$, we have $f_i(\mathbf{ub}) < \mathbf{ub}_i$.

Clearly, this holds at the beginning (when $\mathbf{ub} = \overline{1}$). The invariants are clearly preserved by the assignment in line 8. Repeated application of Lemma A.9 shows that the floating assignment in line 9 is valid and that the invariants are preserved. Lemma A.8 implies that the floating assignment in line 13 are valid and preserve the invariants. Hence, the invariants hold.

Next we prove that for any component i with $(\mu_f)_i < 1$, we eventually have $\mathbf{ub}_i < 1$. Let us assume for the sake of a contradiction that there exists a component i with the property $P(i)$, where $P(i)$ means that $(\mu_f)_i < 1$ and $\mathbf{ub}_i = 1$ holds during the entire execution of the algorithm. Choose i "minimal" in the sense that for all variables X_j on which X_i depends we have that either X_i and X_j are in the same SCC, or $P(j)$ does not hold. Let S be the SCC of X_i and let X_j be any variable from $\mathrm{Var}\setminus S$ on which X_i depends. Since $P(i)$ holds, we must have $\mathbf{ub}_j = 1$ during the entire execution of the algorithm, because if $\mathbf{ub}_j < 1$ were true at some point, it would take at most n iterations before $\mathbf{ub}_i < 1$. As $P(j)$ cannot hold by the minimality of i, we have $(\mu_f)_j = 1$. Therefore, letting g denote the PSP obtained by restricting f to the S-components and replacing all variables from other SCCs by the constant 1, we have $\mu_g = (\mu_f)_S \prec \overline{1}$. Since $\mathbf{ub}_S = \overline{1}$ holds during the execution of the algorithm, we have $g(\overline{1}) = \overline{1}$, i.e., $\overline{1}$ is a fixed point of g. Therefore, by Lemma A.1, g cannot be linear, as $\mu_g \prec \overline{1}$. Since f is perfectly superlinear, g must then be purely superlinear. Application of Lemma A.10 (with $\mathbf{x} := \mu_g$ and $\mathbf{u} := \overline{1} - \mu_g$) yields

$$g'(\overline{1})(\overline{1} - \mu_g) \succ g(\overline{1}) - g(\mu_g) = \overline{1} - \mu_g \; .$$

Since the sequence of $\mathbf{lb}_S$ computed during the execution of the algorithm converges to μ_g, the continuity of $g'(\overline{1})$ implies that eventually $g'(\overline{1})(\overline{1} - \mathbf{lb}_S) \succ \overline{1} - \mathbf{lb}_S$ holds. But this means that the condition of line 12 is satisfied and, thus, the following assignment causes $\mathbf{ub}_S \prec \overline{1}$, contradicting our assumption that $P(i)$ holds. So we have shown that for any component i with $(\mu_f)_i < 1$, we eventually have $\mathbf{ub}_i < 1$.

Denote by $(\mathbf{ub}^{(k)})_k$ the sequence of upper bounds $\mathbf{ub}$ computed by the algorithm. It remains to show that this sequences converges linearly to μ_f. We have shown above that there exists k_0 such that for all $k \geq k_0$ we have that $\mathbf{ub}_i^{(k+1)} \leq \mathbf{ub}_i^{(k)} < 1$ holds for all components i with $(\mu_f)_i < 1$. Choose a real number r with $0 < r < 1$ such that for the point $\mathbf{p} := \mu_f + r(\overline{1} - \mu_f)$ we have $\mathbf{ub}^{(k_0)} \leq \mathbf{p} \leq \overline{1}$ and the following is true for all components i: either $(\mathbf{ub}^{(k_0)})_i = 1$ or $\mathbf{ub}_i^{(k_0)} < \mathbf{p}_i < 1$. Define the sequence $(\mathbf{p}^{(k)})_{k \geq k_0}$ by setting $\mathbf{p}^{(k_0)} := \mathbf{p}$ and $\mathbf{p}^{(k+1)} := f(\mathbf{p}^{(k)})$ for all $k \geq k_0$. By Lemma A.2 and Lemma A.3, this sequence converges linearly to μ_f. To prove that the same holds for $(\mathbf{ub}^{(k)})$, it suffices to show that $\mathbf{ub}^{(k)} \leq \mathbf{p}^{(k)}$ holds for all $k \geq k_0$. We proceed by induction on k. The induction base ($k = k_0$) holds by definition of $\mathbf{p}^{(k_0)}$. Let $k \geq k_0$. Then we have:

$$\begin{aligned} \mathbf{ub}^{(k+1)} &\leq f(\mathbf{ub}^{(k)}) && (\textbf{such that} \text{ clause of line 9}) \\ &\leq f(\mathbf{p}^{(k)}) && (\text{induction hypothesis}) \\ &= \mathbf{p}^{(k+1)} && (\text{definition of } \mathbf{p}^{(k+1)}) \end{aligned}$$

This completes the proof. □

A.3 Proof of Proposition 4.5

Here is a restatement of Proposition 4.5.

Proposition 4.5. *Let f be an scPSP. Let $\mathbf{t} \succ \overline{0}$ be a vector with $f'(\overline{1})\mathbf{t} \leq \mathbf{t}$. Then f is consistent.*

Proof. Lemma 3.2 (4) implies $\rho(f'(\overline{1})) \leq 1$. Hence, f is consistent by Proposition 3.1. □