Derivation Tree Analysis for Accelerated Fixed-Point Computation

Javier Esparza, Stefan Kiefer, Michael Luttenberger

Institut für Informatik, Technische Universität München, 85748 Garching, Germany
{esparza,kiefer,luttenbe}@model.in.tum.de

Abstract. We show that for several classes of idempotent semirings the least fixed-point of a polynomial system of equations $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ is equal to the least fixed-point of a *linear* system obtained by "linearizing" the polynomials of $\boldsymbol{f}$ in a certain way. Our proofs rely on derivation tree analysis, a proof principle that combines methods from algebra, calculus, and formal language theory, and was first used in [5] to show that Newton's method over commutative and idempotent semirings converges in a linear number of steps. Our results lead to efficient generic algorithms for computing the least fixed-point. We use these algorithms to derive several consequences, including an $O(N^3)$ algorithm for computing the throughput of a context-free grammar (obtained by speeding up the $O(N^4)$ algorithm of [2]), and a generalization of Courcelle's result stating that the downward-closed image of a context-free language is regular [3].

1 Introduction

Systems $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ of fixed-point equations, where $\boldsymbol{f}$ is a system of polynomials, appear naturally in semantics, interprocedural program analysis, language theory, and in the study of probabilistic systems (see e.g. [7, 8, 10, 13]). In all these applications the equations are interpreted over ω-continuous semirings, an algebraic structure that guarantees the existence of a least solution $\mu\boldsymbol{f}$. The key algorithmic problem is to compute or at least approximate $\mu\boldsymbol{f}$.

In [5, 4] we generalized Newton's method—the well-known method of numerical mathematics for approximating a zero of a differentiable function—to arbitrary ω-continuous semirings. Given a polynomial system $\boldsymbol{f}$, our generalized method computes a sequence of increasingly accurate approximations to $\mu\boldsymbol{f}$, called Newton approximants. We showed in [5] that the n-th Newton approximant of a system of n equations over an idempotent (w.r.t. addition) and commutative (w.r.t. multiplication) semiring is already equal to $\mu\boldsymbol{f}$. This theorem leads to a generic computing procedure.

Our proof of this result uses a (to the best of our knowledge) novel technique, which we call *derivation tree analysis*. The system $\boldsymbol{f}$ induces a set $\mathcal{T}$ of *derivation trees*, a generalization of the well-known derivation trees of context-free grammars. Each tree can be naturally assigned a semiring element, called the *yield* of the tree. It is easy to show that $\mu\boldsymbol{f}$ is equal to the sum of the yields of all derivation trees. Derivation tree analysis first identifies a subset T' of derivation trees whose total yield $\mathsf{Y}(T')$ is easy to compute in some sense, and then proves that T' satisfies the *embedding property*: $\mathsf{Y}(t) \sqsubseteq \mathsf{Y}(T')$ for every derivation tree t. If the semiring is idempotent, the embedding property implies $\mathsf{Y}(\mathcal{T}) = \mathsf{Y}(T')$, and so $\mu \boldsymbol{f} = \mathsf{Y}(T')$. In [5], the set T' was chosen so that $\mathsf{Y}(T')$ is equal to the n-th Newton approximant, and the embedding property was proved using some tree surgery and exploiting the commutativity of the semiring.

The computation of the n-th Newton approximant can still require considerable resources. In this paper we present a further application of derivation tree analysis to idempotent semirings, leading to more efficient computation algorithms. For this, we define the set $\mathcal{B}$ of *bamboos* of a system $\boldsymbol{f}$. Loosely speaking, bamboos are derivation trees with an arbitrarily long stem but only short branches. We first show that $\mathsf{Y}(\mathcal{B})$ is the solution of a linear system of equations whose functions are similar (but not identical) to the straightforward linearisation of $\boldsymbol{f}$. Then, we prove that the following three classes of semirings satisfy the embedding property:
• *Star-distributive semirings* are idempotent and commutative semirings satisfying the additional axiom $(a+b)^* = a^* + b^*$ (where * is the well-known Kleene iteration operator). The so-called "tropical" $(\min, +)$-semiring over the reals (extended with $+\infty$ and $-\infty$) is star-distributive. Our tree analysis leads to an algorithm for computing $\mu \boldsymbol{f}$ very similar to the generalized Bellman-Ford algorithm of Gawlitza and Seidl [9]. We use it to derive a new algorithm for computing the throughput of a context-free grammar, a problem introduced and analyzed by Caucal et al. in [2]. Our algorithm runs in $O(N^3)$, a factor N faster than the algorithm presented in [2].
• *Lossy semirings* are idempotent semirings satisfying the additional axiom $a + 1 = a$ where 1 is the neutral element of multiplication. A natural model are downward-closed languages with union and concatenation as operations. Lossy semirings find application in the verification of lossy channel systems, a model of computation thoroughly investigated by Abdulla et al. (see e.g. [1]). Our tree analysis leads to an algebraic proof of Courcelle's theorem stating that the downward closure of a context-free language is effectively regular [3].
• 1-*bounded semirings* are idempotent semirings where the equation $a+1 = 1$ holds. A natural example is the "maximum probability" semiring with the interval $[0, 1]$ as carrier, maximum as addition, and standard multiplication over the reals. Using derivation tree analysis it is very easy to show that the least fixed-point $\mu \boldsymbol{f}$ of a polynomial system $\boldsymbol{f}$ with n variables is given by $\boldsymbol{f}^n(\mathbf{0})$, the n-fold application of $\boldsymbol{f}$ to $\mathbf{0}$.

The rest of the paper is organized as follows. After the preliminaries in Section 2 we introduce derivation tree analysis in Section 3. Bamboos are defined in Section 4. In the Sections 5, 6 and 7 we apply derivation tree analysis to the semiring classes mentioned above. A technical report [6] includes the missing proofs.

2 Preliminaries

As usual, $\mathbb{N}$ denotes the set of natural numbers including 0.

An *idempotent semiring* $\mathcal{S} = \langle S, +, \cdot, 0, 1\rangle$ consists of a commutative, idempotent additive monoid $\langle S, +, 0\rangle$, and a multiplicative monoid $\langle S, \cdot, 1\rangle$. In the following we often omit the dot $\cdot$ in products. Both algebraic structures are connected by left- and right-distributivity, e.g. $a(b + c) = ab + ac$, and by the requirement that $0 \cdot a = 0$ for all $a \in S$. The *natural-order relation* $\sqsubseteq\, S \times S$ is defined by $a \sqsubseteq b \Leftrightarrow a + b = b$. The semiring $\mathcal{S}$ is *naturally ordered* if $\sqsubseteq$ is a partial order.

An idempotent, naturally ordered semiring $\mathcal{S}$ is *ω-continuous*, if countable summation $\sum_{i\in\mathbb{N}} a_i \in S$ is defined (with $a_i \in S$), and satisfies the following requirements: (i) summation is continuous, i.e., $\sup^{\sqsubseteq}\{a_0 + a_1 + \ldots + a_k \mid k \in \mathbb{N}\} = \sum_{i\in\mathbb{N}} a_i$ for all sequences $a : N \to S$; (ii) distributivity extends in the natural way to countable summation; and (iii) $\sum_{j\in J}\sum_{i\in I_j} a_i = \sum_{i\in\mathbb{N}} a_i$ holds for all partitions $(I_j)_{j\in J}$ of $\mathbb{N}$. In every such ω-continuous semiring the Kleene-star operator $^* : S \to S$ is well-defined by $a^* := \sum_{k\in\mathbb{N}} a^k$ for all $a \in S$. In the following we consider only idempotent ω-continuous semirings $\mathcal{S}$. We refer to them as *io-semirings*.

We fix a finite, non-empty set $\mathcal{X}$ of *variables* for the rest of the section, and use n to denote $|\mathcal{X}|$ in the following. A map from $\mathcal{X}$ to S is called a *vector*. The set of all vectors is denoted by V. We write both $\boldsymbol{v}(X)$ and $\boldsymbol{v}_X$ for the value of a vector $\boldsymbol{v}$ at $X \in \mathcal{X}$, also called the X-component of $\boldsymbol{v}$. Sum of vectors is defined componentwise: given a countable set I and a vector $\boldsymbol{v}_i$ for every $i \in I$, we denote by $\sum_{i\in I} \boldsymbol{v}_i$ the vector given by $\left(\sum_{i\in I} \boldsymbol{v}_i\right)(X) = \sum_{i\in I} \boldsymbol{v}_i(X)$ for every $X \in \mathcal{X}$.

A *monomial of degree* k is a finite expression $a_1X_1a_2\cdots a_kX_ka_{k+1}$ where $k \geq 0$, $a_1,\ldots,a_{k+1} \in S\setminus\{0\}$ and $X_1,\ldots,X_k \in \mathcal{X}$. A *polynomial* is an expression of the form $m_1 + \cdots + m_k$ where $k \geq 0$ and $m_1,\ldots,m_k$ are monomials. Since $\mathcal{S}$ is idempotent, we assume w.l.o.g. that all monomials of a polynomial are distinct. The degree of a polynomial is the largest degree of its monomials. We let $\mathcal{S}[\mathcal{X}]$ denote the set of all polynomials.

Let $f = \alpha_1X_1\alpha_2\ldots X_k\alpha_{k+1}$ be a monomial and let $\boldsymbol{v}$ be a vector. The *evaluation of f at $\boldsymbol{v}$*, denoted by $f(\boldsymbol{v})$, is the product $\alpha_1\boldsymbol{v}_{X_1}\alpha_2\cdots\alpha_k\boldsymbol{v}_{X_k}\alpha_{k+1}$. We extend this to any polynomial: if $f = \sum_{i=1}^k m_i$, then $f(\boldsymbol{v}) = \sum_{i=1}^k m_i(\boldsymbol{v})$.

A *system of polynomials* or polynomial system is a map $\boldsymbol{f} : \mathcal{X} \to \mathcal{S}[\mathcal{X}]$. We write $\boldsymbol{f}_X$ for $\boldsymbol{f}(X)$. Every polynomial system induces a map from V to V by componentwise evaluation of the polynomials: $\boldsymbol{f}(\boldsymbol{v})_X := \boldsymbol{f}_X(\boldsymbol{v})$ for all $\boldsymbol{v} \in V$, and $X \in \mathcal{X}$. The following proposition, which follows easily from Kleene's theorem and the fact that $\boldsymbol{f}$ is a monotone and continuous mapping, shows that any polynomial system $\boldsymbol{f}$ has a least fixed-point $\mu\boldsymbol{f}$, which is by definition the least solution of $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$.

Proposition 1. *A polynomial system $\boldsymbol{f}$ has a unique least fixed-point $\mu\boldsymbol{f}$, i.e., $\mu\boldsymbol{f} = \boldsymbol{f}(\mu\boldsymbol{f})$, and $\mu\boldsymbol{f} \sqsubseteq \boldsymbol{v}$ holds for all $\boldsymbol{v}$ with $\boldsymbol{v} = \boldsymbol{f}(\boldsymbol{v})$. Further, $\mu\boldsymbol{f}$ is the supremum (w.r.t. $\sqsubseteq$) of the* Kleene sequence $(\boldsymbol{f}^i(\boldsymbol{0}))_{i\in\mathbb{N}}$, *where $\boldsymbol{f}^i$ denotes the i-fold application of $\boldsymbol{f}$.*

3 Derivation Trees

We generalize the notion of derivation tree, as known from formal languages and grammars. We identify a node u of a (ordered) tree t with the subtree of t rooted at u. In particular, we identify a tree with its root.

Let $\boldsymbol{f}$ be a polynomial system over a set $\mathcal{X}$ of variables. A *derivation tree* t of $\boldsymbol{f}$ is an ordered (finite) tree whose nodes are labelled with both a variable X and a monomial m of $\boldsymbol{f}_X$. We write λ_v, resp. λ_m for the corresponding labelling-functions. Moreover, if the monomial labelling of a node u is $\lambda_m(u) = a_1X_1a_2\ldots X_sa_{s+1}$ for some $s \geq 0$, then u has exactly s children $u_1,\ldots,u_s$, ordered from left to right, with $\lambda_v(u_i) = X_i$ for all $i = 1,\ldots,s$. A derivation tree t is an X-*tree* if $\lambda_v(t) = X$. The set of all X-trees of $\boldsymbol{f}$ is denoted by $\mathcal{T}_{\boldsymbol{f},X}$, or just by $\mathcal{T}_X$ if $\boldsymbol{f}$ is clear from the context.

The left part of Figure 1 shows a derivation tree of the system $\boldsymbol{f}$ over the variables X and Y given by $\boldsymbol{f}_X = aXYb + c$ and $\boldsymbol{f}_Y = dX + Ye$. The derivation trees of $\boldsymbol{f}$ are very similar to the derivation trees of the context-free grammar with productions $X \to aXYb|c$ and $Y \to dX|Ye$. For technical reasons, the nodes of "our" trees are labeled by "productions" (for instance, the label $(X, aXYb)$ corresponds to the production $X \to aXYb$). On the right of Figure 1 we show how the tree would look like according to the standard definition. The height $h(t)$ of a derivation tree t is the length of a longest

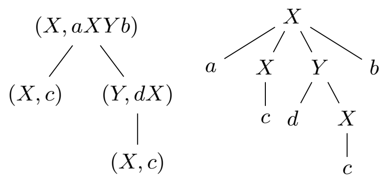


Fig. 1. A derivation tree on the left, and its standard representation on the right

path from the root to a leaf. The set of X-trees (of $\boldsymbol{f}$) of height *at most* h is denoted by $\mathcal{T}_X^{(h)}$. The yield $\mathsf{Y}(t)$ of a derivation tree t with $\lambda_m(t) = a_1 X_1 a_2 \cdots X_s a_{s+1}$ is inductively defined to be $\mathsf{Y}(t) = a_1 \mathsf{Y}(t_1) a_2 \cdots \mathsf{Y}(t_s) a_{s+1}$. We extend the definition of Y to sets $T \subseteq \mathcal{T}_X$ by setting $\mathsf{Y}(T) := \sum_{t \in T} \mathsf{Y}(t)$. E.g., the system $\boldsymbol{f}$ defined above has exactly two X-trees of height at most 2: the tree consisting of a single node labeled by (X, c), and the left tree of Figure 1. Their yields are c and $acdcb$, respectively, and so $\mathsf{Y}(\mathcal{T}_X^{(2)}) = c + acdcb$. It follows $\mathsf{Y}(\mathcal{T}_X^{(2)}) = \boldsymbol{f}^3(\mathbf{0})_X$, i.e., the yield of the X-trees of height at most 2 is equal to the "Kleene approximant" $\boldsymbol{f}^3(\mathbf{0})_X$ from Proposition 1. The following proposition, easy to prove [4], shows that this is not a coincidence.

Proposition 2. *For all $h \in \mathbb{N}$ and $X \in \mathcal{X}$, we have* $\mathsf{Y}(\mathcal{T}_X^{(h)}) = \left(\boldsymbol{f}^{h+1}(\mathbf{0})\right)_X$.

Together with Proposition 1 we get:

Corollary 1. $\mu \boldsymbol{f}_X = \mathsf{Y}(\mathcal{T}_X)$.

3.1 Derivation Tree Analysis

We say that a set T_X of X-trees satisfies the *embedding property* if $\mathsf{Y}(t) \sqsubseteq \mathsf{Y}(T_X)$ holds for every X-tree t. Loosely speaking, the yield of every X-tree can be "embedded" in the yield of T_X. As addition is idempotent, the embedding property immediately implies that $\mathsf{Y}(\mathcal{T}_X) \sqsubseteq \mathsf{Y}(T_X)$. Of course, as $T_X \subseteq \mathcal{T}_X$, we also have the other direction, which leads to the following result.

Proposition 3. *Let $\boldsymbol{f}$ be a system of polynomials over an io-semiring, and let X be a variable of $\boldsymbol{f}$. If a set T_X of X-trees of $\boldsymbol{f}$ satisfies the embedding property, then* $\mu \boldsymbol{f} = \mathsf{Y}(T_X)$.

This proposition suggests a technique for the design of efficient algorithms computing $\mu \boldsymbol{f}$: (1) define a set T_X of derivation trees whose yield is "easy to compute" in some io-semiring, and (2) identify "relevant" classes of io-semirings for which T_X satisfies the embedding property. By Proposition 3, $\mu \boldsymbol{f}$ is "easy to compute" for these classes. We call this technique *derivation tree analysis*.

4 Bamboos and their Yield

The difficulty of derivation tree analysis lies in finding a set T_X exhibiting a good balance between the contradictory requirements "easy to compute" and "relevant": if $T_X = \emptyset$ then the yield is trivial to compute, but T_X does not satisfy the embedding property in any interesting case. Conversely, $T_X = \mathcal{T}_X$ trivially satisfies the embedding property for every io-semiring, but is not easy to compute. The main contribution of this paper is the identification of a class of derivation trees, *bamboos*, exhibiting this balance. In this section we define bamboos and show that their yield is the least solution of a system of *linear* equations easily derivable from $\boldsymbol{f}$. The "easy to compute" part is justified by the fact that in most semirings used in practice linear equations are far easier to solve than polynomial equations (e.g. in the real semiring or the language semiring with union and concatenation as operations). The "relevance" of bamboos is justified in the next three sections.

Definition 1. *Let $\boldsymbol{f}$ be a system of polynomials. A tree $t \in \mathcal{T}_{\boldsymbol{f},X}$ is an X-*bamboo *if there is a path leading from the root of t to some leaf of t, the* stem, *such that the height of every subtree of t not containing a node of the stem is at most $n-1$. The set of all X-bamboos of $\boldsymbol{f}$ is denoted by $\mathcal{B}_{f,X}$, or just by $\mathcal{B}_X$ if $\boldsymbol{f}$ is clear from the context.*

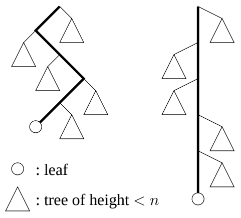


Fig. 2. A bamboo with its stem printed bold; on the right it is shown with its stem straightened.

In order to define the system of linear equations mentioned above we need the notion of differential of a system of polynomials.

Definition 2. *Let $f \in \mathcal{S}[\mathcal{X}]$ be a polynomial and let $\boldsymbol{v} \in V$ be a vector. The* differential *of f* at *$\boldsymbol{v}$* w.r.t. a variable *X is the map $D_X f|_{\boldsymbol{v}} \colon V \to S$ inductively defined as follows:*

$$D_X f|_{\boldsymbol{v}}(\boldsymbol{a}) = \begin{cases} 0 & \text{if } f \in S \text{ or } f \in \mathcal{X} \setminus \{X\} \\ \boldsymbol{a}_X & \text{if } f = X \\ D_X g|_{\boldsymbol{v}}(\boldsymbol{a}) \cdot h(\boldsymbol{v}) + g(\boldsymbol{v}) \cdot D_X h|_{\boldsymbol{v}}(\boldsymbol{a}) & \text{if } f = g \cdot h \\ \sum_{i=1}^{k} D_X m_i|_{\boldsymbol{v}}(\boldsymbol{a}) & \text{if } f = \sum_{i=1}^{k} m_i . \end{cases}$$

Further, we define the differential of f *at* $\boldsymbol{v}$ *by* $Df|_{\boldsymbol{v}}(\boldsymbol{a}) := \sum_{X \in \mathcal{X}} D_X f|_{\boldsymbol{v}}(\boldsymbol{a})$. *The differential of a system of polynomials* $\boldsymbol{f}$ *at* $\boldsymbol{v}$ *is defined componentwise by* $(D\boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{a}))_X := D(\boldsymbol{f}_X)|_{\boldsymbol{v}}(\boldsymbol{a})$ *for all* $X \in \mathcal{X}$.

Example 1. For $f(X,Y) = a \cdot X \cdot X \cdot Y \cdot b$, $\boldsymbol{v} = (v_X, v_Y)$, $\boldsymbol{c} = (c_X, c_Y)$ we have:

$$D_X f|_{\boldsymbol{v}}(\boldsymbol{c}) = a \cdot c_X \cdot v_X \cdot v_Y \cdot b + a \cdot v_X \cdot c_X \cdot v_Y \cdot b$$
$$D_Y f|_{\boldsymbol{v}}(\boldsymbol{c}) = a \cdot v_X \cdot v_X \cdot c_Y \cdot b$$

Using differentials we define a particular linearization of a polynomial system.

Definition 3. *Let* $\boldsymbol{f}$ *be a system of* n *polynomials. The* bamboo system $\boldsymbol{f}_{\mathcal{B}}$ *associated to* $\boldsymbol{f}$ *is the linear system* $\boldsymbol{f}_{\mathcal{B}}(\boldsymbol{X}) = D\boldsymbol{f}|_{\boldsymbol{f}^n(\boldsymbol{0})}(\boldsymbol{X}) + \boldsymbol{f}(\boldsymbol{0})$. *The least solution of the system of equations* $\boldsymbol{X} = \boldsymbol{f}_{\mathcal{B}}(\boldsymbol{X})$ *is denoted by* $\mu\boldsymbol{f}_{\mathcal{B}}$.

Now we can state the relation between bamboos and bamboo systems.

Theorem 1. *Let* $\boldsymbol{f}$ *be a system of polynomials over an io-semiring. For every variable* X *of* $\boldsymbol{f}$ *we have* $\mathsf{Y}(\mathcal{B}_X) = (\mu\boldsymbol{f}_{\mathcal{B}})_X$, *i.e., the yield of the* X*-bamboos is equal to the* X*-component of the least solution of the bamboo system.*

Together with Proposition 3 we get the following corollary.

Corollary 2 (derivation tree analysis for bamboos). *Let* $\boldsymbol{f}$ *be a system of polynomials over an io-semiring. If* $\mathcal{B}_X$ *satisfies the embedding property for all* X, *i.e., for all* X*-trees* t *it holds* $\mathsf{Y}(t) \sqsubseteq \mathsf{Y}(\mathcal{B}_X)$, *then* $\mu\boldsymbol{f} = \mu\boldsymbol{f}_{\mathcal{B}}$.

5 Star-Distributive Semirings

Definition 4. *A commutative (w.r.t. multiplication) io-semiring* $\mathcal{S}$ *is* star-distributive *if* $(a+b)^* = a^* + b^*$ *holds for all* $a, b \in S$.

A commutative io-semiring is star-distributive whenever the natural order $\sqsubseteq$ is total:

Proposition 4. *Any totally ordered commutative io-semiring is star-distributive.*

Proof. Let w.l.o.g. $a \sqsubseteq b$. Then $(a+b)^* = b^* \sqsubseteq a^* + b^* \sqsubseteq (a+b)^*$. □

In particular, the $(\min, +)$-semiring over the integers or reals is star-distributive.

We have already considered commutative idempotent semirings in [5] where we showed that $\mu\boldsymbol{f}$ can be computed by solving n *linear* equation systems by means of a Newton-like method, improving the $\mathcal{O}(3^n)$ bound of Hopkins and Kozen [12]. In this section we improve this result even further for star-distributive semirings: One single linear system, the bamboo system $\boldsymbol{f}_{\mathcal{B}}$, needs to be solved. This leads to an efficient algorithm for computing $\mu\boldsymbol{f}$ in arbitrary star-distributive semirings. In Section 5.1 we instantiate this algorithm for the $(\min, +)$-semiring; in Section 5.2 we use it to improve the algorithm of [2] for computing the throughput of a context-free grammar.
We start by stating two useful properties of star-distributive semirings.

Proposition 5. *In any star-distributive semiring the following equations hold:*
(1) $a^*b^* = a^* + b^*$, *and (2)* $(ab^*)^* = a^* + ab^*$.

We can now state and prove our result:

Theorem 2. *$\mu \boldsymbol{f} = \mu \boldsymbol{f}_{\mathcal{B}}$ holds for polynomial systems $\boldsymbol{f}$ over star-distributive semirings.*

Proof Sketch (see [6] for a complete proof). The proof is by derivation tree analysis. So it suffices to discharge the precondition of Corollary 2. More precisely we show for any X-tree t that $\mathsf{Y}(t) \sqsubseteq \mathsf{Y}(\mathcal{B}_X)$ holds. It suffices to consider the case where t is not an X-bamboo. Then the height of t is at least n, and so t is "pumpable", i.e., one can choose a path p in t from the root to a leaf such that two different nodes on the path share the same variable-label. So t can be decomposed into three (partial) trees with yields a, b, c, respectively, such that $\mathsf{Y}(t) = abc$, see the left side of Figure 3(a). Notice that, by commutativity of product, ab^*c is the yield of a set of trees obtained by

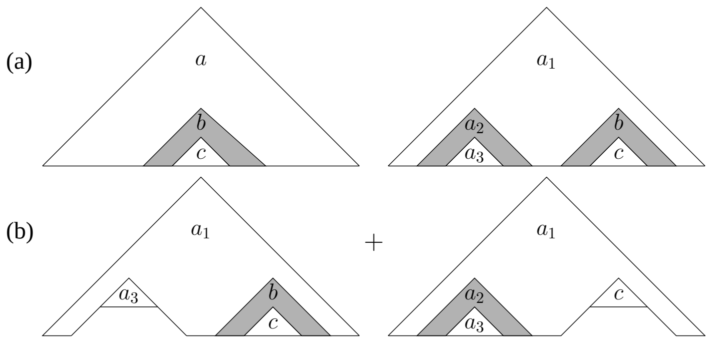


Fig. 3. "Unpumping" trees to make them bamboos

"pumping" t. We show $ab^*c \sqsubseteq \mathsf{Y}(\mathcal{B}_X)$ which implies $\mathsf{Y}(t) \sqsubseteq \mathsf{Y}(\mathcal{B}_X)$. As t is not an X-bamboo, t has a pumpable subtree disjoint from p. In this sketch we assume that it is a subtree of that part of t whose yield is a, see the right side of Figure 3(a). Now we have $a = a_1 a_2 a_3$, and so $ab^*c = a_1 a_2 a_3 b^* c \sqsubseteq a_1 a_2^* a_3 b^* c = a_1 a_3 b^* c + a_1 a_2^* a_3 c$, where we used commutativity and Proposition 5(1) in the last step. Both summands in above sum are yields of sets of trees obtained by pumping pumpable trees smaller than t, see Figure 3(b). By an inductive argument those yields are both included in $\mathsf{Y}(\mathcal{B}_X)$. $\square$

5.1 The $(\min, +)$-Semiring

Consider the "tropical" semiring $\mathcal{R} = (\mathbb{R} \cup \{-\infty, \infty\}, \wedge, +_{\mathbb{R}}, \infty, 0)$. By $\wedge$ resp. $+_{\mathbb{R}}$ we mean minimum resp. addition over the reals. Observe that the natural order $\sqsubseteq$ is the order $\geq$ on the reals.[1] As $\mathcal{R}$ is totally ordered, Proposition 4 implies that $\mathcal{R}$ is star-distributive. Assume for the rest of this section that $\boldsymbol{f}$ is a polynomial system over $\mathcal{R}$ of degree at most 2. We can apply Theorem 2, i.e., $\mu \boldsymbol{f} = \mu \boldsymbol{f}_{\mathcal{B}}$ holds. This immediately suggests a polynomial algorithm to compute the least fixed-point: Compute $\boldsymbol{f}^n(\infty)$ by

[1] By symmetry, we could equivalently consider maximum instead of minimum.

performing n Kleene iterations, and solve the linear system $\boldsymbol{X} = D\boldsymbol{f}|_{\boldsymbol{f}^n(\boldsymbol{\infty})}(\boldsymbol{X}) \wedge \boldsymbol{f}(\boldsymbol{\infty})$. The latter can be done by means of the Bellman-Ford algorithm.

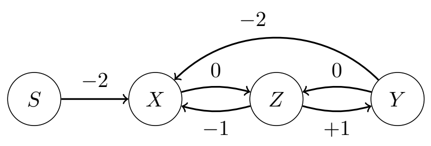
Example 2. Consider the following equation system.

$$\big(X, \quad Y, \quad Z\big) = \big(-2 \wedge (Y +_{\mathbb{R}} Z), \quad Z +_{\mathbb{R}} 1, \quad X \wedge Y\big) =: \boldsymbol{f}(\boldsymbol{X})$$

We have $\boldsymbol{f}(\boldsymbol{\infty}) = (-2, \infty, \infty), \boldsymbol{f}^2(\boldsymbol{\infty}) = (-2, \infty, -2), \boldsymbol{f}^3(\boldsymbol{\infty}) = (-2, -1, -2)$. The linear system $\boldsymbol{X} = D\boldsymbol{f}|_{\boldsymbol{f}^n(\boldsymbol{\infty})}(\boldsymbol{X}) \wedge \boldsymbol{f}(\boldsymbol{\infty}) = \boldsymbol{f}_{\mathcal{B}}(\boldsymbol{X})$ looks as follows:

$$\big(X, \quad Y, \quad Z\big) = \big(-2 \wedge (-1 +_{\mathbb{R}} Z) \wedge (Y +_{\mathbb{R}} -2), \quad Z +_{\mathbb{R}} 1, \quad X \wedge Y\big).$$

This equation system corresponds in a straightforward way to the following graph.



We claim that the V-component of $\mu\boldsymbol{f}_{\mathcal{B}}$ equals the least weight of any path from S to V where $V \in \{X, Y, Z\}$. To see this, notice that $(\boldsymbol{f}_{\mathcal{B}}^k(\boldsymbol{\infty}))_V$ corresponds to the least weight of any path from S to V of length at most k. The claim follows by Kleene's theorem. So we can compute $\mu\boldsymbol{f}_{\mathcal{B}}$ with the Bellman-Ford algorithm. In our example, X, Y, Z are all reachable from S via a negative cycle, so $\mu\boldsymbol{f}_{\mathcal{B}} = (-\infty, -\infty, -\infty)$. By Theorem 2, $\mu\boldsymbol{f} = \mu\boldsymbol{f}_{\mathcal{B}} = (-\infty, -\infty, -\infty)$. $\square$

The Bellman-Ford algorithm can be used here as it handles negative cycles correctly. The overall runtime of our algorithm to compute $\mu\boldsymbol{f}$ is dominated by the Bellman-Ford algorithm. Its runtime is in $\mathcal{O}(n \cdot m)$, where m is the number of monomials appearing in $\boldsymbol{f}$. We conclude that our algorithm has the same asymptotic complexity as the "generalized Bellman-Ford" algorithm of [9]. It is by a factor of n faster than the algorithm deducible from [5] because our new algorithm uses the Bellman-Ford algorithm only once instead of n times.

5.2 Throughput of Grammars

In [2], a polynomial algorithm for computing the *throughput* of a context-free grammar was given. Now we show that the algorithm can be both simplified and accelerated by computing least fixed-points according to Theorem 2.

Let us define the problem following [2]. Let Σ be a finite alphabet and $\rho : \Sigma \to \mathbb{N}$ a weight function. We extend ρ to words $a_1 \cdots a_k \in \Sigma^*$ by setting $\rho(a_1 \cdots a_k) := \rho(a_1) + \ldots + \rho(a_k)$.[2] The mean weight of a non-empty word w is defined as $\overline{\rho}(w) := \rho(w)/|w|$. The throughput of a non-empty language $L \subseteq \Sigma^+$ is defined as the infimum of the mean weights of the words in L: $tp(L) := \inf\{\overline{\rho}(w) \mid w \in L\}$. Let $G = (\Sigma, \mathcal{X}, P, S)$ be a context-free grammar and $L = L(G)$ its language. The problem is to

[2] We write $+$ for the addition of reals in this section.

compute $tp(L)$. As in [2] we assume that G has at most 2 symbols on the right hand side of every production and that L is non-empty and contains only non-empty words.

Note that we cannot simply construct a polynomial system having $tp(L)$ as its least fixed-point, as the throughput of two non-terminals is not additive. In [2] an ingenious algorithm is proposed to avoid this problem. Assume we already know a routine, the *comparing routine*, that decides for a given $t \in \mathbb{Q}$ whether $tp(L) \geq t$ holds. Assume further that this routine has $\mathcal{O}(N^k)$ time complexity for some k. Using the comparing routine we can approximate $tp(L)$ up to any given accuracy by means of binary search. Let $d = \max_{a \in \Sigma} \rho(a) - \min_{a \in \Sigma} \rho(a)$. A dichotomy result of [2] shows that $\mathcal{O}(N + \log d)$ iterations of binary search suffice to approximate $tp(L)$ up to an ε that allows to compute the exact value of $tp(L)$ in time $\mathcal{O}(N^3)$. This is proved by showing that, once a value t has been determined such that $t - \varepsilon < tp(L) \leq t$, one can:

– transform G in $\mathcal{O}(N^3)$ time into a grammar G' of size $\mathcal{O}(N^3)$ generating a finite language, and having the same throughput as G (this construction does not yet depend on $tp(L)$);
– compute the throughput of G' in linear time in the size of G', i.e., in $\mathcal{O}(N^3)$ time.

The full algorithm for the throughput runs then in $\mathcal{O}(N^k(N + \log d)) + \mathcal{O}(N^3)$ time.

The algorithm of [2] and our new algorithm differ in the comparing routine. In the routine of [2] the transformation of G into the grammar G' is done *before* $tp(L)$ has been determined. Then a linear time algorithm can be applied to G' to decide whether $tp(L) \geq t$ holds. (This algorithm does not work for arbitrary context-free grammars, and that is why one needs to transform G into G'.) Since G' has size $\mathcal{O}(N^3)$, the comparing routine has $k = 3$, and so the full algorithm runs in $\mathcal{O}(N^4 + N^3 \log d)$ time.

We give a more efficient comparing routine with $k = 2$. Given a $t \in \mathbb{Q}$, assign to each word $w \in \Sigma^+$ its *throughput balance* $\sigma_t(w) = \rho(w) - |w| \cdot t$. Notice that $\sigma_t(w) \geq 0$ if and only if $\overline{\rho}(w) \geq t$. Further, for two words w, u we now have $\sigma_t(wu) = \sigma_t(w) + \sigma_t(u)$. So we can set up a polynomial system $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ over the tropical semiring $\mathcal{R}$ where $\boldsymbol{f}$ is constructed such that each variable $X \in \mathcal{X}$ in the equation system corresponds to the minimum (infimum) throughput balance of the words derivable from X. More formally, define a map m by setting $m(a) = \rho(a) - t$ for $a \in \Sigma$ and $m(X) = X$ for $X \in \mathcal{X}$. Extend m to words in $(\Sigma \cup \mathcal{X})^*$ by setting $m(\alpha_1 \cdots \alpha_k) = m(\alpha_1) + \cdots + m(\alpha_k)$. Let P_X be the productions of G with X on the left hand side. Then set $\boldsymbol{f}_X(\boldsymbol{X}) := \bigwedge_{(X \to w) \in P_X} m(w)$. For instance, if P_X consists of the rules $X \to aXY$ and $X \to bZ$, we have $\boldsymbol{f}_X(\boldsymbol{X}) = \rho(a) - t + X + Y \wedge \rho(b) - t + Z$.

It is easy to see that the relevant solution of the system $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ is the least one w.r.t. $\sqsubseteq$, i.e., $(\mu \boldsymbol{f})_S \geq 0$ if and only if $tp(L) \geq t$. So we can use the algorithm from Section 5.1 as our comparing routine. This takes time $\mathcal{O}(N^2)$ where N is the size of the grammar. With that comparing routine we obtain an algorithm for computing the throughput with $\mathcal{O}(N^3 + N^2 \log d)$ runtime.

6 Lossy Semirings

Definition 5. *An io-semiring $\mathcal{S}$ is called* lossy *if $1 \sqsubseteq a$ holds for all $a \neq 0$.*

Note that by definition of natural order the requirement $1 \sqsubseteq a$ is equivalent to $a = a+1$. In the free semiring generated by a finite alphabet Σ, and augmented by the equation $a = a + 1$ ($a \in S \setminus \{0\}$), every language $L \subseteq \Sigma^*$ is "downward closed", i.e. for every word $w = a_1 a_2 \ldots a_l \in L$ all possible subwords $\{a'_1 a'_2 \ldots a'_l \mid a'_i \in \{\varepsilon, a_i\}\}$ are also included in L. By virtue of Higman's lemma [11] the downward-closure of a context-free language is regular. This has been used in [1] for an efficient analysis of systems with unbounded, lossy FIFO channels. Downward closure was used there to model the loss of messages due to transmission errors.

We say that a system $\boldsymbol{f}$ of polynomials is *clean* if $\mu \boldsymbol{f}_X \neq 0$ for all $X \in \mathcal{X}$. Every system can be *cleaned* in linear time by removing the equations of all variables X such that $\mu \boldsymbol{f}_X = 0$ and setting these variables to 0 in the other equations (the procedure is similar to the one that eliminates non-productive variables in context-free grammars). We consider only clean systems, and introduce a normal form for them.

Definition 6. *Let $\boldsymbol{f} \in \mathcal{S}[\mathcal{X}]^{\mathcal{X}}$ be a system of polynomials over a lossy semiring. $\boldsymbol{f}$ is in* quadratic normal form *if every polynomial $\boldsymbol{f}_X$ has the form*

$$c + \sum_{Y,Z \in \mathcal{X}} a_{Y,Z} \cdot Y \cdot Z + \sum_{Y \in \mathcal{X}} b_{l,Y} \cdot Y \cdot b_{r,Y}$$

where (i) $c \in S \setminus \{0\}$, (ii) $a_{Y,Z} \in \{0,1\}$, and (iii) if $\sum_{Z \in \mathcal{X}} a_{Y,Z} \neq 0$, then $b_{l,Y} \neq 0 \neq b_{r,Y}$ for all $Y, Z \in \mathcal{X}$.

Lemma 1. *For every clean $\boldsymbol{g} \in \mathcal{S}[\mathcal{X}]^{\mathcal{X}}$ we can construct in linear time a system $\boldsymbol{f} \in \mathcal{S}[\mathcal{X}']^{\mathcal{X}'}$ in quadratic normal form, where $\mathcal{X} \subseteq \mathcal{X}'$ and $\mu \boldsymbol{g}_X = \mu \boldsymbol{f}_X$ for all $X \in \mathcal{X}$.*

Proof Sketch. Note that, as $\boldsymbol{g}$ is clean, we have $\mathbf{1} \sqsubseteq \mu \boldsymbol{g}$. Hence, requirement (i) is no restriction. The transformation that normalizes a system is similar to the one that brings a context-free grammar into Chomsky normal-form (CNF). The superset $\mathcal{X}' \supset \mathcal{X}$ results from the introduction of new variables by this transformation into CNF. □

Our main result in this section is that for *strongly connected* systems $\boldsymbol{f}$ in quadratic normal form we again have that $\mu \boldsymbol{f} = \mu \boldsymbol{f}_{\mathcal{B}}$. We then show how this result leads to an algorithm for arbitrary systems.

Given two variables $X, Y \in \mathcal{X}$, we say that X depends on Y (w.r.t. $\boldsymbol{f}$) if Y occurs in a monomial of $\boldsymbol{f}_X$ or there is a variable Z such that X depends on Z and Z depends on Y. The system $\boldsymbol{f}$ is *strongly connected* if X depends on Y for all variables X, Y.

Theorem 3. *$\mu \boldsymbol{f} = \mu \boldsymbol{f}_{\mathcal{B}}$ holds for strongly connected polynomial systems $\boldsymbol{f}$ in quadratic normal form over lossy semirings.*

We again use derivation tree analysis to show that every derivation tree t can be transformed into a bamboo subsuming the yield of t, see [6] for details.

Because of the preceding theorem, given a strongly connected system $\boldsymbol{f}$, we may use the linear system $\boldsymbol{f}_{\mathcal{B}}(\boldsymbol{X}) = \boldsymbol{f}(\mathbf{0}) + D\boldsymbol{f}|_{\boldsymbol{f}^n(\mathbf{0})}(\boldsymbol{X})$ for calculating $\mu \boldsymbol{f}$. As $\boldsymbol{f}$ is strongly connected, $\boldsymbol{f}_{\mathcal{B}}$ is also strongly connected. The least fixed-point of such a strongly connected linear system $\boldsymbol{f}_{\mathcal{B}}$ is easily calculated: all non-constant monomials appearing in $\boldsymbol{f}_{\mathcal{B}}$ have the form $b_l X b_r$ for some $X \in \mathcal{X}$, and $b_l, b_r \in S \setminus \{0\}$. As $\boldsymbol{f}_{\mathcal{B}}$ is strongly connected, every polynomial $(\boldsymbol{f}_{\mathcal{B}})_Y$ is substituted for Y in $(\boldsymbol{f}_{\mathcal{B}})_X$ again and again when calculating the Kleene sequence $(\boldsymbol{f}_{\mathcal{B}}^k(\mathbf{0}))_{k\in\mathbb{N}}$. So, let l be the sum of all left-handed coefficients b_l (appearing in *any* $\boldsymbol{f}_X$), and similarly define r. We then have $(\mu\boldsymbol{f}_{\mathcal{B}})_X = l^*\big(\sum_{Y\in\mathcal{X}} \boldsymbol{f}_Y(\mathbf{0})\big)r^*$ for all $X \in \mathcal{X}$.

If $\boldsymbol{f}$ is not strongly connected, we first decompose $\boldsymbol{f}$ into strongly connected subsystems, and then we solve these systems bottom-up. Note that substituting the solutions from underlying SCCs into a given SCC leads to a new system in normal form. As there are at most $n = |\mathcal{X}|$ many strongly connected components for a given system $\boldsymbol{f} \in \mathcal{S}[\mathcal{X}]^{\mathcal{X}}$, we obtain the following theorem which was first stated explicitly for context-free grammars in [3].

Theorem 4. *The least fixed-point $\mu\boldsymbol{f}$ of a polynomial system $\boldsymbol{f}$ over a lossy semiring is representable by regular expressions over $\mathcal{S}$. If $\boldsymbol{f}$ is in normal form $\mu\boldsymbol{f}$ can be calculated solving at most n bamboo systems.*

7 1-bounded Semirings

Definition 7. *An io-semiring $\mathcal{S}$ is called* 1-bounded *if $a \sqsubseteq 1$ holds for all $a \in S$.*

Natural examples are the tropical semiring over the natural numbers $(\mathbb{N}\cup\{\infty\}, \wedge, +, \infty, 0)$ and the "maximum-probability" semiring $([0,1], \vee, \cdot, 0, 1)$, where $\wedge$ and $\vee$ denote minimum and maximum, respectively. Notice that any commutative 1-bounded semiring is star-distributive (as $a^* = 1$ for all a), but not all 1-bounded semirings have commutative multiplication. Consider for example the semiring of those languages L over Σ that are *upward-closed*, i.e., $w \in L$ implies $u \in L$ for all u such that w is a subword of u. This semiring is 1-bounded and has Σ^* as 1-element. Upward-closed languages form a natural dual to downward-closed languages from the previous section.
We show that $\mu\boldsymbol{f}$ can be computed very easily in the case of 1-bounded semirings:

Theorem 5. *$\mu\boldsymbol{f} = \boldsymbol{f}^n(\mathbf{0})$ holds for polynomial systems over 1-bounded semirings.*

Proof Sketch. Recall that, by Proposition 2, we have $(\boldsymbol{f}^n(\mathbf{0}))_X = \mathsf{Y}(\mathcal{T}_X^{(n-1)})$, where $\mathcal{T}_X^{(n-1)}$ contains all X-trees of height at most $n-1$. We proceed by derivation tree analysis, i.e., by discharging the precondition of Proposition 3. So it suffices to show that for any X-tree t there is an X-tree t' of height at most $n-1$ with $\mathsf{Y}(t) \sqsubseteq \mathsf{Y}(t')$. Such a tree t' can be constructed by pruning t as long as some variable label occurs more than once along any path. $\square$

Theorem 5 appears to be rather easy from our point of view, i.e., from the point of view of derivation trees. However, even this simple result has very concrete applications in the domain of interprocedural program analysis [14]. The main algorithms of [14], the so-called *post** and *pre** algorithms, can be seen as solvers of fixed-point equations over *bounded* semirings, which are semirings that do not have infinite ascending chains. Those solvers are based on Kleene's iteration and the complexity result given there depends on the maximal length of ascending chains in the semiring (cf. [14], page 28). Such a bound may not exist, and does not exist for the tropical semiring over the natural numbers $(\mathbb{N}\cup\{\infty\}, \wedge, +, \infty, 0)$ which is considered as an example in [14], pages 13 and 18. However, Theorem 5 can be applied to this semiring, which shows that the program analysis algorithms of [14] applied to 1-bounded semirings are polynomial-time algorithms, independent of the length of chains in the semiring.

8 Conclusion

We have shown that derivation tree analysis, a proof technique first introduced in [5], is an efficient tool for the design of efficient fixed-point algorithms on io-semirings. We have considered three classes of io-semirings with applications to language theory and verification. We have shown that for star-distributive semirings and lossy semirings the least fixed-point of a polynomial system of equations is equal to the least fixed-point of a *linear system*, the bamboo system. This improves the results of [5]: The generic algorithm given there requires to solve N different systems of linear equations in the star-distributive case (where N is the original number of polynomial equations), and is not applicable to the lossy case.

We have used our results to design an efficient fixed-point algorithm for the $(\min, +)$-semiring. In turn, we have applied this algorithm to provide a cubic algorithm for computing the throughput of a context-free language, improving the $\mathcal{O}(N^4)$ upper bound obtained by Caucal et al. in [2].

For lossy semirings, derivation tree analysis based on bamboos has led to an algebraic generalization of a result of Courcelle stating that the downward-closure of a context-free language is effectively regular. Finally we have used derivation tree analysis to derive a simple proof that $\mu\boldsymbol{f} = \boldsymbol{f}^n(\mathbf{0})$ holds for 1-bounded semirings, with some applications in interprocedural program analysis.

References

1. P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *CAV'98*, LNCS 1427, pages 305–318. Springer, 1998.
2. D. Caucal, J. Czyzowicz, W. Fraczak, and W. Rytter. Efficient computation of throughput values of context-free languages. In *CIAA'07*, LNCS 4783, pages 203–213. Springer, 2007.
3. B. Courcelle. On constructing obstruction sets of words. *EATCS Bulletin*, 44:178–185, 1991.
4. J. Esparza, S. Kiefer, and M. Luttenberger. An extension of Newton's method to ω-continuous semirings. In *DLT'07*, LNCS 4588, pages 157–168. Springer, 2007.
5. J. Esparza, S. Kiefer, and M. Luttenberger. On fixed point equations over commutative semirings. In *STACS'07*, LNCS 4397, pages 296–307. Springer, 2007.
6. J. Esparza, S. Kiefer, and M. Luttenberger. Derivation tree analysis for accelerated fixed-point computation. Technical report, Technische Universität München, 2008.
7. J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2006.
8. K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In *STACS*, pages 340–352, 2005.
9. T. Gawlitza and H. Seidl. Precise fixpoint computation through strategy iteration. In *ESOP'07*, LNCS 4421, pages 300–315. Springer, 2007.
10. T.E. Harris. *The Theory of Branching Processes*. Springer, 1963.
11. G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 2, 1952.
12. M. W. Hopkins and D. Kozen. Parikh's theorem in commutative Kleene algebra. In *LICS'99*.
13. F. Nielson, H.R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
14. T. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Science of Computer Programming*, 58(1–2):206–263, October 2005. Special Issue on the Static Analysis Symposium 2003.