Newtonian Program Analysis

JAVIER ESPARZA, STEFAN KIEFER, AND MICHAEL LUTTENBERGER

Technische Universität München

Abstract. This article presents a novel generic technique for solving dataflow equations in interprocedural dataflow analysis. The technique is obtained by generalizing Newton's method for computing a zero of a differentiable function to ω-continuous semirings. Complete semilattices, the common program analysis framework, are a special class of ω-continuous semirings. We show that our generalized method always converges to the solution, and requires at most as many iterations as current methods based on Kleene's fixed-point theorem. We also show that, contrary to Kleene's method, Newton's method always terminates for arbitrary idempotent and commutative semirings. More precisely, in the latter setting the number of iterations required to solve a system of n equations is at most n.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Program Verification—*Formal Methods*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program analysis*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*algebraic language theory*

General Terms: Languages, Theory, Verification

Additional Key Words and Phrases: Newton's method, polynomial fixed-point equations, interprocedural program analysis, semirings

ACM Reference Format:

Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.
DOI = 10.1145/1857914.1857917 http://doi.acm.org/10.1145/1857914.1857917

1. *Introduction*

This article presents a novel generic technique for solving dataflow equations in interprocedural dataflow analysis. It is obtained by generalizing Newton's method, the 300-year-old technique for computing a zero of a differentiable function.

Our approach to interprocedural analysis is very similar to Sharir and Pnueli's functional approach [Sharir and Pnueli 1981; Jones and Muchnick 1982; Knoop

This work was partially supported by the DFG project *Algorithms for Software Model Checking*.

Preliminary versions of this work have appeared in Esparza et al. [2007a, 2007b].

Authors' address: J. Esparza, S. Kiefer, and M. Luttenberger, Institut für Informatik, Technische Universität München, 85748 Garching, Germany, e-mail: {esparza,kiefer,luttenbe}@in.tum.de.

DOI 10.1145/1857914.1857917 http://doi.acm.org/10.1145/1857914.1857917

and Steffen 1992; Reps et al. 1995, 2005; Sagiv et al. 1996; Nielson et al. 1999] Sharir and Pnueli [1981] assume the following as given: a (join-) semilattice[1] of *values*, a mapping assigning to every program instruction a value, and a concatenation operator that, given the values of two sequences of instructions, returns the value corresponding to their concatenation. Sharir and Pnueli assume that the concatenation operator distributes over the lattice's join.[2] Sharir and Pnueli define a system of *abstract data flow equations*, containing one variable for each program point. They show that for every procedure P of the program and for every program point p of P, the least solution of the system is the join of the values of all valid program paths starting at the initial node of P and leading to p. Sharir and Pnueli's result was later extended by Knoop and Steffen [1992] to programs with local variables and to non-distributive concatenation operators, which allows us to deal with certain nondistributive analyses [Nielson et al. 1999].

We slightly generalize Sharir and Pnueli's setting. Loosely speaking, we allow to replace the join operator with any operator satisfying the same algebraic properties with the possible exception of idempotence. In algebraic terms, we extend the framework from lattices considered in Sharir and Pnueli [1981] to *ω-continuous semirings* [Kuich 1997], an algebraic structure with two operations, usually called sum and product. The interest of this otherwise simple extension is that our framework now encompasses equations over the semiring of the nonnegative reals with addition and multiplication. This allows us to compare the efficiency of generic solution methods for dataflow analysis when applied to the reals with the efficiency of methods supplied by classic numerical mathematics, in particular Newton's method.

It is well-known that Newton's method, when it converges to a solution, usually converges much faster than classical fixed-point iteration (see, e.g., Ortega and Rheinboldt [1970]). Furthermore, Etessami and Yannakakis [2009] have recently proved that Newton's method is guaranteed to converge for an analysis concerning the probability of termination of recursive programs. These facts raise the question whether Newton's method can be generalized to the more abstract dataflow setting, where values are arbitrary entities, while preserving the good properties of Newton's method.

In the first part of the article, we show that the generalization is indeed possible. Inspired by work of Hopkins and Kozen [1999] on Kleene algebras, we show that the notion of a differential of a function lying at the heart of Newton's method, and the method itself can be suitably generalized. This allows us to apply Newton's method to, for instance, language equations. We then apply the method to two small case studies: a may-alias analysis and an average runtime analysis.

In the second part of this article, we study the properties of Newton's method on idempotent semirings, the classical domain of program analysis. Recall that the method is iterative: it constructs better and better approximations to the solution

[1] For reasons that will be clear later, we use join-semilattices rather than meet-semilattices, deviating from the classical dataflow analysis literature such as Kildall [1973], Kam and Ullman [1977], and Sharir and Pnueli [1981]. As a consequence, we also replace greatest fixed points by least fixed points, meet-over-all-paths by join-over-all-paths, etc. This change is purely notational.

[2] Actually, in Sharir and Pnueli [1981], the value of a program instruction is the function describing its effect on program variables, and the extension operator is function composition. However, the extension to an arbitrary distributive concatenation operator is unproblematic.

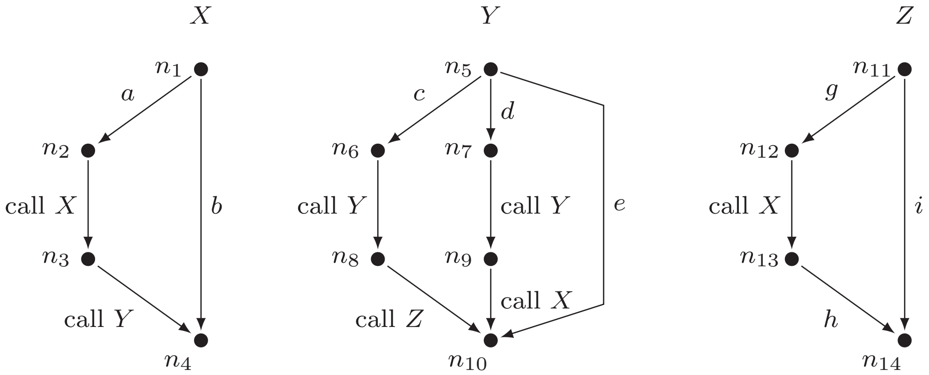


FIG. 1. Flowgraphs of three procedures.

of the equation system. We obtain a characterization of the approximants, and apply it to the case of commutative idempotent semirings, previously studied by Hopkins and Kozen in a beautiful paper [1999]. Hopkins and Kozen propose a generic solution method for the equations, and prove that it terminates after $\mathcal{O}(3^n)$ iterations, where n is the number of equations. We show that their method is in fact Newton's method, and, applying our characterization of the approximants, show that it terminates after at most n iterations.

Finally, in a short section we extend our framework to the non-distributive case. We show that Newton's method, like the classical fixed-point iteration, computes an overapproximation of the join of the values of all valid program paths.

In the rest of this introduction, we go again through the article's skeleton sketched above, but providing some more details.

1.1. A Summary of Sharir and Pnueli's Approach.

Sharir and Pnueli [1981] assume as given a lattice of data values with a join operator. They show how to compute for every program point p of every procedure P the join of the values of all valid program paths leading from the initial node of P to p. This is called the *join-over-all-valid-paths* for p, or JOP(p) for short. The computation, which works for distributive analyses, proceeds in two steps: first, the join over all *same-level* valid program paths is computed, where a path is same-level if every procedure call has a matching return. We denote this join by $\text{JOP}_0(p)$. The second step is usually described today in terms of *summary edges* (see, e.g., Reps et al. [1995]). $\text{JOP}_0(p)$ is used to construct a new flowgraph without procedure calls. Edges calling P are replaced by edges with the same source and target nodes, but labelled with $\text{JOP}_0(ex_P)$ (the *effect* of P) where ex_P is the exit node of P; new edges are added leading from the source of each call to P to P's entry point. The result is a flowgraph without procedure calls, such that JOP(p) for the old and new graphs coincide. The JOP for flowgraphs without procedures (the *intra*procedural case) is the least solution of a system of linear dataflow equations [Kildall 1973; Kam and Ullman 1977].

Sharir and Pnueli [1981] show that JOP_0 is equal to the least solution of a system of dataflow equations. We sketch how to construct the equations by means of an example. Consider a program with three procedures X, Y, Z, whose flowgraphs are shown in Figure 1. Nodes correspond to program points, and edges to program instructions. For instance, procedure X can execute b and terminate, or execute a, call itself recursively, and, after the call has terminated, call Y.

To define the equations, Sharir and Pnueli assume a complete lattice[3] of values with a join operator $\vee$; a mapping ϕ assigning to each non-call edge (m, n) a lattice value $\phi(m, n)$, and a concatenation operator $\cdot$ that distributes over $\vee$ and has a neutral element (1). The system of equations contains a variable and an equation for each program node. If n is the initial node of a procedure then it contributes the equation $v_n = 1$, where v_n denotes n's variable. Otherwise, it contributes the equation

$$v_n = \bigvee_{m \in pred(n)} v_m \cdot h(m, n)$$

where $pred(n)$ denotes the set of immediate predecessors of n, and $h(m, n)$ is defined as follows: if (m, n) is a call edge calling, for example, procedure X, then $h(m, n)$ is the variable for the return node of X; otherwise $h(m, n) = \phi(m, n)$.

The system of equations for Figure 1 can be more compactly represented if variables for all program points other than return points are eliminated by substitution. Only three equations remain, namely those for the return points n_4, n_{10}, and n_{14}. If moreover, and abusing language, we reuse X, Y, Z to denote the variables for these points, and $a, \ldots, i$ to denote the values $\phi(n_1, n_2), \ldots, \phi(n_{11}, n_{14})$, we obtain the system

$$\begin{aligned} X &= a \cdot X \cdot Y \ \vee \ b \\ Y &= c \cdot Y \cdot Z \ \vee \ d \cdot Y \cdot X \ \vee \ e \\ Z &= g \cdot X \cdot h \ \vee \ i \end{aligned} \tag{1}$$

which very closely resembles the structure of the flowgraphs. Since the right-hand sides of the equations are monotonic mappings, and distributes over $\vee$, the existence of the least fixed point is guaranteed by Kleene's fixed-point theorem.

1.2. A Slight Generalization: From Semilattices to Semirings.

Let us examine the properties of the join operator $\vee$. First of all, since the lattice is complete, it is defined for arbitrary sets of lattice elements. Furthermore, it is associative, commutative, idempotent, and concatenation distributes over it. If we use the symbols 0 for the bottom element of the lattice (corresponding to an abort operation) and 1 for the element corresponding to a NOP instruction, then we have $0 \vee a = a \vee 0 = a$ and $1 \cdot a = a \cdot 1 = a$ for every a. It is argued in Seidl and Fecht [2000] that one can transform every program analysis to an essentially equivalent one that satisfies $0 \cdot a = a \cdot 0 = 0$. So the lattice, together with the two operations $\vee$ and $\cdot$ and the elements 0 and 1, constitutes an *idempotent semiring*. In the following, we write '$+$' for '$\vee$' to conform with the standard semiring notation.

Idempotence of the join operator is not crucial for the existence of the least fixed point; it can be replaced by a weaker property. Consider the relation $\sqsubseteq$ on semiring elements defined as follows: $a \sqsubseteq a + b$ for all elements a, b. A semiring is

[3] More precisely, Sharir and Pnueli [1981] initially consider semilattices with a least and a greatest element that satisfy the ascending-chain property (every nondecreasing chain eventually becomes stationary). However, the paper later concentrates on finite lattices, which are complete.

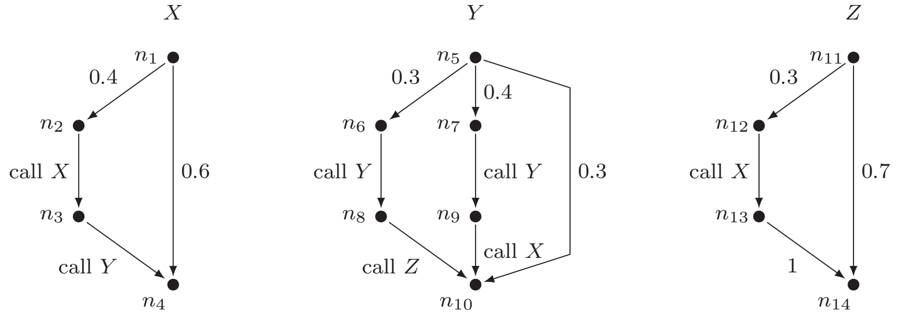


FIG. 2. Probabilistic flowgraphs.

naturally ordered if this relation is a partial order, and a naturally ordered semiring in which infinite sums exist and satisfy standard properties is called *ω-continuous*. Using Kleene's fixed-point theorem it is easy to show that systems of equations over ω-continuous semirings still have a least fixed point with respect to the partial order $\sqsubseteq$ (see, for instance Kuich [1997]).

As an example of application of this more general setting, assume that the program of Figure 1 is probabilistic, and the values $a, \ldots, i$ are real numbers corresponding to the probabilities of taking the transitions. A particular case is shown in Figure 2. The semiring operations are addition and multiplication over the nonnegative reals. Notice that addition is not idempotent. The semiring is ω-continuous if a new element ∞ with the usual properties is added. It is not difficult to show [Esparza et al. 2004; Etessami and Yannakakis 2009] that the least solution of the system

$$\begin{aligned} X &= 0.4XY + 0.6 \\ Y &= 0.3YZ + 0.4YX + 0.3 \\ Z &= 0.3X + 0.7 \end{aligned}$$

yields the probability of termination of each procedure. (Incidentally, notice that, contrary to the intraprocedural case, this probability may be different from 1 even if every execution can be extended to a terminating execution.)

1.3. SOLVING SYSTEMS OF EQUATIONS. Current generic algorithms for solving Sharir and Pnueli's equations (like the classical worklist algorithm of dataflow analysis) are based on variants of Kleene's fixed-point theorem [Kuich 1997]. The theorem states that the least solution $\mu \boldsymbol{f}$ of a system of equations $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ over an ω-continuous semiring is equal to the supremum of the sequence $(\boldsymbol{\kappa}^{(i)})_{i\in\mathbb{N}}$ of *Kleene approximants* given by $\boldsymbol{\kappa}^{(0)} = \mathbf{0}$ (the vector of 0-elements) and $\boldsymbol{\kappa}^{(i+1)} = \boldsymbol{f}(\boldsymbol{\kappa}^{(i)})$. This yields a procedure (let us call it *Kleene's method*) to compute or at least approximate $\mu \boldsymbol{f}$. If the domain satisfies the well-known *ascending chain condition* [Nielson et al. 1999], then the procedure terminates, because there exists an i such that $\boldsymbol{\kappa}^{(i)} = \boldsymbol{\kappa}^{(i+1)} = \mu \boldsymbol{f}$.

Kleene's method is generic and robust: it always converges when started at $\mathbf{0}$, for any ω-continuous semiring and for any system of equations. On the other hand, it often fails to terminate, and it can converge very slowly to the solution. We illustrate this point by means of two simple examples. Consider the equation

$X = a \cdot X + b$ over the lattice of subsets of the language $\{a, b\}^*$. The least solution is the regular language a^*b, but we have $\boldsymbol{\kappa}^{(i)} = \{b, ab, \ldots, a^{i-1}b\}$ for $i \geq 1$, that is, the solution is not reached in any finite number of steps. For our second example, consider a very simple probabilistic procedure that can either terminate or call itself twice, both with probability $1/2$. The probability of termination of this program is given by the least solution of the equation $X = 1/2 + 1/2 \cdot X^2$ (where X^2 abbreviates $X \cdot X$). It is easy to see that the least solution is equal to 1, but we have $\kappa^{(i)} \leq 1 - \frac{1}{i+1}$ for every $i \geq 0$, that is, in order to approximate the solution within i bits of precision we have to compute about 2^i Kleene approximants. For instance, we have $\kappa^{(200)} = 0.9990$, that is, 200 iterations produce only three digits of precision.

After our slight generalization of Sharir and Pnueli's framework, quantitative analyses like the probability of termination fall within the scope of the approach. So we can look at numerical mathematics for help with the inefficiencies of Kleene's method.

As could be expected, faster approximation techniques for equations over the reals have been known for a long time. In particular, Newton's method, suggested by Isaac Newton more than 300 years ago, is a standard efficient technique to approximate a zero of a differentiable function, and can be adapted to our problem. Since the least solution of $X = 1/2 + 1/2 \cdot X^2$ is a zero of $1/2 + 1/2 \cdot X^2 - X$, the method can be applied, and it yields $\nu^{(i)} = 1 - 2^{-i}$ for the ith *Newton approximant*. So the ith Newton approximant already has i bits of precision, instead of $\log i$ bits for the Kleene approximant.

However, Newton's method also has a number of disadvantages, at least at first sight. Newton's method on the real field is by far not as robust and well behaved as Kleene's method on semirings. The method may converge very slowly, converge only locally (only when started in a small neighborhood of the zero), or even not converge at all [Ortega and Rheinboldt 1970]. So we face the following situation. Kleene's method, a robust and general solution technique for arbitrary ω-continuous semirings, is inefficient in many cases. Newton's method is usually very efficient, but it is only defined for the real field, and it is not robust.

As part of their study of Recursive Markov Chains, Etessami and Yannakakis [2009] showed that a variant of Newton's method is robust for certain systems of equations over the real *semiring*: the method always converges when started at zero. In other words, moving from the real field to the real semiring (only nonnegative numbers) makes the instability problems disappear. Inspired by this work, in this paper we obtain a more general result. We show that Newton's method can be generalized to *arbitrary* ω-continuous semirings, and prove that on these structures it is as robust as Kleene's method. Since lattices, the classical domain of program analysis, are very close to idempotent semirings, we study in detail Newton's method in idempotent semirings. We pay special attention to idempotent semirings with commutative multiplication. Loosely speaking, these semirings correspond to *counting analysis*, in which one is interested in how often program points are visited, but not in which order. These semirings do not always satisfy the ascending chain condition, and Kleene's method may not terminate. We show that a very elegant iterative solution method for these semirings due to Hopkins and Kozen [1999], is exactly Newton's method, and always terminates in a finite number of steps. As mentioned above, we further use our characterization of Newton approximants to show that the least fixed point is reached after at most n iterations, a tight bound, improving on the $\mathcal{O}(3^n)$ bound of Hopkins and Kozen [1999].

The article is divided into two parts. The first part introduces our generalization of Newton's method, and ends with two examples of application to program analysis problems: a may-alias analysis for a program transforming a tree into a list, and an average runtime analysis for lazy evaluation of And/Or-trees. The second part presents the proofs of our results, investigates Newton's method in idempotent and commutative semirings, and extends our approach to semi-distributive program analyses. It is wellknown that in this case fixed-point iteration overapproximates the join-over-all-paths value (see, e.g., Knoop and Steffen [1992], Reps et al. [1995], Sagiv et al. [1996], and Nielson et al. [1999]). We show that the same property holds for Newton's method.

The first part of the article is organized as follows. Section 2 introduces ω-continuous semirings, systems of fixed-point equations, and some semirings investigated in the rest of the article. Section 3 recalls Newton's method, and generalizes it to arbitrary ω-continuous semirings. Section 4 presents the case studies. The second part starts with Section 5 where we prove the fundamental properties of our generalization, mainly convergence to the least fixed point. Section 6 characterizes the Newton approximants in terms of *derivation trees*, a generalization of the derivation trees of language theory. Section 7 uses this characterization to prove that for idempotent and commutative semirings Newton's method always terminates in at most n iterations for a system of dimension n. Finally, Section 8 deals with nondistributive program analyses.

2. *ω-Continuous Semirings*

Definition 2.1. A *semiring* is a tuple $\langle S, +, \cdot, 0, 1\rangle$ where S is a set containing two distinguished elements 0 and 1, and the binary operations $+, \cdot \colon S \times S \to S$ satisfy the following conditions:

(1) $\langle S, +, 0\rangle$ is a commutative monoid.

(2) $\langle S, \cdot, 1\rangle$ is a monoid.

(3) $0 \cdot a = a \cdot 0 = 0$ for all $a \in S$.

(4) $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in S$.
A semiring $\langle S, +, \cdot, 0, 1\rangle$ is *ω-continuous* if the following additional conditions hold:

(5) The relation $\sqsubseteq := \{(a, b) \in S \times S \mid \exists d \in S : a + d = b\}$ is a partial order.

(6) Every *ω-chain* $(a_i)_{i\in\mathbb{N}}$ (i.e., $a_i \sqsubseteq a_{i+1}$ with $a_i \in S$) has a supremum with respect to $\sqsubseteq$ denoted by $\sup_{i\in\mathbb{N}} a_i$.

(7) Given an arbitrary sequence $(b_i)_{i\in\mathbb{N}}$, define

$$\sum_{i\in\mathbb{N}} b_i := \sup\{b_0 + b_1 + \cdots + b_i \mid i \in \mathbb{N}\}$$

(the supremum exists by condition (6)). For every sequence $(a_i)_{i\in\mathbb{N}}$, for every $c \in S$, and for every partition $(I_j)_{j\in J}$ of $\mathbb{N}$:

$$c\cdot\left(\sum_{i\in\mathbb{N}} a_i\right) = \sum_{i\in\mathbb{N}}(c\cdot a_i),\ \left(\sum_{i\in\mathbb{N}} a_i\right)\cdot c = \sum_{i\in\mathbb{N}}(a_i\cdot c),\ \sum_{j\in J}\left(\sum_{i\in I_j} a_j\right) = \sum_{i\in\mathbb{N}} a_i\,.$$

An (ω-continuous) semiring is *idempotent*, if $a + a = a$ holds for all $a \in S$. It is *commutative*, if $a \cdot b = b \cdot a$ for all $a, b \in S$. In an ω-continuous semiring we define the Kleene-star $^* : S \to S$ by

$$a^* := \sum_{k\in\mathbb{N}} a^k = \sup\{1 + a + a\cdot a + \cdots + a^k \mid k \in \mathbb{N}\} \text{ for } a \in S.$$

For ω-continuous semirings, we have the following important property that addition and multiplication, and subsequently polynomials are ω-continuous, too.

LEMMA 2.2. *In any ω-continuous semiring $\langle S, +, \cdot, 0, 1\rangle$ addition and multiplication are ω-continuous, that is, for any ω-chain $(a_i)_{i\in\mathbb{N}}$ and any $c \in S$ we have*

$$c\cdot(\sup_{i\in\mathbb{N}} a_i) = \sup_{i\in\mathbb{N}}(c\cdot a_i), \quad (\sup_{i\in\mathbb{N}} a_i)\cdot c = \sup_{i\in\mathbb{N}}(a_i\cdot c), \quad c + (\sup_{i\in\mathbb{N}} a_i) = \sup_{i\in\mathbb{N}}(c + a_i).$$

PROOF. By (5) and (6) in the previous definition, for any ω-chain $(a_i)_{i\in\mathbb{N}}$, there exists a sequence $(d_i)_{i\in\mathbb{N}}$ such that $d_0 = a_0$ and $a_i + d_i = a_{i+1}$ (i.e., d_i is *a difference* of a_{i+1} and a_i), and so $\sup_{i\in\mathbb{N}} a_i = \sum_{i\in\mathbb{N}} d_i$. The result follows by applying (7) to this sequence. □

Example 2.3. Common examples of ω-continuous semirings are the *real semiring*, that is, nonnegative real numbers extended by infinity $\langle\mathbb{R}_{\geq 0}\cup\{\infty\}, +, \cdot, 0, 1\rangle$, and the language semiring over some finite alphabet Σ, that is, $\langle 2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\}\rangle$ where $\cdot$ stands for the canonical concatenation of languages, and ε for the empty word. In both of these instances the natural order coincides with the canonical order on the respective carrier, that is, in the real semiring we have $\sqsubseteq\,\equiv\,\leq$, and in the language semiring $\sqsubseteq\,\equiv\,\subseteq$.

In the following, we often write ab instead of $a \cdot b$.

2.1. VECTORS, POLYNOMIALS AND POWER SERIES. Let $\mathcal{S}$ be an ω-continuous semiring and let $\mathcal{X}$ be a finite set of variables. A *vector* is a mapping $\boldsymbol{v} : \mathcal{X} \to S$ which assigns every variable $X \in \mathcal{X}$ the value $\boldsymbol{v}(X)$. We usually write $\boldsymbol{v}_X$ for $\boldsymbol{v}(X)$. If there is some natural total order given on $\mathcal{X}$ like, for example, the lexicographic order in the case $\mathcal{X} = \{X, Y, Z\}$ or the total order on the indices in the case $\mathcal{X} = \{X_1, X_2, X_3\}$ we will also write a vector $\boldsymbol{v}$ as a column vector of dimension $|\mathcal{X}|$ enumerating the values starting with the lowest variable as the topmost value. The set of all vectors is denoted by V.

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 $X \in \mathcal{X}$. Throughout the article, we use bold letters like '$\boldsymbol{v}$' or '$\boldsymbol{a}$' for vectors.

A *monomial* is a finite expression $a_1X_1a_2X_2\cdots a_kX_ka_{k+1}$, where $k \geq 0$, $a_1, \ldots, a_{k+1} \in S$ and $X_1, \ldots, X_k \in \mathcal{X}$. Note that this general definition of monomial is necessary as we do not require that multiplication is commutative. 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. A *power series* is an expression of the form $\sum_{i \in I} m_i$, where I is a countable set and m_i is a monomial for every $i \in I$.

Given a monomial $f = a_1 X_1 a_2 X_2 \ldots a_k X_k a_{k+1}$ and a vector $\boldsymbol{v}$, we define $f(\boldsymbol{v})$, the *value of f at $\boldsymbol{v}$*, as $a_1 \boldsymbol{v}_{X_1} a_2 \boldsymbol{v}_{X_2} \cdots a_k \boldsymbol{v}_{X_k} a_{k+1}$. We extend this to any power series $f = \sum_{i \in I} f_i$ by $f(\boldsymbol{v}) = \sum_{i \in I} f_i(\boldsymbol{v})$.

A *vector of power series* is a mapping $\boldsymbol{f}$ that assigns to each variable $X \in \mathcal{X}$ a power series $\boldsymbol{f}(X)$. Again we write $\boldsymbol{f}_X$ for $\boldsymbol{f}(X)$. Given a vector $\boldsymbol{v}$, we define $\boldsymbol{f}(\boldsymbol{v})$ as the vector satisfying $(\boldsymbol{f}(\boldsymbol{v}))_X = \boldsymbol{f}_X(\boldsymbol{v})$ for every $X \in \mathcal{X}$, that is, $\boldsymbol{f}(\boldsymbol{v})$ is the vector that assigns to X the result of evaluating the power series $\boldsymbol{f}_X$ at $\boldsymbol{v}$. So, $\boldsymbol{f}$ naturally induces a mapping $\boldsymbol{f} \colon V \to V$.

2.2. FIXED-POINT EQUATIONS AND KLEENE'S THEOREM. The partial order $\sqsubseteq$ on the semiring $\mathcal{S}$ can be lifted to a partial order on vectors, also denoted by $\sqsubseteq$, and defined by $\boldsymbol{v} \sqsubseteq \boldsymbol{v}'$ if $\boldsymbol{v}_X \sqsubseteq \boldsymbol{v}'_X$ for every $X \in \mathcal{X}$.

Given a vector of power series $\boldsymbol{f}$, we are interested in the least fixed point of $\boldsymbol{f}$, that is, the least vector $\boldsymbol{v}$ with respect to $\sqsubseteq$ satisfying $\boldsymbol{v} = \boldsymbol{f}(\boldsymbol{v})$. We briefly recall Kleene's theorem, which guarantees that the least fixed point exists.

A mapping $f \colon \mathcal{S} \to \mathcal{S}$ is *monotone* if $a \sqsubseteq b$ implies $f(a) \sqsubseteq f(b)$, and ω-*continuous* if for any infinite chain $a_0 \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq \cdots$ we have $\sup\{f(a_i)\} = f(\sup\{a_i\})$. These definitions are extended to mappings $\boldsymbol{f} \colon V \to V$ from vectors to vectors by requiring them to hold in every component of $\boldsymbol{f}$. The following result is taken from Kuich [1997] and relies on the fact that multiplication and addition are ω-continuous on ω-continuous semirings, see Lemma 2.2.

PROPOSITION 2.4. *Let $\boldsymbol{f}$ be a vector of power series. The mapping induced by $\boldsymbol{f}$ is monotone and ω-continuous. Hence, by Kleene's theorem, $\boldsymbol{f}$ has a unique least fixed point $\mu\boldsymbol{f}$. Further, $\mu\boldsymbol{f}$ is the supremum (with respect to $\sqsubseteq$) of the* Kleene sequence *given by $\boldsymbol{\kappa}^{(0)} = \boldsymbol{f}(\boldsymbol{0})$, and $\boldsymbol{\kappa}^{(i+1)} = \boldsymbol{f}(\boldsymbol{\kappa}^{(i)})$.*[4]

2.3. SOME SEMIRING INTERPRETATIONS. We recall that different interesting pieces of information about the program of Figure 1 correspond to the least solution of Equations (1) from page 4 over different semirings.[5] For the rest of the section let $\Sigma = \{a, b, \ldots, i\}$ be the set of actions in the program, and let σ denote an arbitrary element of Σ.

2.3.1. *Language Interpretation.* Consider the following semiring. The carrier is 2^{Σ^*} (i.e., the set of languages over Σ). The semiring element σ is interpreted as the singleton language $\{\sigma\}$. The sum and product operations are union and concatenation of languages, respectively. We call this structure *language semiring* over Σ. Under this interpretation, Eq. (1) are nothing but the following context-free grammar in Backus-Naur form:

$$X \to aXY \mid b \qquad Y \to cYZ \mid dYX \mid e \qquad Z \to gXh \mid i$$

The least solution of (1) is the triple $(L(X), L(Y), L(Z))$, where, for $U \in \{X, Y, Z\}$, $L(U)$ denotes the set of terminating executions of the program with U as main

[4] Defining $\boldsymbol{\kappa}^{(0)} = \boldsymbol{0}$ would be more straightforward, but less convenient for this article.

[5] This will be no surprise for the reader acquainted with abstract interpretation, but the examples will be used all throughout the article.

procedure, or, in language-theoretic terms, the language of the associated grammar with U as axiom.

2.3.2. *Relational Interpretation.* Assume that an action σ corresponds to a program instruction whose semantics is described by means of a relation $R_\sigma(V, V')$ over a set V of program variables (as usual, primed and unprimed variables correspond to the values before and after executing the instruction). Consider now the following semiring. The carrier is the set of all relations over (V, V'). The semiring element σ is interpreted as the relation R_σ. The sum and product operations are union and join of relations, respectively, that is, $(R_1 \cdot R_2)(V, V') = \exists V'' R_1(V, V'') \wedge R_2(V'', V')$. Under this interpretation, the U-component of the least solution of (1) is the *summary* relation $R_U(V, V')$ containing the pairs V, V' such that if procedure U starts at valuation V, then it may terminate at valuation V'.

2.3.3. *Counting Interpretation.* Assume we wish to know how many as, bs, etc. we can observe in a (terminating) execution of the program, but we are not interested in the order in which they occur. In the terminology of abstract interpretation, we abstract an execution $w \in \Sigma^*$ by the vector $(n_a, \ldots, n_i) \in \mathbb{N}^{|\Sigma|}$ where $n_a, \ldots, n_i$ are the number of occurrences of $a, \ldots, i$ in w. We call $(n_a, \ldots, n_i)$ the *Parikh image* of w. The Parikh images of $L(X)$, $L(Y)$, $L(Z)$ are the least solution of (1) for the following semiring. The carrier is $2^{\mathbb{N}^{|\Sigma|}}$. The jth action of Σ is interpreted as the singleton set $\{(0, \ldots, 0, 1, 0 \ldots, 0)\}$ with the "1" at the jth position. The sum operation is set union, and the product operation is given by

$$S \cdot T = \{(s_a + t_a, \ldots, s_i + t_i) \mid (s_a, \ldots, s_i) \in S, (t_a, \ldots, t_i) \in T\} .$$

2.3.4. *Probabilistic Interpretations.* Assume that the choices between actions are stochastic. For instance, actions a and b are chosen with probability p and $(1 - p)$, respectively. The probability of termination is given by the least solution of (1) when interpreted over the following semiring (the *real semiring*) [Esparza et al. 2004; Etessami and Yannakakis 2009]. The carrier is the set of nonnegative real numbers, enriched with an additional element ∞. The semiring element σ is interpreted as the probability of choosing σ among all enabled actions. Sum and product are the standard operations on real numbers, suitably extended to ∞.

Assume now that actions are assigned not only a probability, but also a *duration*. Let d_σ denote the duration of σ. We are interested in the expected termination time of the program, under the condition that the program terminates (the *conditional expected time*). For this we consider the following semiring. The elements are the set of pairs (r_1, r_2), where r_1, r_2 are nonnegative reals or ∞. We interpret σ as the pair (p_σ, d_σ), that is, the probability and the duration of σ. The sum operation is defined as follows (where to simplify the notation we use $+_e$ and $\cdot_e$ for the operations of the semiring, and $+$ and $\cdot$ for sum and product of reals):

$$\begin{aligned}
(p_1, d_1) +_e (p_2, d_2) &= \left(p_1 + p_2, \frac{p_1 \cdot d_1 + p_2 \cdot d_2}{p_1 + p_2}\right) \\
(p_1, d_1) \cdot_e (p_2, d_2) &= (p_1 \cdot p_2, d_1 + d_2)
\end{aligned}$$

The reader can easily check that this definition satisfies the semiring axioms. The U-component of the least solution of (1) is now the pair (t_U, e_U), where t_U is the probability that procedure U terminates, and e_U is its conditional expected time.

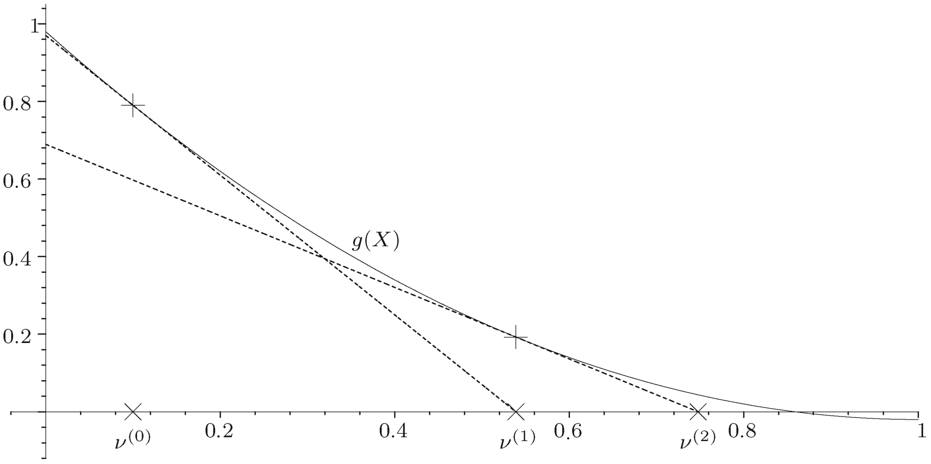


FIG. 3. Newton's method to find a zero of a one-dimensional function $g(X)$.

3. *Newton's Method for ω-Continuous Semirings*

We introduce our generalization of Newton's method for ω-continuous semirings. In Section 3.1, we consider the univariate case, i.e. the case of one equation in a single variable, which already allows us to introduce all important ideas. Here, we first recall Newton's method as known from calculus, that is, as a method for approximating a zero of a differentiable function. We then take a close look at the analytical definition, and identify the obstacles for a generalization to ω-continuous semirings. Finally, we propose a definition that overcomes the obstacles. In Section 3.2, we turn to the multivariate case and state a fundamental theorem which shows that our generalization of Newton's method is well defined and converges to the least fixed point. This lays the foundation to what we call *Newtonian program analysis*, the application of the generalized version of Newton's method to program analysis. We illustrate the concepts at the end of this section.

3.1. THE UNIVARIATE CASE. Given a differentiable function $g : \mathbb{R} \to \mathbb{R}$, Newton's method computes a zero of g, that is, a solution of the equation $g(X) = 0$. The method starts at some value $\nu^{(0)}$ "close enough" to the zero, and proceeds iteratively: given $\nu^{(i)}$, it computes a value $\nu^{(i+1)}$ closer to the zero than $\nu^{(i)}$. For that, the method *linearizes* g at $\nu^{(i)}$, that is, computes the tangent to g passing through the point $(\nu^{(i)}, g(\nu^{(i)}))$, and takes $\nu^{(i+1)}$ as the zero of the tangent (i.e., the x-coordinate of the point at which the tangent cuts the x-axis), see Figure 3 for an illustration.

It is convenient for our purposes to formulate Newton's method in terms of the *differential* of g at a given point $v \in \mathbb{R}$. Recall that the differential of g is the mapping $Dg|_v : \mathbb{R} \to \mathbb{R}$ that assigns to each $v \in \mathbb{R}$ the linear function describing the tangent of g at the point $(v, g(v))$ in the coordinate system having $(v, g(v))$ as origin. If we denote the differential of g at v by $Dg|_v$, then we have $Dg|_v(X) = g'(v) \cdot X$ (for example, if $g(X) = X^2 + 3X + 1$, then $Dg|_3(X) = 9X$). In terms of differentials, Newton's method is formulated as follows. Starting at some $\nu^{(0)}$, compute iteratively $\nu^{(i+1)} = \nu^{(i)} + \Delta^{(i)}$, where $\Delta^{(i)}$ is the solution of the linear equation $Dg|_{\nu^{(i)}}(X) + g(\nu^{(i)}) = 0$ (assume for simplicity that the solution of the linear system is unique). In particular, for a univariate function g on the real numbers, we obtain for $\Delta^{(i)}$

$$0 = Dg|_{\nu^{(i)}}(\Delta^{(i)}) + g(\nu^{(i)}) = g'(\nu^{(i)}) \cdot \Delta^{(i)} + g(\nu^{(i)}), \text{ that is, } \Delta^{(i)}, = -\frac{g(\nu^{(i)})}{g'(\nu^{(i)})}$$

and, thus, the standard formulation of Newton's method:

$$\nu^{(i+1)} = \nu^{(i)} + \Delta^{(i)} = \nu^{(i)} - \frac{g(\nu^{(i)})}{g'(\nu^{(i)})}.$$

Computing the solution of a fixed-point equation, $f(X) = X$ amounts to computing a zero of $g(X) = f(X) - X$, and so we can apply Newton's method. Since for every real number v, we have $Dg|_v(X) = Df|_v(X) - X$, the method looks as follows:

Starting at some $\nu^{(0)}$, compute iteratively

$$\nu^{(i+1)} = \nu^{(i)} + \Delta^{(i)} \tag{2}$$

where $\Delta^{(i)}$ is the solution of the linear equation

$$Df|_{\nu^{(i)}}(X) + f(\nu^{(i)}) - \nu^{(i)} = X. \tag{3}$$

So Newton's method "breaks down" the problem of finding a solution to a non-linear system $f(X) = X$ into finding solutions to the sequence (3) of linear systems.

3.1.1. *Generalization.* Generalizing Newton's method to arbitrary ω-continuous semirings requires us to overcome two obstacles. First, the notion of differential seems to require a richer algebraic structure than a semiring: differentials are usually defined in terms of derivatives, which are the limit of a quotient of differences, which requires both the sum and product operations to have inverses. Second, Eq. (3) contains the term $f(\nu^{(i)}) - \nu^{(i)}$, which again seems to be defined only if summation has an inverse.

3.1.1.1 THE FIRST OBSTACLE. Differentiable functions satisfy well known algebraic rules with respect to sums and products of functions. We take these rules as the definition of the differential of a power series f over an ω-continuous semiring $\mathcal{S}$. We remark that this definition of differential generalizes the usual algebraic definition of derivatives.

Definition 3.1. Let f be a power series in one variable X over an ω-continuous semiring $\mathcal{S}$. The *differential of* f at the point v is the mapping $Df|_v : S \rightarrow S$ inductively defined as follows for every $b \in S$:

$$Df|_v(b) = \begin{cases} 0 & \text{if } f \in S \\ b & \text{if } f = X \\ Dg|_v(b) \cdot h(v) + g(v) \cdot Dh|_v(b) & \text{if } f = g \cdot h \\ \sum_{i \in I} Df_i|_v(b) & \text{if } f = \sum_{i \in I} f_i(b). \end{cases}$$

Example 3.2. First, consider a polynomial f over some *commutative* ω-continuous semiring. Because of commutative multiplication, we may write any monomial as $a \cdot X^k$ for some $k \in \mathbb{N}$ and $a \in S$, and so $f = \sum_{k=0}^{n} a_k \cdot X^k$ for suitable $n \in \mathbb{N}$ and $a_k \in S$. Let f' denote the usual algebraic derivative of f with respect to X, that is, $f' = \sum_{k=1}^{n} k \cdot a_k \cdot X^{k-1}$ where $k \cdot a_k$ is an abbreviation of $\sum_{i=1}^{k} a_k$. We then have

$$\begin{aligned}
Df|_v(b) &= \textstyle\sum_{k=0}^{n} D(a_k \cdot X^k)|_v(b) \\
&= \textstyle\sum_{k=0}^{n}(Da_k|_v(b) \cdot (X^k)(v) + \sum_{j=0}^{k-1} a_k \cdot (X^j)(v) \cdot DX|_v(b) \cdot (X^{k-1-j})(v)) \\
&= \textstyle\sum_{k=0}^{n} \sum_{j=0}^{k-1} a_k \cdot v^j \cdot DX|_v(b) \cdot v^{k-1-j} \\
&= (\textstyle\sum_{k=1}^{n} k \cdot a_k \cdot v^{k-1}) \cdot b \\
&= f'(v) \cdot b.
\end{aligned}$$

So, on commutative semirings, we have $Df|_v(b) = f'(v) \cdot b$ for all $v, b \in S$.

Now, assume that multiplication is not commutative, and consider the simple case of a quadratic monomial $m = a_0 X a_1 X a_2$. We then have

$$\begin{aligned}
Dm|_v(b) &= a_0 \cdot DX|_v(b) \cdot a_1 \cdot v \cdot a_2 + a_0 \cdot v \cdot a_1 \cdot DX|_v(b) \cdot a_2 \\
&= a_0 \cdot b \cdot a_1 \cdot v \cdot a_2 + a_0 \cdot v \cdot a_1 \cdot b \cdot a_2.
\end{aligned}$$

The important point here is that the differential "remembers" the position of the variables, and therefore does not simply append the value b.

Remark 3.3. Let Σ be a finite alphabet, $L \subseteq \Sigma^*$ a language and $u \in \Sigma^*$ a finite word. In Brzozowski [1964], the derivative $D_u L$ of L with respect to u is defined to be the language $\{w \mid uw \in L\}$. One may relate this notion of derivative to our definition of differential for the special case of univariate power series on idempotent and commutative semirings. For instance, writing the power series $f(X) = a + Xb + XXc$ as the language $L_f := \{a, Xb, XXc\}$ (with $a, b, c, X \in \Sigma$), its derivative with respect to X is $D_X L_f = \{b, Xc\}$. Writing this language as power series $g(X) = b + Xc$, we see that $g(X)$ is related to the differential Df by $Df|_v(e) = be + vce = g(v) \cdot e$ in this case. If multiplication is *not* commutative, then $Df|_v(e) = eb + evc + vec$, so the equality $Df|_v(e) = g(v) \cdot e$ no longer holds.

3.1.1.2 THE SECOND OBSTACLE. Profiting from the fact that 0 is the unique minimal element of $\mathcal{S}$ with respect to $\sqsubseteq$, we fix $\nu^{(0)} = f(0)$, which guarantees $\nu^{(0)} \sqsubseteq f(\nu^{(0)})$. We *guess* that with this choice $\nu^{(i)} \sqsubseteq f(\nu^{(i)})$ will hold not only for $i = 0$, but for every $i \geq 0$ (the correctness of this guess is proved in Theorem 3.9). If the guess is correct, then, by the definition of $\sqsubseteq$, the semiring contains an element $\delta^{(i)}$ such that $f(\nu^{(i)}) = \nu^{(i)} + \delta^{(i)}$. We replace $f(\nu^{(i)}) - \nu^{(i)}$ by any such $\delta^{(i)}$. This leads to the following definition:

Definition 3.4. Let f be a power series in one variable. A *Newton sequence* $(\nu^{(i)})_{i \in \mathbb{N}}$ is given by:

$$\nu^{(0)} = f(0) \quad \text{and} \quad \nu^{(i+1)} = \nu^{(i)} + \Delta^{(i)} \tag{4}$$

where $\Delta^{(i)}$ is the least solution of

$$Df|_{\nu^{(i)}}(X) + \delta^{(i)} = X \tag{5}$$

and $\delta^{(i)}$ is any element satisfying $f(\nu^{(i)}) = \nu^{(i)} + \delta^{(i)}$.

Theorem 3.9 below shows that Newton sequences always exist (i.e., there is always at least one possible choice for $\delta^{(i)}$), and that they all converge at least as fast as the Kleene sequence. More precisely, we show that for every $i \geq 0$

$$\kappa^{(i)} \sqsubseteq \nu^{(i)} \sqsubseteq \nu^{(i+1)} \sqsubseteq \mu f \; .$$

Since we have $\mu f = \sup_{i \in \mathbb{N}} \kappa^{(i)}$ by Kleene's theorem, Newton sequences converge to μf.

In general, there can be more than one choice for $\delta^{(i)}$. But Theorem 3.9 also shows that the Newton sequence $(\nu^{(i)})_{i \geq 0}$ itself is uniquely determined by $\boldsymbol{f}$ (and $\mathcal{S}$). In other words, the choice of $\delta^{(i)}$ does not influence the Newton approximants $\nu^{(i)}$.

Let us consider some examples for Newton sequences.

3.1.2. *Examples.* We compute the Newton sequence for a program that can execute a and terminate, or execute b and then call itself twice, recursively (the abstract scheme of a divide-and-conquer procedure). The abstract equation of the program is

$$X = a + b \cdot X \cdot X \tag{6}$$

3.1.2.1. THE REAL SEMIRING. Consider the case $a = b = 1/2$ (we can interpret a and b as probabilities). We have $Df|_v(X) = v \cdot X$, and one single possible choice for $\delta^{(i)}$, namely $\delta^{(i)} = f(\nu^{(i)}) - \nu^{(i)} = 1/2 + 1/2\,(\nu^{(i)})^2 - \nu^{(i)}$. Equation (5) becomes

$$\nu^{(i)}\, X + 1/2 + 1/2\,(\nu^{(i)})^2 - \nu^{(i)} = X$$

with $\Delta^{(i)} = (1 - \nu^{(i)})/2$ as unique solution. We get

$$\nu^{(0)} = 1/2 \qquad \nu^{(i+1)} = (1 + \nu^{(i)})/2$$

and therefore $\nu^{(i)} = 1 - 2^{(i+1)}$. So the Newton sequence converges to 1, and gains one bit of accuracy per iteration.

3.1.2.2 THE LANGUAGE SEMIRING. Consider the language semiring with $\Sigma = \{a, b\}$. The product operation is concatenation of languages, and hence non-commutative. So we have $Df|_v(X) = bvX + bXv$. We show in Proposition 7.1 that when sum is idempotent (as in this case, where it is union of languages) the definition of the Newton sequence can be simplified to

$$\nu^{(0)} = f(0) \quad \text{and} \quad \nu^{(i+1)} = \Delta^{(i)}, \tag{7}$$

where $\Delta^{(i)}$ is the least solution of

$$Df|_{\nu^{(i)}}(X) + f(\nu^{(i)}) = X. \tag{8}$$

With $f = a + b \cdot X \cdot X$ from Eq. (6), Eq. (8) becomes

$$\underbrace{b\nu^{(i)}X + bX\nu^{(i)}}_{Df|_{\nu^{(i)}}(X)} + \underbrace{a + b\nu^{(i)}\nu^{(i)}}_{f(\nu^{(i)})} = X. \tag{9}$$

Its least solution (which by (7) is equal to the $(i + 1)$-st Newton approximant) is a context-free language. Let $G^{(i)}$ be a grammar with axiom $S^{(i)}$ such that $\nu^{(i)} = L(G^{(i)})$. Since $\nu^{(0)} = f(0)$, the grammar $G^{(0)}$ contains one single production, namely $S^{(0)} \rightarrow a$. Equation (9) allows us to define $G^{(i+1)}$ in terms of $G^{(i)}$, and we get:

$$\begin{aligned} G^{(0)} &= \{S^{(0)} \rightarrow a\} \\ G^{(i+1)} &= G^{(i)} \cup \{S^{(i+1)} \rightarrow a \mid bS^{(i+1)}S^{(i)} \mid bS^{(i)}S^{(i+1)} \mid bS^{(i)}S^{(i)}\} \end{aligned}$$

3.1.2.3. The Counting Semiring. Consider the counting semiring with $r_a = \{(1,0)\}$ and $r_b = \{(0,1)\}$. Since the sum operation is union of sets of vectors, it is idempotent and Eqs. (7) and (8) hold. Since the product operation is now commutative, we obtain for our example

$$b \cdot \nu^{(i)} \cdot X + a + b \cdot \nu^{(i)} \cdot \nu^{(i)} = X \tag{10}$$

Using Kleene's fixed-point theorem (Proposition 2.4), it is easy to see that the least solution of a linear equation $X = u \cdot X + v$ over a commutative ω-continuous semiring is $u^* \cdot v$, where $u^* = \sum_{i \in \mathbb{N}} u^i$. The least solution $\Delta^{(i)}$ of Eq. (10) is then given by

$$\Delta^{(i)} = (r_b \cdot \nu^{(i)})^* \cdot (r_a + r_b \cdot \nu^{(i)} \cdot \nu^{(i)})$$

and we obtain:

$$\begin{aligned}
\nu^{(0)} &= r_a = \{(1,0)\} \\
\nu^{(1)} &= (r_b \cdot r_a)^* \cdot (r_a + r_b \cdot r_a \cdot r_a) = \{(n,n) \mid n \geq 0\} \cdot \{(1,0),(2,1)\} \\
&= \{(n+1,n) \mid n \geq 0\} \\
\nu^{(2)} &= (\{(n,n) \mid n \geq 1\})^* \cdot (\{(1,0)\} \cup \{(2n+2, 2n+1) \mid n \geq 0\}) \\
&= \{(n+1,n) \mid n \geq 0\}
\end{aligned}$$

So the Newton sequence reaches a fixed point after one iteration. In Section 7, we show that the Newton sequence of a system of n equations over *any commutative* and *idempotent* semiring converges after at most n iterations. Further note that the counting semiring does not satisfy the ascending-chain property, that is, there are monotonically increasing sequences in the counting semiring that do not become stationary. Therefore, the Kleene sequence and its variations do not reach $\mu \boldsymbol{f}$ after a finite number of steps in general.

3.2. The Multivariate Case. Newton's method can be easily generalized to the multivariate case. Given differentiable functions $g_1, \ldots, g_n : \mathbb{R}^n \to \mathbb{R}$, the method computes a solution of $\boldsymbol{g}(\boldsymbol{X}) = \boldsymbol{0}$, where $\boldsymbol{g} = (g_1, \ldots, g_n)$; starting at some $\boldsymbol{\nu}^{(0)}$, the method computes $\boldsymbol{\nu}^{(i+1)} = \boldsymbol{\nu}^{(i)} + \boldsymbol{\Delta}^{(i)}$, where $\boldsymbol{\Delta}^{(i)}$ is the solution of the *system* of linear equations

$$\begin{aligned}
Dg_1|_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{X}) + g_1(\boldsymbol{\nu}^{(i)}) &= 0 \\
&\vdots \\
Dg_n|_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{X}) + g_n(\boldsymbol{\nu}^{(i)}) &= 0
\end{aligned}$$

and $Dg_j|_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{X})$ is the differential of g_j at $\boldsymbol{\nu}^{(i)}$, that is, the function corresponding to the tangent hyperplane of g_j at the point $(\boldsymbol{\nu}^{(i)}, g_j(\boldsymbol{\nu}^{(i)}))$.

Given a function $g : \mathbb{R}^n \to \mathbb{R}$ differentiable at a point $\boldsymbol{\nu}$, there exists a function $D_X g|_\nu$ for each variable $X \in \mathcal{X}$ such that $Dg|_{\boldsymbol{\nu}} = \sum_{X \in \mathcal{X}} D_X g|_{\boldsymbol{\nu}}$. These functions are closely related to the partial derivatives, more precisely we have $D_X g|_{\boldsymbol{\nu}}(\boldsymbol{X}) = \left.\frac{\partial g}{\partial X}\right|_{\boldsymbol{\nu}} \cdot X$.

We denote the system above by $\boldsymbol{Dg}|_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{X}) + \boldsymbol{g}(\boldsymbol{\nu}^{(i)}) = \boldsymbol{0}$. For the problem of computing a solution of a system of fixed-point equations, the method looks as follows:

Starting at some $\boldsymbol{v}^{(0)}$, compute iteratively

$$\boldsymbol{v}^{(i+1)} = \boldsymbol{v}^{(i)} + \boldsymbol{\Delta}^{(i)} \tag{11}$$

where $\boldsymbol{\Delta}^{(i)}$ is the least solution of the linear system of fixed-point equations

$$D\boldsymbol{f}|_{\boldsymbol{v}^{(i)}}(\boldsymbol{X}) + \boldsymbol{f}(\boldsymbol{v}^{(i)}) - \boldsymbol{v}^{(i)} = \boldsymbol{X}. \tag{12}$$

3.2.1. *Generalization.* Again, we use the algebraic definition of differential:

Definition 3.5. Let f be a power series over an ω-continuous semiring $\mathcal{S}$ and let $X \in \mathcal{X}$ be a variable. The *differential of f with respect to X* at the point $\boldsymbol{v}$ is the mapping $D_X f|_{\boldsymbol{v}} : V \to S$ inductively defined as follows:

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

Further, we define the *differential of f* at $\boldsymbol{v}$ as the function

$$Df|_{\boldsymbol{v}} := \sum_{X \in \mathcal{X}} D_X f|_{\boldsymbol{v}}.$$

Finally, the differential of a vector of power series $\boldsymbol{f}$ at $\boldsymbol{v}$ is defined as the function $D\boldsymbol{f}|_{\boldsymbol{v}} : V \to V$ with

$$(D\boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}))_X := D\boldsymbol{f}_X|_{\boldsymbol{v}}(\boldsymbol{b}).$$

As in the univariate case, we guess that $\boldsymbol{v}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{v}^{(i)})$ will hold for every $i \geq 0$. If the guess is correct, then the semiring contains an element $\boldsymbol{\delta}^{(i)}$ such that $\boldsymbol{f}(\boldsymbol{v}^{(i)}) = \boldsymbol{v}^{(i)} + \boldsymbol{\delta}^{(i)}$, and Eq. (12) becomes

$$D\boldsymbol{f}|_{\boldsymbol{v}^{(i)}}(\boldsymbol{X}) + \boldsymbol{\delta}^{(i)} = \boldsymbol{X}. \tag{13}$$

This leads to the following definition:

Definition 3.6. Let $\boldsymbol{f} : V \to V$ be a vector of power series.

- Let $i \in \mathbb{N}$. An ith *Newton approximant* $\boldsymbol{v}^{(i)}$ is inductively defined by

  $$\boldsymbol{v}^{(0)} = \boldsymbol{f}(\boldsymbol{0}) \quad \text{and} \quad \boldsymbol{v}^{(i+1)} = \boldsymbol{v}^{(i)} + \boldsymbol{\Delta}^{(i)},$$

  where $\boldsymbol{\Delta}^{(i)}$ is the least solution of Eq. (13) and $\boldsymbol{\delta}^{(i)}$ is any vector satisfying $\boldsymbol{f}(\boldsymbol{v}^{(i)}) = \boldsymbol{v}^{(i)} + \boldsymbol{\delta}^{(i)}$.
- A sequence $(\boldsymbol{v}^{(i)})_{i \in \mathbb{N}}$ of Newton approximants is called *Newton sequence*.

Remark 3.7. One can easily show by induction that for any $\boldsymbol{v}, \boldsymbol{b}, \boldsymbol{b}' \in V$, and any vector of power series $\boldsymbol{f}$ we have

$$D\boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b} + \boldsymbol{b}') = D\boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}) + D\boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}').$$

Remark 3.8. If the product operation of the semiring is commutative, the differential $D_X f|_{\boldsymbol{v}}(\boldsymbol{a})$ can be written as $\frac{\partial f}{\partial X}|_{\boldsymbol{v}} \cdot \boldsymbol{a}_X$, where $\frac{\partial f}{\partial X}|_{\boldsymbol{v}}$ denotes the usual partial derivative of the power series f with respect to X, taken at $\boldsymbol{v}$, as known from algebra:

$$\left.\frac{\partial f}{\partial X}\right|_{\boldsymbol{v}} = \begin{cases} 0 & \text{if } f \in S \text{ or } f \in \mathcal{X} \setminus \{X\} \\ 1 & \text{if } f = X \\ \left.\frac{\partial g}{\partial x}\right|_{\boldsymbol{v}} \cdot h(\boldsymbol{v}) + g(\boldsymbol{v}) \cdot \left.\frac{\partial h}{\partial X}\right|_{\boldsymbol{v}} & \text{if } f = g \cdot h \\ \sum_{i \in I} \left.\frac{\partial f_i}{\partial X}\right|_{\boldsymbol{v}} & \text{if } f = \sum_{i \in I} f_i. \end{cases}$$

So, in commutative semirings we may use the usual representation of the differential by means of the gradient of a power series f, or more generally, by the Jacobian of a vector $\boldsymbol{f}$ of power series.

The following fundamental theorem shows that there exists exactly one Newton sequence, that it converges to the least fixed point, and that it does so at least as fast as the Kleene sequence.

THEOREM 3.9. *Let $\boldsymbol{f} \colon V \to V$ be a vector of power series.*

- *There is exactly one Newton sequence $(\boldsymbol{v}^{(i)})_{i \in \mathbb{N}}$.*
- *The Newton sequence is monotonically increasing, converges to the least fixed point and does so at least as fast as the Kleene sequence. More precisely, it satisfies*

$$\boldsymbol{\kappa}^{(i)} \sqsubseteq \boldsymbol{v}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{v}^{(i)}) \sqsubseteq \boldsymbol{v}^{(i+1)} \sqsubseteq \mu \boldsymbol{f} = \sup_{j \in \mathbb{N}} \boldsymbol{\kappa}^{(j)} \textit{ for all } i \in \mathbb{N}.$$

Before giving the formal proof of Theorem 3.9 (see Section 5), we present two examples of *Newtonian program analysis*, which illustrate the use of our generalized Newton's method to program analysis.

4. *Two Case Studies*

We apply our results to the analysis of two small programs. In the first one, a may-alias analysis where we use the counting semiring, Kleene iteration does not terminate, while Newton's method terminates in one step. In the second case, an average runtime analysis where we use the real semiring, neither technique terminates, but Newton's method converges substantially faster to the solution.

4.1. A MAY-ALIAS ANALYSIS. We conduct a may-alias analysis in the spirit of Deutsch [1994]. We consider a program listify() that transforms a binary tree (all nonleaf nodes have two children) of pointers into a list of pointers by reading the nodes of the tree in preorder. An implementation in C++ could look as shown in Figure 4, where move_right() follows the right child pointer, and similarly for move_left() and move_up().

The flowgraphs of listify(), listifyL(), and listifyR() are shown in Figure 5.

We wish to compute may-alias information, that is, information on which *data access paths* of the tree and the list may point to the same element. A data access path of the tree can be represented as a word over the alphabet $\{l, r\}$: for instance, the path llr corresponds to the element found as follows: start at the root node, follow twice the pointer to the left child, then once the pointer to the right child, and then the pointer to the data. Similarly, a data access path of the list can be represented as a word over $\{s\}$ (for *successor*). So may-alias information can be represented as a set of pairs (w_1, w_2), where $w_1 \in \{l, r\}^*$ and $w_2 \in \{s\}^*$.

```
class Tree{
  struct Node {
    void *data;
    Node *left, *right,
         *parent;
     ...
  };

  Node *root;
public:
  Tree() { ... }
 ~Tree() { ... }
  ...
  void move_left()  { ... }
  ...
  bool is_leaf() { ... }
};


class Listify {
  Tree* T;
  list<void*> L;
```

```
  void listifyL() {
    T.move_left();
    listify();
    T.move_up();
  }

  void listifyR() {
    T.move_right();
    listify();
    T.move_up();
  }

  void listify() {
    L.push_back(T->get_data());

    if( T.is_leaf() == false ) {
      listifyL(); listifyR();
    }
  }
public:
  Listify() : T(0), L() {}
  void make_it_so( Tree& t ) {
    T = &t; T->go_top(); listify();
  }
};
```

FIG. 4. Code snippet of the class Listify that serializes a tree into a list.

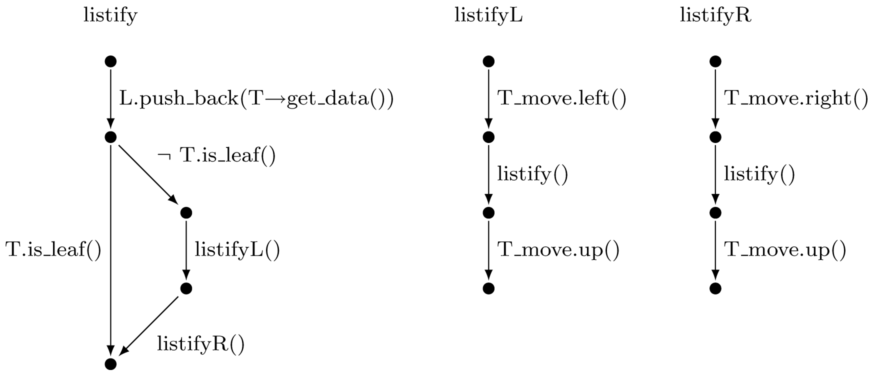


FIG. 5. Flowgraph for listify(), listifyL(), and listifyR().

We are interested in may-alias information at the *entry* point of listify(), directly before the execution of L.push_back(T→get.data()), which creates an alias. More exactly, we wish to overapproximate the alias pairs generated by any valid program path leading from the entry point of listify to itself, i.e., the "join-over-all-paths" (or JOP) solution of the program.

Recall from Section 1.1 how we compute the JOP-values of a procedural program: We first use Newton's method to compute or overapproximate, for any procedure P, the effect of P, denoted by $\mathrm{JOP}_0(P)$. Then, the label of an edge calling P is replaced by $\mathrm{JOP}_0(P)$, and additional edges (labelled with the 1-element) from

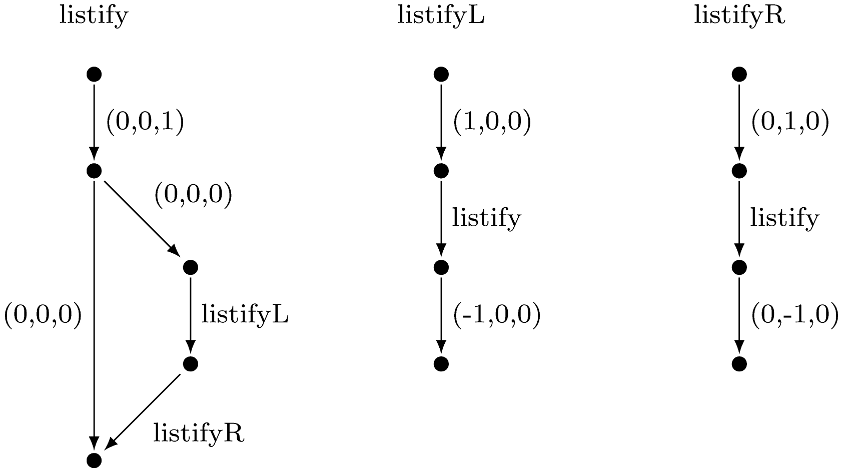


FIG. 6. Abstract flowgraphs for listify(), listifyL(), and listifyR().

the source of the call to the entry point of P are inserted. The resulting flowgraph no longer contains procedure calls. For any program point p, we obtain JOP(p) by solving the system of linear dataflow equations derived from that flowgraph. We apply this approach to the listify() program.

In order to guarantee that the computation of JOP_0 terminates, we use the *Parikh abstraction*, in which we abstract a word $w \in \{l, r\}^*$ by a vector $(\#_l w, \#_r w)$, where $\#_l w$ and $\#_r w$ denote the number of l's and r's in w. The result of the analysis will be a set of triples $(n_l, n_r, n_s) \in \mathbb{N}^3$. A triple (n_l, n_r, n_s) indicates that there may be an alias between some data access path containing n_l times the letter l and n_r times the letter r, and the (unique) data access path containing n_s times the letter s (the s-th element of the list).

We can then work over the counting semiring described in Section 2.3.3, with $2^{\mathbb{N}^3}$ as carrier. Recall that the sum operation is set union, and the product operation, denoted by $\cdot_c$, is given by

$$N \cdot_c M = \{(n_l + m_l, n_r + m_r, n_s + m_s) \mid (n_l, n_r, n_s) \in N, (m_l, m_r, m_s) \in M\} .$$

In our abstraction, T.move_left() adds 1 to the number of l's in the data access path of the tree, leaving the number of r's and s's untouched. So we replace the edge label "T.move_left()" with the one-element set $\{(1, 0, 0)\}$. Proceeding similarly with the rest of the edges, we obtain the abstract flowgraphs of Figure 6 (we omit the curly brackets of one-element sets).

From the abstract flowgraphs we get the equations (with li, li_R and li_L as abbreviations of listify(), listifyL() and listifyR()):

$$\begin{aligned} li &= \{(0,0,1)\} \cdot_c \left(\{(0,0,0)\} \cup \{(0,0,0)\} \cdot_c li_L \cdot_c li_R\right) \\ li_L &= \{(1,0,0)\} \cdot_c li \cdot_c \{(-1,0,0)\} \\ li_R &= \{(0,1,0)\} \cdot_c li \cdot_c \{(0,-1,0)\} \end{aligned}$$

which can be simplified applying the commutativity of $\cdot_c$, yielding $li_L = li$ and $li_R = li$. So, we only have to solve the univariate quadratic equation

$$li = \{(0,0,1)\} \cup \{(0,0,1)\} \cdot_c li \cdot_c li. \tag{14}$$

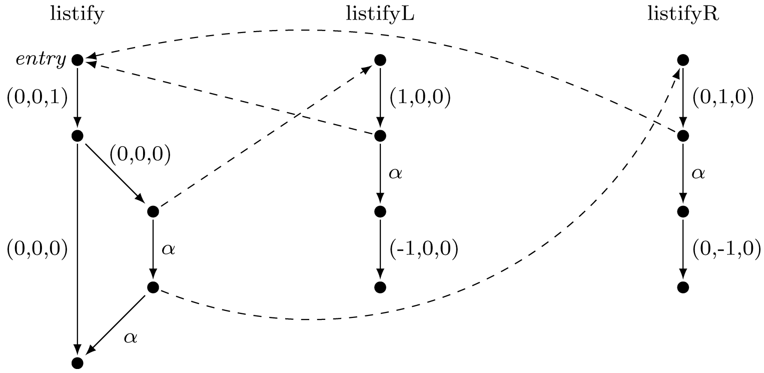


FIG. 7. Abstract flowgraphs for listify(), listifyL(), and listifyR().

Kleene iteration does not terminate for (14): we obtain $\boldsymbol{\kappa}^{(i)} = \{(0, 0, 2j + 1) \mid 0 \leq j \leq i\}$, never reaching the least solution. But, since our semiring is idempotent and commutative, Theorem 7.7 (see Section 7.1) guarantees that Newton's method terminates in one step. It follows that $\boldsymbol{\nu}^{(1)} = \{(0, 0, 2j + 1) \mid 0 \leq j\}$ is the least solution of (14). This is our desired overapproximation of JOP_0. The interpretation is simple: after termination of listify(), an arbitrary *odd* number of items may have been added to the list, but it is not possible to have added an even number of items.

As described before, we can use JOP_0 to construct a flowgraph without procedure calls, see Figure 7, where $\alpha = \{(0, 0, 2j + 1) \mid j \in \mathbb{N}\}$ and dashed lines indicate edges labelled with (0, 0, 0).Since we are interested in the value of the JOP for the entry point, we get the linear equation

$$\textit{entry} = \{(0, 0, 0)\} \;\cup\; \{(1, 0, 1)\} \cdot_c \textit{entry} \;\cup\; \{(0, 1, 1)\} \cdot_c \alpha \cdot_c \textit{entry}$$

where the second and third terms on the right-hand-side correspond to the loops involving listifyL() and listifyR(). The least solution is

$$\textit{entry} = \big(\{(1, 0, 1)\} \cup \{(0, 1, 1)\} \cdot_c \alpha\big)^{*_c}$$

which corresponds to the set

$$\{(n_l, n_r, n_s) \in \mathbb{N}^3 \mid (n_r = 0 \wedge n_s = n_l) \vee (n_r > 0 \wedge \exists k \in \mathbb{N} : n_s = 2n_r + n_l + 2k)\}.$$

This result gives the following information on may-aliases:

—A data access path of the tree containing no r and n_l times l can only be aliased to the n_lth element of the list.

—A data access path of the tree with $n_r > 0$ times r and n_l times l can only be aliased to the $2n_r + n_l$th element of the list, or to the larger elements of the same parity.

The problem that Kleene iteration does not terminate for these recursive examples has been addressed by many researchers. The *k-limiting* technique was introduced as a way to palliate the problem: basically, it computes the aliases exactly for data access paths of length at most k, and abstracts the rest very crudely. The Parikh abstraction can provide information on data access paths of arbitrary depth. It was used (together with some other features) in Deutsch [1994]. Notice, however, that

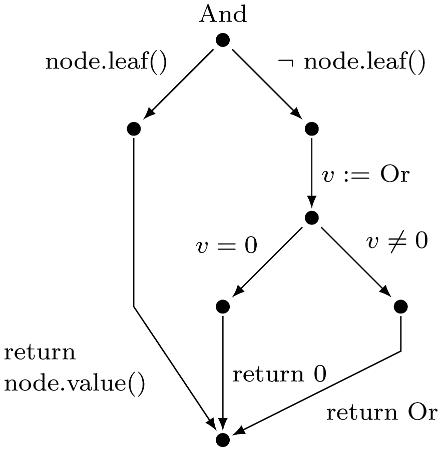


FIG. 8. Flowgraph for And().

in our case we derive termination for this abstraction from a generic argument, namely from Theorem 7.7.

4.2. AN AVERAGE RUNTIME ANALYSIS. In this example, we show how by just changing the semiring our approach can also be applied to average runtime analysis. We consider a program for lazy evaluation of And/Or-trees. For this example, an And/Or-tree is a tree where (i) every node has either zero or two children, (ii) every inner node of the tree is either an And-node or an Or-node, and (iii) on any path from the root to a leaf And- and Or-nodes alternate.

The program constructs and evaluates nodes of the tree (to 0 or 1) only if needed. For instance, if the left subtree of an And-node evaluates to 0, then the program neither constructs nor evaluates the right subtree. More specifically, we assume the existence of functions node.leaf(), node.value(), node.left() and node.right(), where node.leaf() checks if a node is a leaf, node.value() evaluates a leaf node, and node.left() and node.right() create the left and the right child of a node which is not a leaf. Notice that because of lazy evaluation the program may terminate even if the input is an infinite tree.

```
function And(node)
  if node.leaf() then
    return node.value()
  else
    $v$ := Or(node.left())
    if $v = 0$ then
      return 0
    else
      return Or(node.right())
```

```
function Or(node)
  if node.leaf() then
    return node.value()
  else
    $v$ := And(node.left())
    if $v = 1$ then
      return 1
    else
      return And(node.right())
```

Figure 8 shows the flowgraph of And(), the one of Or() is similar. We assume that the root of the tree is always an And-node, i.e., the main procedure is And().

Assume the probabilities that node.leaf() and node.value() return 0 or 1 are known, as well as the time taken by each instruction. For our example, we assume

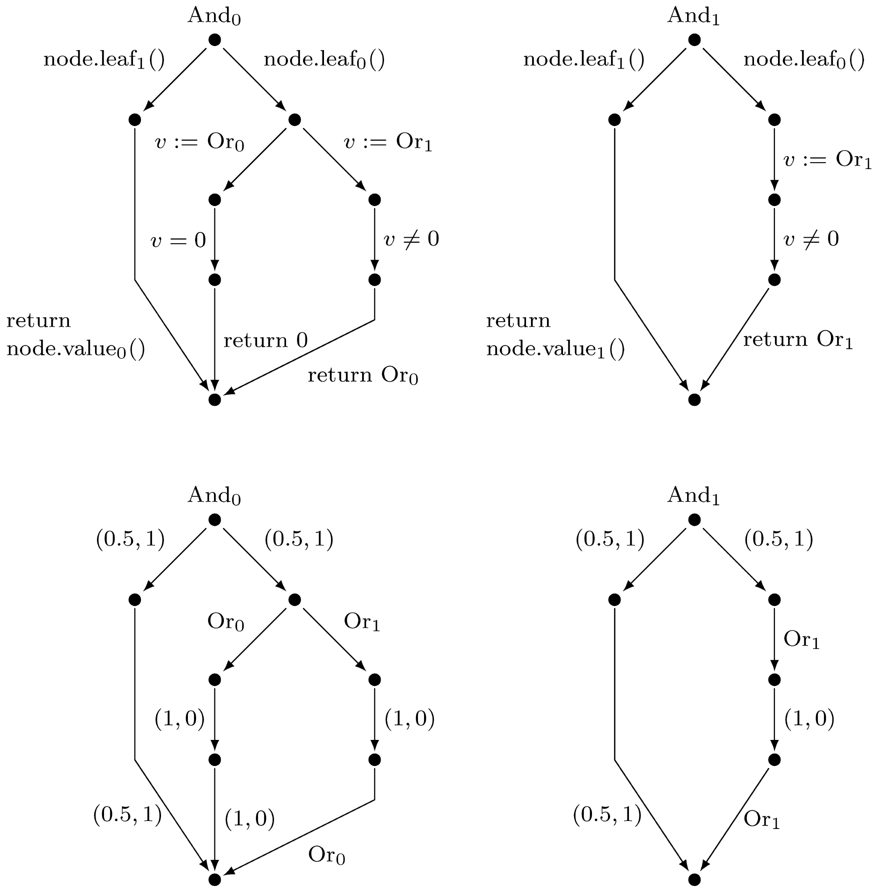


FIG. 9. Flowgraphs for the procedures And$_0$ and And$_1$.

that all probabilities are equal to 0.5, that node.leaf() and node.value() take one time unit, and all other instructions take no time. We perform an analysis to compute (a) the probability that the evaluation terminates (with results 0 or 1), and (b) the average runtime. This corresponds to taking the semiring for the second probabilistic interpretation in Section 2.3.4.

The functions And() and Or() return values, and their control flow depends on the values returned by calls to node.leaf(), node.value(), and recursive calls to Or() and And(). We need an analysis that captures these dependencies. For this, we use a standard instrumentation: we interpret a program procedure, say P, that may return k different values, say $v_0, \ldots, v_{k-1}$, as k different procedures, $P_0, \ldots, P_{k-1}$, where P_i returns v_i; more precisely, the control flow of P_i contains the valid flow paths of P that finish with **return** v_i. In our example, we get four procedures: And$_0$, And$_1$, Or$_0$ and Or$_1$. The flowgraphs of And$_0$ and And$_1$ are shown in the first row Figure 9. Notice, for instance, that these flowgraphs exclude paths where a call to Or$_1$ (i.e., a call to Or() that returns 1) is followed by the **then** branch of "**if** $v = 0$ **then return** 0". The flowgraphs of Or$_0$ and Or$_1$ are similar. By construction, the probabilities of termination of And$_0$() and And$_1$() are equal to the probabilities that And() terminates with value 0 and with value 1.

Recall that the semiring values of Section 2.3.4 are of the form (p, d), where $p \in [0, 1]$ stands for the probability of a given set of paths and $d \in [0, \infty)$ for their expected execution time (duration). The second row of Figure 9 shows how to assign semiring values to the edges. For instance, the edge labelled by node.leaf$_1$() gets (0.5, 1) as semiring value, because node.leaf() returns 1 with probability 0.5, and it takes one unit of time.

Using the framework of Section 1.1, the probability that a procedure returns a value and the expected time to return this value is given as the least solution of the following equation system:

$$\begin{aligned}
\mathrm{And}_0 &= (0.25, 2) +_e (0.5, 1) \cdot_e (\mathrm{Or}_0 +_e \mathrm{Or}_1 \cdot_e \mathrm{Or}_0)\\
\mathrm{And}_1 &= (0.25, 2) +_e (0.5, 1) \cdot_e \mathrm{Or}_1 \cdot_e \mathrm{Or}_1\\
\mathrm{Or}_0 &= (0.25, 2) +_e (0.5, 1) \cdot_e \mathrm{And}_0 \cdot_e \mathrm{And}_0\\
\mathrm{Or}_1 &= (0.25, 2) +_e (0.5, 1) \cdot_e (\mathrm{And}_1 +_e \mathrm{And}_0 \cdot_e \mathrm{And}_1)
\end{aligned} \tag{15}$$

where $+_e$ and $\cdot_e$ are the semiring operations defined in Section 2.3.4.

The equation system (15) happens to be solvable analytically. For instance, the And$_0$-component of the least solution is $(\frac{\sqrt{10}}{2} - 1, \frac{19}{6} + \frac{37\sqrt{10}}{30}) \approx (0.581, 7.067)$. This means that the procedure And() terminates and returns the value 1 with probability 0.581 and needs in average 7.067 time steps to do so. For equation systems stemming from larger programs, the solution may not be representable by roots, cf. Etessami and Yannakakis [2009]. Therefore, approximation methods are generally needed. We compute the first elements of the Kleene and Newton sequences for (15). Rounding to three decimals, we obtain:

i	$\boldsymbol{\kappa}^{(i)}_{\mathrm{And}_0}$	$\boldsymbol{\nu}^{(i)}_{\mathrm{And}_0}$	$\boldsymbol{\kappa}^{(i)}_{\mathrm{And}_1}$	$\boldsymbol{\nu}^{(i)}_{\mathrm{And}_1}$
0	(0.250, 2.000)	(0.250, 2.000)	(0.250, 2.000)	(0.250, 2.000)
1	(0.406, 2.538)	(0.495, 3.588)	(0.281, 2.333)	(0.342, 3.383)
2	(0.448, 2.913)	(0.568, 5.784)	(0.333, 3.012)	(0.409, 5.906)
3	(0.491, 3.429)	(0.581, 6.975)	(0.350, 3.381)	(0.419, 7.194)
4	(0.511, 3.793)	(0.581, 7.067)	(0.370, 3.904)	(0.419, 7.295)

We have $\boldsymbol{\kappa}^{(i)}_{\mathrm{Or}_0} = \boldsymbol{\kappa}^{(i)}_{\mathrm{And}_1}$ and $\boldsymbol{\nu}^{(i)}_{\mathrm{Or}_0} = \boldsymbol{\nu}^{(i)}_{\mathrm{And}_1}$ and similarly for Or$_1$. We observe that the Newton sequence converges faster than the Kleene sequence. In particular, while the first entry of $\boldsymbol{\nu}^{(4)}_{\mathrm{And}_0}$ is > 0.58, further computation shows that $i = 21$ is the smallest index i such that the first entry of $\boldsymbol{\kappa}^{(i)}_{\mathrm{And}_0}$ is > 0.58.

The performance gap between Kleene and Newton iteration can be widened by lowering the leaf probability from 0.5 to 0.4. In this case, the procedure And() takes, in average, a time of about 29.81 to return the value 0; in other words, in this case, the second entry of the And$_0$-component of the least solution of (15) is approximately 29.81. It takes around 222 Kleene iterations to determine that this value is greater than 29.8, whereas 6 Newton iterations suffice to establish the same fact. Actually, numerical analysis shows that when the leaf probability tends to $(\sqrt{33} - 5)/2 \approx 0.372$, the average runtime tends to infinity, and the gap between Newton and Kleene iteration grows unboundedly. However, it should be mentioned that a Newton step is more expensive in general than a Kleene step, since a Newton step requires solving a linear equation system of dimension 4. In Kiefer et al. [2007] and Esparza et al. [2008, 2010] we have given a detailed analysis of the convergence speed of Newton's method applied to (numerical) fixed-point equations. In general, the more precision is required, the better is the performance of Newton's method compared to Kleene iteration.

5. *Proof of Fundamental Properties of the Newton Sequences*

In this section, we prove Theorem 3.9 that is states that there exists exactly one Newton sequence, that it converges to the least fixed point, and that it does so at least as fast as the Kleene sequence. The proof is split in two propositions. Proposition 5.6 in Section 5.1 states that there is only one Newton sequence. The following proposition covers the rest of Theorem 3.9:

PROPOSITION 5.1. *Let $\boldsymbol{f}: V \to V$ be a vector of power series.*

- *For every Newton approximant $\boldsymbol{\nu}^{(i)}$, there exists a vector $\boldsymbol{\delta}^{(i)}$ such that $\boldsymbol{f}(\boldsymbol{\nu}^{(i)}) = \boldsymbol{\nu}^{(i)} + \boldsymbol{\delta}^{(i)}$. So there is at least one Newton sequence.*
- *Any Newton sequence satisfies $\boldsymbol{\kappa}^{(i)} \sqsubseteq \boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) \sqsubseteq \boldsymbol{\nu}^{(i+1)} \sqsubseteq \mu\boldsymbol{f} = \sup_{j\in\mathbb{N}} \boldsymbol{\kappa}^{(j)}$ for all $i \in \mathbb{N}$.*

The proof of Proposition 5.1 is based on two lemmata. The first one, an easy consequence of Kleene's theorem, provides a closed form for the least solution of a linear system of fixed-point equations in terms of the Kleene star operator, defined as follows:

Definition 5.2. Let $\boldsymbol{g}: V \to V$ be a monotone map. The map $\boldsymbol{g}^*: V \to V$ is defined as $\boldsymbol{g}^*(\boldsymbol{v}) := \sum_{i\in\mathbb{N}} \boldsymbol{g}^i(\boldsymbol{v})$, where $\boldsymbol{g}^0(\boldsymbol{v}) := \boldsymbol{v}$, $\boldsymbol{g}^{i+1}(\boldsymbol{v}) := \boldsymbol{g}(\boldsymbol{g}^i(\boldsymbol{v}))$ for every $i \geq 0$. Similarly, we set for all $j \in \mathbb{N}$: $\boldsymbol{g}^{\leq j} := \sum_{0\leq i\leq j} \boldsymbol{g}^i(\boldsymbol{v})$.

The existence of $\sum_{i\in\mathbb{N}} \boldsymbol{g}^i(\boldsymbol{v})$ is guaranteed by the properties of ω-continuous semirings. Observe that $\boldsymbol{v} \sqsubseteq \boldsymbol{g}^*(\boldsymbol{v})$ and $\boldsymbol{g}^*(\boldsymbol{v}) = \boldsymbol{v} + \boldsymbol{g}(\boldsymbol{g}^*(\boldsymbol{v}))$ hold.

LEMMA 5.3. *Let $\boldsymbol{f}: V \to V$ be a vector of power series, and $\boldsymbol{u}, \boldsymbol{v} \in V$. Then the least solution of $D\boldsymbol{f}|_{\boldsymbol{u}}(\boldsymbol{X}) + \boldsymbol{v} = \boldsymbol{X}$ is $D\boldsymbol{f}|_{\boldsymbol{u}}^*(\boldsymbol{v})$. In particular, a Newton sequence from Definition* 3.6 *can be equivalently defined by setting $\boldsymbol{\nu}^{(0)} = \boldsymbol{f}(\boldsymbol{0})$ and $\boldsymbol{\nu}^{(i+1)} = \boldsymbol{\nu}^{(i)} + D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}^*(\boldsymbol{\delta}^{(i)})$.*

PROOF. Set $\boldsymbol{g}(\boldsymbol{X}) := D\boldsymbol{f}|_{\boldsymbol{u}}(\boldsymbol{X}) + \boldsymbol{v}$. The vector $\boldsymbol{g}$ is a power series in every component and thus a monotone map from V to V. By Kleene's fixed-point theorem, the least solution of $\boldsymbol{g}(\boldsymbol{X}) = \boldsymbol{X}$ is given by $\sup\{\boldsymbol{g}^i(\boldsymbol{0}) \mid i \in \mathbb{N}\} = \sup\{D\boldsymbol{f}|_{\boldsymbol{u}}^{\leq i}(\boldsymbol{v}) \mid i \in \mathbb{N}\} = D\boldsymbol{f}|_{\boldsymbol{u}}^*(\boldsymbol{v})$. □

The second lemma, which is interesting by itself, is a generalization of Taylor's theorem to arbitrary ω-continuous semirings.

LEMMA 5.4. *Let $\boldsymbol{f}: V \to V$ be a vector of power series and let $\boldsymbol{u}, \boldsymbol{v}$ be two vectors. We have*

$$\boldsymbol{f}(\boldsymbol{u}) + D\boldsymbol{f}|_{\boldsymbol{u}}(\boldsymbol{v}) \sqsubseteq \boldsymbol{f}(\boldsymbol{u} + \boldsymbol{v}) \sqsubseteq \boldsymbol{f}(\boldsymbol{u}) + D\boldsymbol{f}|_{\boldsymbol{u}+\boldsymbol{v}}(\boldsymbol{v}).$$

PROOF. It suffices to show those inequalities for each component separately, so let w.l.o.g. $\boldsymbol{f} = f: V \to S$ be a power series. We proceed by induction on the construction of f. The base case (where f is a constant) and the case where f is a sum of polynomials are easy, and so it suffices to consider the case in which f is a monomial. So let

$$f = g \cdot X \cdot a$$

for a monomial g, a variable $X \in \mathcal{X}$ and a constant a. We have

$$f(\boldsymbol{u}) = g(\boldsymbol{u}) \cdot \boldsymbol{u}_X \cdot a \quad \text{and} \quad Df|_{\boldsymbol{u}}(\boldsymbol{v}) = g(\boldsymbol{u}) \cdot \boldsymbol{v}_X \cdot a + Dg|_{\boldsymbol{u}}(\boldsymbol{v}) \cdot \boldsymbol{u}_X \cdot a.$$

By induction we obtain:

$$\begin{aligned}
f(\boldsymbol{u}+\boldsymbol{v}) &= g(\boldsymbol{u}+\boldsymbol{v}) \cdot (\boldsymbol{u}_X + \boldsymbol{v}_X) \cdot a \\
&\sqsupseteq \big(g(\boldsymbol{u}) + Dg|_{\boldsymbol{u}}(\boldsymbol{v})\big) \cdot (\boldsymbol{u}_X + \boldsymbol{v}_X) \cdot a \\
&= g(\boldsymbol{u}) \cdot \boldsymbol{u}_X \cdot a + g(\boldsymbol{u}) \cdot \boldsymbol{v}_X \cdot a + Dg|_{\boldsymbol{u}}(\boldsymbol{v}) \cdot (\boldsymbol{u}_X + \boldsymbol{v}_X) \cdot a \\
&\sqsupseteq f(\boldsymbol{u}) + g(\boldsymbol{u}) \cdot \boldsymbol{v}_X \cdot a + Dg|_{\boldsymbol{u}}(\boldsymbol{v}) \cdot \boldsymbol{u}_X \cdot a \\
&= f(\boldsymbol{u}) + Df|_{\boldsymbol{u}}(\boldsymbol{v})
\end{aligned}$$

and

$$\begin{aligned}
f(\boldsymbol{u}+\boldsymbol{v}) &= g(\boldsymbol{u}+\boldsymbol{v}) \cdot (\boldsymbol{u}_X + \boldsymbol{v}_X) \cdot a \\
&\sqsubseteq \big(g(\boldsymbol{u}) + Dg|_{\boldsymbol{u}+\boldsymbol{v}}(\boldsymbol{v})\big) \cdot (\boldsymbol{u}_X + \boldsymbol{v}_X) \cdot a \\
&= g(\boldsymbol{u}) \cdot \boldsymbol{u}_X \cdot a + g(\boldsymbol{u}) \cdot \boldsymbol{v}_X \cdot a + Dg|_{\boldsymbol{u}+\boldsymbol{v}}(\boldsymbol{v}) \cdot (\boldsymbol{u}_X + \boldsymbol{v}_X) \cdot a \\
&\sqsubseteq f(\boldsymbol{u}) + g(\boldsymbol{u}+\boldsymbol{v}) \cdot \boldsymbol{v}_X \cdot a + Dg|_{\boldsymbol{u}+\boldsymbol{v}}(\boldsymbol{v}) \cdot (\boldsymbol{u}_X + \boldsymbol{v}_X) \cdot a \\
&= f(\boldsymbol{u}) + Df|_{\boldsymbol{u}+\boldsymbol{v}}(\boldsymbol{v}) \qquad \Box
\end{aligned}$$

We can now proceed to prove Proposition 5.1.

PROOF OF PROPOSITION 5.1. First, we prove for all $i \in \mathbb{N}$ that a suitable $\boldsymbol{\delta}^{(i)}$ exists and, at the same time, that the inequality $\boldsymbol{\kappa}^{(i)} \sqsubseteq \boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)})$ holds. We proceed by induction on i. The base case $i = 0$ is easy. For the induction step, let $i \geq 0$.

$$\begin{aligned}
\boldsymbol{\kappa}^{(i+1)} &= \boldsymbol{f}(\boldsymbol{\kappa}^{(i)}) && \text{(definition of } \boldsymbol{\kappa}^{(i)}\text{)} \\
&\sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) && \text{(induction: } \boldsymbol{\kappa}^{(i)} \sqsubseteq \boldsymbol{\nu}^{(i)}\text{)} \\
&= \boldsymbol{\nu}^{(i)} + \boldsymbol{\delta}^{(i)} \text{ for some } \boldsymbol{\delta}^{(i)} && \text{(induction)} \\
&\sqsubseteq \boldsymbol{\nu}^{(i)} + D\boldsymbol{f}|^{*}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}) && (\boldsymbol{v} \sqsubseteq \boldsymbol{g}^{*}(\boldsymbol{v})) \\
&= \boldsymbol{\nu}^{(i+1)} && \text{(Lemma 5.3)} \\
&= \boldsymbol{\nu}^{(i)} + \boldsymbol{\delta}^{(i)} + D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}(D\boldsymbol{f}|^{*}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)})) && (\boldsymbol{g}^{*}(\boldsymbol{v}) = \boldsymbol{v} + \boldsymbol{g}(\boldsymbol{g}^{*}(\boldsymbol{v}))\ ) \\
&= \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) + D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}(D\boldsymbol{f}|^{*}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)})) && \text{(definition of } \boldsymbol{\delta}^{(i)}\text{)} \\
&\sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)} + D\boldsymbol{f}|^{*}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)})) && \text{(Lemma 5.4)} \\
&= \boldsymbol{f}(\boldsymbol{\nu}^{(i+1)}) && \text{(Lemma 5.3)}
\end{aligned}$$

Since $\boldsymbol{\nu}^{(i+1)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i+1)})$, there exists a $\boldsymbol{\delta}^{(i+1)}$ such that $\boldsymbol{\nu}^{(i+1)} + \boldsymbol{\delta}^{(i+1)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i+1)})$.

Next, we prove $\boldsymbol{f}(\boldsymbol{\nu}^{(i)}) \sqsubseteq \boldsymbol{\nu}^{(i+1)}$:

$$\begin{aligned}
\boldsymbol{f}(\boldsymbol{\nu}^{(i)}) &= \boldsymbol{\nu}^{(i)} + \boldsymbol{\delta}^{(i)} && \text{(as proved above)}\\
&\sqsubseteq \boldsymbol{\nu}^{(i)} + D\boldsymbol{f}|^{*}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}) && (\boldsymbol{\nu} \sqsubseteq \boldsymbol{g}^{*}(\boldsymbol{\nu}))\\
&= \boldsymbol{\nu}^{(i+1)} && \text{(Lemma 5.3)}
\end{aligned}$$

It remains to prove $\sup_{j\in\mathbb{N}} \boldsymbol{\kappa}^{(j)} = \mu \boldsymbol{f}$ and $\boldsymbol{\nu}^{(i)} \sqsubseteq \mu\boldsymbol{f}$ for all i. The equation $\sup_{j\in\mathbb{N}} \boldsymbol{\kappa}^{(j)} = \mu\boldsymbol{f}$ holds by Kleene's theorem (Proposition 2.4). To prove $\boldsymbol{\nu}^{(i)} \sqsubseteq \mu\boldsymbol{f}$, for all i, we need a lemma.

LEMMA 5.5. *Let $\boldsymbol{f}(\boldsymbol{x}) \sqsupseteq \boldsymbol{x}$. For all $d \geq 0$, there exists a vector $\boldsymbol{e}^{(d)}(\boldsymbol{x})$ such that*

$$\begin{aligned}
\boldsymbol{f}^{d}(\boldsymbol{x}) + \boldsymbol{e}^{(d)}(\boldsymbol{x}) &= \boldsymbol{f}^{d+1}(\boldsymbol{x}) \ \textit{and}\\
\boldsymbol{e}^{(d)}(\boldsymbol{x}) &\sqsupseteq D\boldsymbol{f}|_{\boldsymbol{f}^{d-1}(\boldsymbol{x})}(D\boldsymbol{f}|_{\boldsymbol{f}^{d-2}(\boldsymbol{x})}(\ldots D\boldsymbol{f}|_{\boldsymbol{x}}(\boldsymbol{e}^{(0)}(\boldsymbol{x}))\ldots))\\
&\sqsupseteq D\boldsymbol{f}|^{d}_{\boldsymbol{x}}(\boldsymbol{e}^{(0)}(\boldsymbol{x})).
\end{aligned}$$

PROOF OF LEMMA. By induction on d. For $d = 0$, there is an appropriate $\boldsymbol{e}^{(0)}(\boldsymbol{x})$ by assumption. Let $d \geq 0$.

$$\begin{aligned}
\boldsymbol{f}^{d+2}(\boldsymbol{x}) &= \boldsymbol{f}(\boldsymbol{f}^{d}(\boldsymbol{x}) + \boldsymbol{e}^{(d)}(\boldsymbol{x})) && \text{(induction)}\\
&\sqsupseteq \boldsymbol{f}^{d+1}(\boldsymbol{x}) + D\boldsymbol{f}|_{\boldsymbol{f}^{d}(\boldsymbol{x})}(\boldsymbol{e}^{(d)}(\boldsymbol{x})) && \text{(Lemma 5.4)}\\
&\sqsupseteq \boldsymbol{f}^{d+1}(\boldsymbol{x}) + D\boldsymbol{f}|_{\boldsymbol{f}^{d}(\boldsymbol{x})}(\ldots D\boldsymbol{f}|_{\boldsymbol{x}}(\boldsymbol{e}^{(0)}(\boldsymbol{x}))\ldots) && \text{(induction)}
\end{aligned}$$

Therefore, there exists an $\boldsymbol{e}^{(d+1)}(\boldsymbol{x}) \sqsupseteq D\boldsymbol{f}|_{\boldsymbol{f}^{d}(\boldsymbol{x})}(\cdots D\boldsymbol{f}|_{\boldsymbol{x}}(\boldsymbol{e}^{(0)}(\boldsymbol{x}))\cdots)$. Since $D\boldsymbol{f}|_{\boldsymbol{y}}$ is monotone in $\boldsymbol{y}$ and $\boldsymbol{x} \sqsubseteq \boldsymbol{f}(\boldsymbol{x}) \sqsubseteq \boldsymbol{f}^{2}(\boldsymbol{x}) \sqsubseteq \ldots$, the second inequality also holds. This completes the proof of the lemma. □

Notice that Lemma 5.5 holds for $\boldsymbol{x} = \boldsymbol{\nu}^{(i)}$ and $\boldsymbol{e}^{(0)}(\boldsymbol{\nu}^{(i)}) = \boldsymbol{\delta}^{(i)}$, because we have already shown $\boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)})$. Now we can prove $\boldsymbol{\nu}^{(i)} \sqsubseteq \mu\boldsymbol{f}$ by induction on i. The case $i = 0$ is trivial. Let $i \geq 0$. We have:

$$\begin{aligned}
\boldsymbol{\nu}^{(i+1)} &= \boldsymbol{\nu}^{(i)} + D\boldsymbol{f}|^{*}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}) && \text{(Lemma 5.3)}\\
&= \boldsymbol{\nu}^{(i)} + \sum_{d\in\mathbb{N}} D\boldsymbol{f}|^{d}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}) && \text{(definition of } D\boldsymbol{f}|^{*}_{\boldsymbol{\nu}^{(i)}})\\
&\sqsubseteq \boldsymbol{\nu}^{(i)} + \sum_{d\in\mathbb{N}} \boldsymbol{e}^{(d)}(\boldsymbol{\nu}^{(i)}) && \text{(Lemma 5.5)}\\
&= \sup_{d\in\mathbb{N}} \boldsymbol{f}^{d}(\boldsymbol{\nu}^{(i)}) && (\omega\text{-continuity})\\
&\sqsubseteq \mu\boldsymbol{f} && \text{(induction:}\\
& && \boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) \sqsubseteq \boldsymbol{f}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)})) \sqsubseteq \ldots \sqsubseteq \mu\boldsymbol{f})
\end{aligned}$$

This completes the proof of Proposition 5.1. □

5.1. UNIQUENESS. In Definition 3.6 the Newton approximant $\boldsymbol{v}^{(i)}$ is defined in terms of a vector $\boldsymbol{\delta}^{(i)}$ satisfying $\boldsymbol{v}^{(i)} + \boldsymbol{\delta}^{(i)} = \boldsymbol{f}(\boldsymbol{v}^{(i)})$. In the previous section we have shown that such a vector always exists. However, in a semiring there there may be multiple such $\boldsymbol{\delta}^{(i)}$'s, and so in principle there could be multiple Newton sequences. We show now that this is *not* the case, that is, there is only one Newton sequence $(\boldsymbol{v}^{(i)})_{i \in \mathbb{N}}$, independent of the choice of $\boldsymbol{\delta}^{(i)}$:

PROPOSITION 5.6. *Let $\boldsymbol{f} : V \to V$ be a vector of power series. There is exactly one Newton sequence $(\boldsymbol{v}^{(i)})_{i \in \mathbb{N}}$.*

Theorem 3.9 follows directly by combining Proposition 5.1 and Proposition 5.6. So, for Theorem 3.9, it remains to prove Proposition 5.6, which we do in the rest of this section.

It is convenient for this proof to introduce *substitutionals*, a notion related to differentials, see Definition 3.5.

Definition 5.7. Let f be a power series over an ω-continuous semiring $\mathcal{S}$ and let $s \in \mathbb{N}_+$. The *substitutional of f with respect to s* at the point $\boldsymbol{v}$ is the mapping $\$_s f|_{\boldsymbol{v}} : V \to S$ defined as follows:

If f is a monomial, that is, of the form $f = a_1 X_1 \cdots a_k X_k a_{k+1}$, then

$$\$_s f|_{\boldsymbol{v}}(\boldsymbol{b}) = \begin{cases} a_1 \boldsymbol{v}_{X_1} \cdots a_{s-1} \boldsymbol{v}_{X_{s-1}} a_s \boldsymbol{b}_{X_s} a_{s+1} \boldsymbol{v}_{X_{s+1}} \cdots a_k \boldsymbol{v}_{X_k} a_{k+1} & \text{if } 1 \leq s \leq k \\ 0 & \text{otherwise.} \end{cases}$$

If f is a power series, that is, of the form $f = \sum_{i \in I} f_i$, then

$$\$_s f|_{\boldsymbol{v}}(\boldsymbol{b}) = \sum_{i \in I} \$_s f_i|_{\boldsymbol{v}}(\boldsymbol{b}).$$

In other words: if f is a monomial with at least s variables then $\$_s f|_{\boldsymbol{v}}(\boldsymbol{b})$ is obtained from f by replacing the sth variable X_s by $\boldsymbol{b}_{X_s}$ and all other variables by the corresponding component of $\boldsymbol{v}$. If f is a monomial with less than s variables then $\$_s f|_{\boldsymbol{v}}(\boldsymbol{b}) = 0$. If f is a power series then the substitutional of f is the sum of the substitutionals of f's monomials.

Analogously to differentials, we extend the definition of substitutionals to vectors of power series by applying the substitution componentwise. Formally, we define the substitutional of a vector of power series $\boldsymbol{f}$ at $\boldsymbol{v}$ as the function $\$_s \boldsymbol{f}|_{\boldsymbol{v}} : V \to V$ with

$$\left(\$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b})\right)_X := \$_s \boldsymbol{f}_X|_{\boldsymbol{v}}(\boldsymbol{b}) \,.$$

Observe that, like the differential (see Remark 3.7), the substitutional is "linear", that is, $\$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b} + \boldsymbol{b}') = \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}) + \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}')$.

Notation 5.8. For any $j \in \mathbb{N}$ and any sequence $s = (s_1, \ldots s_j) \in \mathbb{N}_+^j$ we write $\$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b})$ for $\$_{s_1} \boldsymbol{f}|_{\boldsymbol{v}}(\$_{s_2} \boldsymbol{f}|_{\boldsymbol{v}}(\cdots \$_{s_j} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}) \cdots))$, and $\$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}) = \boldsymbol{b}$ if $j = 0$.

The following facts are immediate from the definitions.

PROPOSITION 5.9. *Let f be a monomial. Then*

$$D_X f|_{\boldsymbol{v}}(\boldsymbol{b}) = \sum \left\{ \$_s f|_{\boldsymbol{v}}(\boldsymbol{b}) \mid X \textit{ is the } s\textit{th variable in } f \right\} .$$

Let $\boldsymbol{f}$ be a vector of power series. Then:

(1) $D\boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b}) = \sum_{s \in \mathbb{N}_+} \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b})$.

(2) $D\boldsymbol{f}|_{\boldsymbol{v}}^{j}(\boldsymbol{b}) = \sum_{s \in \mathbb{N}_+^j} \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{b})$.

(3) *For all $s \in \mathbb{N}_+$ we have $\boldsymbol{f}(\boldsymbol{v}) \sqsupseteq \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v})$.*

Example 5.10. Consider the polynomial $f = aXYX + cY$. Then

$$\begin{aligned}
\$_1 f|_{\boldsymbol{v}}(\boldsymbol{b}) &= a\boldsymbol{b}_X\boldsymbol{v}_Y\boldsymbol{v}_X + c\boldsymbol{b}_Y \\
\$_2 f|_{\boldsymbol{v}}(\boldsymbol{b}) &= a\boldsymbol{v}_X\boldsymbol{b}_Y\boldsymbol{v}_X \\
\$_3 f|_{\boldsymbol{v}}(\boldsymbol{b}) &= a\boldsymbol{v}_X\boldsymbol{v}_Y\boldsymbol{b}_X \\
D_X f|_{\boldsymbol{v}}(\boldsymbol{b}) &= a\boldsymbol{b}_X\boldsymbol{v}_Y\boldsymbol{v}_X + a\boldsymbol{v}_X\boldsymbol{v}_Y\boldsymbol{b}_X \\
D_Y f|_{\boldsymbol{v}}(\boldsymbol{b}) &= a\boldsymbol{v}_X\boldsymbol{b}_Y\boldsymbol{v}_X + c\boldsymbol{b}_Y.
\end{aligned}$$

Observe that $Df|_{\boldsymbol{v}}(\boldsymbol{b}) = D_X f|_{\boldsymbol{v}}(\boldsymbol{b}) + D_Y f|_{\boldsymbol{v}}(\boldsymbol{b}) = \$_1 f|_{\boldsymbol{v}}(\boldsymbol{b}) + \$_2 f|_{\boldsymbol{v}}(\boldsymbol{b}) + \$_3 f|_{\boldsymbol{v}}(\boldsymbol{b})$ and that $f(\boldsymbol{v}) = a\boldsymbol{v}_X\boldsymbol{v}_Y\boldsymbol{v}_X + c\boldsymbol{v}_Y \sqsupseteq \$_s f|_{\boldsymbol{v}}(\boldsymbol{v})$ holds for all $s \in \mathbb{N}_+$. □

For the proof of Proposition 5.6, we need the following two lemmata.

LEMMA 5.11. *Let $\boldsymbol{f}$ be a vector of power series. Let $\boldsymbol{v} + \boldsymbol{\delta} = \boldsymbol{f}(\boldsymbol{v})$. Let $j \in \mathbb{N}$ and $(s_1, \ldots, s_{j+1}) \in \mathbb{N}_+^{j+1}$. Then $\boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j}(\boldsymbol{\delta}) \sqsupseteq \$_{(s_1,\ldots,s_{j+1})} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v})$.*

PROOF. By induction on j. For $j = 0$ we have $\boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq 0}(\boldsymbol{\delta}) = \boldsymbol{v} + \boldsymbol{\delta} = \boldsymbol{f}(\boldsymbol{v}) \sqsupseteq \$_{s_1} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v})$ by Proposition 5.9.3. Let $j \geq 0$. We have:

$$\begin{aligned}
\boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j+1}(\boldsymbol{\delta}) &= \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j}(\boldsymbol{\delta}) + D\boldsymbol{f}|_{\boldsymbol{v}}^{j+1}(\boldsymbol{\delta}) \\
&\sqsupseteq \$_{(s_1,\ldots,s_{j+1})} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v}) + D\boldsymbol{f}|_{\boldsymbol{v}}^{j+1}(\boldsymbol{\delta}) && \text{(induction)} \\
&\sqsupseteq \$_{(s_1,\ldots,s_{j+1})} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v}) + \$_{(s_1,\ldots,s_{j+1})} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{\delta}) && \text{(Prop. 5.9.2.)} \\
&= \$_{(s_1,\ldots,s_{j+1})} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{f}(\boldsymbol{v})) && (\boldsymbol{v} + \boldsymbol{\delta} = \boldsymbol{f}(\boldsymbol{v})) \\
&\sqsupseteq \$_{(s_1,\ldots,s_{j+1})} \boldsymbol{f}|_{\boldsymbol{v}}(\$_{s_{j+2}} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v})) && \text{(Prop. 5.9.3.)} \\
&= \$_{(s_1,\ldots,s_{j+2})} \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v}) && \square
\end{aligned}$$

LEMMA 5.12. *Let $\boldsymbol{f}$ be a vector of power series. Let $\boldsymbol{v} + \boldsymbol{\delta} = \boldsymbol{v} + \boldsymbol{\delta}' = \boldsymbol{f}(\boldsymbol{v})$. Then $\boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{*}(\boldsymbol{\delta}) = \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{*}(\boldsymbol{\delta}')$.*

PROOF. We show $\boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j}(\boldsymbol{\delta}) = \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j}(\boldsymbol{\delta}')$ for all $j \in \mathbb{N}$. Then the lemma follows by ω-continuity. We proceed by induction on j. The induction base ($j = 0$) is clear. Let $j \geq 0$. We have:

$$\begin{aligned}
\boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j+1}(\boldsymbol{\delta}) &= \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j}(\boldsymbol{\delta}) + D\boldsymbol{f}|_{\boldsymbol{v}}^{j+1}(\boldsymbol{\delta}) \\
&= \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j}(\boldsymbol{\delta}') + D\boldsymbol{f}|_{\boldsymbol{v}}^{j+1}(\boldsymbol{\delta}) && \text{(induction)} \\
&= \underbrace{\boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\leq j}(\boldsymbol{\delta}')}_{=:\boldsymbol{u}} + \sum_{s \in \mathbb{N}_+^{j+1}} \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{\delta}) && \text{(Prop. 5.9.2.)}
\end{aligned}$$

By Lemma 5.11, we have $\boldsymbol{u} \sqsupseteq \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v})$ for all $s \in \mathbb{N}_+^{j+1}$. In other words, for all $s \in \mathbb{N}_+^{j+1}$ there is a $\boldsymbol{u}'$ such that $\boldsymbol{u} = \boldsymbol{u}' + \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v})$. Hence, for all $s \in \mathbb{N}_+^{j+1}$, we have $\boldsymbol{u} + \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{\delta}) = \boldsymbol{u}' + \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{v}) + \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{\delta}) = \boldsymbol{u}' + \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{f}(\boldsymbol{v})) = \boldsymbol{u} + \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{\delta}')$. Therefore, in the above equation, we can replace $\boldsymbol{\delta}$ by $\boldsymbol{\delta}'$ due to the "presence" of $\boldsymbol{u}$:

$$\begin{aligned}
&= \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\le j}(\boldsymbol{\delta}') + \sum_{s \in \mathbb{N}_+^{j+1}} \$_s \boldsymbol{f}|_{\boldsymbol{v}}(\boldsymbol{\delta}') \quad \text{(as argued above)} \\
&= \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\le j}(\boldsymbol{\delta}') + D\boldsymbol{f}|_{\boldsymbol{v}}^{j+1}(\boldsymbol{\delta}') \qquad \text{(Prop. 5.9.2.)} \\
&= \boldsymbol{v} + D\boldsymbol{f}|_{\boldsymbol{v}}^{\le j+1}(\boldsymbol{\delta}') \qquad \square
\end{aligned}$$

Now Proposition 5.6 follows immediately from Lemma 5.12 by a straightforward inductive proof. □

6. *Derivation Trees and the Newton Approximants*

The proofs of the previous section were purely algebraical. For deeper and stronger results, we need the notion of *derivation trees*. To this end, we reinterpret a system of power-series as a context-free grammar, and assign it a set of *derivation trees*. We then characterize the Kleene and Newton approximants of the system in terms of those trees. This characterization of the Newton approximants will be crucially used in the rest of this article.

We assume that the reader is familiar with the notion of derivation tree of a context-free grammar. Recall that the yield of a derivation tree (obtained by reading the leaves from left to right) is a word generated by the grammar, and every word generated by the grammar is the yield of one or more derivation trees. In our reinterpretation, the nonterminals will be the variables of the system of power series, and the terminals will be its coefficients.

We show that the Kleene approximants $\boldsymbol{\kappa}^{(i)}$ are equal to the sum of the yields of the derivation trees having a certain height. Similarly, we show that the Newton approximants $\boldsymbol{v}^{(i)}$ are equal to the sum of the yields of the trees having a certain *dimension*, a notion introduced in Definition 6.7 below.

For the rest of the section, we fix a vector $\boldsymbol{f}$ of power series over a fixed but arbitrary ω-continuous semiring. Without loss of generality, we assume that $\boldsymbol{f}_X = \sum_{j \in J} m_{X,j}$ holds for every variable $X \in \mathcal{X}$, that is, we assume that for all variables the sum is over the same countable set J of indices.

Consider the set of ordered trees whose nodes are labeled by pairs (X, j), where $X \in \mathcal{X}$ and $j \in J$. Sometimes we identify a tree and its root. In particular, we say that a tree t is labeled by (X, j) if its root is labeled by (X, j). The mappings λ, λ_v and λ_m are defined by $\lambda(t) := (X, j)$, $\lambda_v(t) := X$, and $\lambda_m(t) := j$. Given a set T of trees, we denote by T_X the set of trees $t \in T$ such that $\lambda_v(t) = X$.

We define the set of derivation trees of $\boldsymbol{f}$, and show how to assign to each tree a semiring element called the yield of the tree. For technical reasons our definition differs slightly from the straightforward generalization of derivation trees for grammars.

Definition 6.1 (*Derivation Tree, Yield*). The *derivation trees* of $\boldsymbol{f}$ and their *yields* are inductively defined as follows:

$$\begin{aligned} X &= aXY + b \\ Y &= cYZ + dYX + e \\ Z &= gXh + i \end{aligned}$$

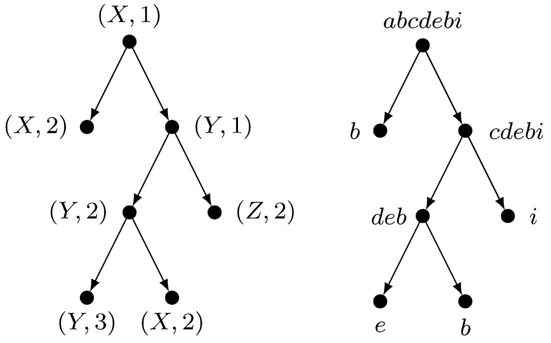


FIG. 10. A system of equations, a derivation tree, and its yield.

—For every monomial $m_{X,j}$ of $\boldsymbol{f}_X$, if no variable occurs in $m_{X,j}$, then the tree t consisting of one single node labeled by (X, j) is a derivation tree of $\boldsymbol{f}$. Its yield $Y(t)$ is equal to $m_{X,j}$.

—Let $m_{X,j} = a_1 X_1 a_2 X_2 \ldots a_k X_k a_{k+1}$ for some $k \geq 1$, and let $t_1, \ldots, t_k$ be derivation trees of $\boldsymbol{f}$ such that $\lambda_v(t_i) = X_i$ for $1 \leq i \leq k$. Then, the tree t labelled by (X, j) and having $t_1, \ldots, t_k$ as (ordered) children is also a derivation tree of $\boldsymbol{f}$, and its yield $Y(t)$ is equal to $a_1 Y(t_1) \ldots a_k Y(t_k) a_{k+1}$.

The *yield* $Y(T)$ of a countable set T of derivation trees is defined by $Y(T) = \sum_{t \in T} Y(t)$. In the following, we mean *derivation tree* whenever we say *tree*.

Example 6.2. Figure 10 shows a system of equations (system (1) from the introduction, on the left). The basic idea is to read these equations as rules of a context-free grammar, for example, the equation $X = aXY + b$ is interpreted as the rules $X \rightarrow aXY$ and $X \rightarrow b$. By this reinterpretation derivation trees are naturally associated with the given equation system. But as addition is not assumed to be idempotent in general, we have to extend the standard definition of derivation tree in order to handle multiplicities correctly. The derivation tree depicted in the middle of Figure 10 therefore records which monomial of which variable gives rise to the children of a given node. For instance, consider the node labelled by $(Y, 1)$ (the right child of the root). Since the first monomial of the equation for Y is cYZ, the node has two children, say c_1, c_2 with $\lambda_v(c_1) = Y$ and $\lambda_v(c_2) = Z$. As $\lambda_m(c_2) = 2$, the children of c_2 are determined by the second monomial of the equation for Z. Since this monomial is h, which contains no variables, c_2 has no children. The right part of the figure shows the result of labelling each node of the tree with the yield of the subtree rooted at it.

6.1. KLEENE SEQUENCE AND HEIGHT.

As a warm-up for the Newton case, we characterize the Kleene sequence $(\boldsymbol{\kappa}^{(i)})_{i \in \mathbb{N}}$ in terms of the derivation trees of a certain height.

Definition 6.3 (*Height*). Let t be a derivation tree. The *height* of t, denoted by $h(t)$, is the length (number of edges) of a longest path from the root to some leaf. We denote by $\mathcal{H}^i$ the set of derivation trees of height at most i.

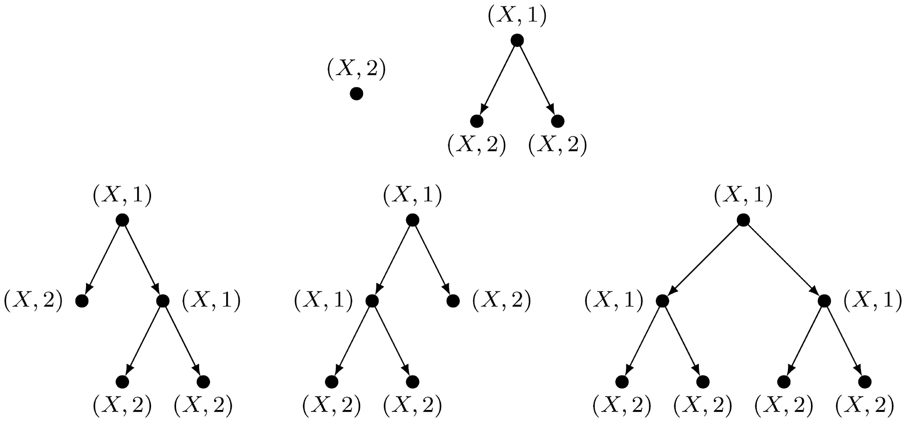


FIG. 11. Trees of height at most 2 for the equation $X = 1/2 \cdot X^2 + 1/2$.

PROPOSITION 6.4. $\left(\boldsymbol{\kappa}^{(i)}\right)_X = Y(\mathcal{H}^i_X)$, *i.e., the X-component of the i-th Kleene approximant* $\boldsymbol{\kappa}^{(i)}$ *is equal to the yield of* $\mathcal{H}^i_X$.

The proof can be found in Appendix A.

Notice that Proposition 6.4 no longer holds if nodes are only labelled with a variable, and not with a pair. Consider for instance the equation $X = a + a$, for which $\kappa^{(0)} = a + a$. There are two derivation trees t_1, t_2 of height 0, both consisting of one single node: t_1 is labelled by $(X, 1)$, and t_2 by $(X, 2)$. We get $Y(t_1) + Y(t_2) = a + a = \kappa^{(0)}$. If we labelled nodes only with variables, then there would be one single derivation tree t, and we would get $Y(t) = a$, which in general is different from $a + a$.

Example 6.5. Consider again the equation $X = 1/2 \cdot X^2 + 1/2$ over the real semiring. We have $\kappa^{(2)} = 89/128$. Figure 11 shows the five derivation trees of height at most 2. It is easy to see that their yields are 1/2, 1/8, 1/32, 1/32, 1/128, which add up to 89/128.

By Kleene's theorem we obtain that the least solution of the equation system is equal to the yield of the set of all trees.

COROLLARY 6.6. *Let* $\mathcal{T}$ *be the set of all derivation trees of* $\boldsymbol{f}$. *For all* $X \in \mathcal{X}$: $(\mu \boldsymbol{f})_X = Y(\mathcal{T}_X)$.

PROOF. By Kleene's Theorem (Proposition 2.4), we have $(\mu \boldsymbol{f})_X = \sup_{i \in \mathbb{N}} (\boldsymbol{\kappa}^{(i)})_X$. The result follows from Proposition 6.4. □

6.2. NEWTON SEQUENCE AND DIMENSION. We introduce a second parameter of a tree, namely its *dimension*. Like the height, it depends only on the tree structure, and not on the labels of its nodes. Loosely speaking, a tree has dimension 0 if it consists of just one node; a tree has dimension i if there is a path from its root to some node which has at least two children with dimension $i - 1$ and all subtrees of the path that are not themselves on the path have dimension at most $i - 1$. The path is called the *backbone* of the tree. The geometric intuition for the name *dimension* is that a tree of dimension i can be naturally represented in $\mathbb{R}^i$: a tree of dimension 1 can essentially be represented as a line (with small "spikes", see

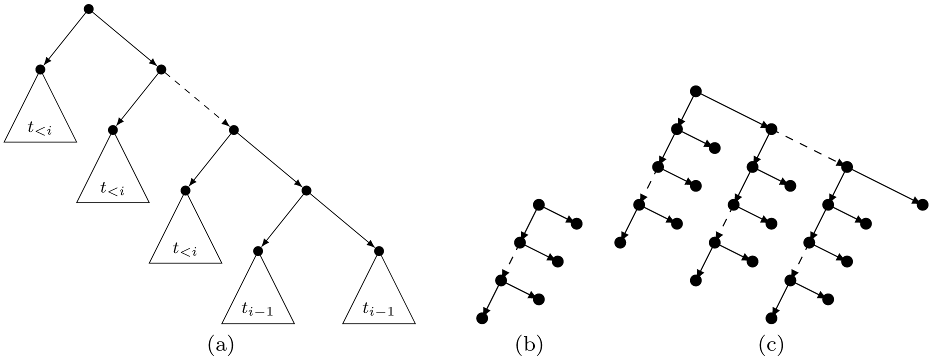


FIG. 12. (a) shows the general structure of a tree of dimension i, where $t_{<i}$ (resp. t_{i-1}) represents any tree of dimension $< i$ (resp. $= i - 1$). (b) and (c) give some idea of the topology of one-, resp. two-dimensional trees.

Figure 12(b)); a tree of dimension 2 can be drawn in the plane, with the backbone as a line, and the subtrees of dimension 1 as lines perpendicular to the backbone (see Figure 12(c)). In general, the subtrees of an i-dimensional tree are drawn in hyperplanes orthogonal to the line for the backbone, yielding a representation in $\mathbb{R}^i$. To the best of our knowledge, the notion of dimension has not been used before. Formally, we use an inductive definition of dimension that is more convenient for proofs.

Definition 6.7 (*Dimension*). The *dimension* $d(t)$ of a tree t is inductively defined as follows:

(1) If t has no children, then $d(t) = 0$.
(2) If t has exactly one child t_1, then $d(t) = d(t_1)$.
(3) If t has at least two children, let t_1, t_2 be two distinct children of t such that $d(t_1) \geq d(t_2)$ and $d(t_2) \geq d(t')$ for every child $t' \neq t_1$. Let $d_1 = d(t_1)$ and $d_2 = d(t_2)$. Then

$$d(t) = \begin{cases} d_1 + 1 & \text{if } d_1 = d_2 \\ d_1 & \text{if } d_1 > d_2. \end{cases}$$

We denote by $\mathcal{D}^i$ the set of derivation trees of dimension at most i.

Remark: It is easy to prove by induction that $h(t) \geq d(t)$ holds for every derivation tree t.

In the rest of the section we show that the i-th Newton approximant $\boldsymbol{\nu}^{(i)}$ is equal to the yield of the derivation trees of dimension at most i:

THEOREM 6.8 (TREE CHARACTERIZATION OF THE NEWTON SEQUENCE). *Let $(\boldsymbol{\nu}^{(i)})_{i\in\mathbb{N}}$ be the Newton sequence of $\boldsymbol{f}$. For every $X \in \mathcal{X}$ and every $i \geq 0$ we have $\left(\boldsymbol{\nu}^{(i)}\right)_X = Y(\mathcal{D}^i_X)$, i.e., the X-component of the i-th Newton approximant is equal to the yield of $\mathcal{D}^i_X$.*

The proof is as follows. We define, in terms of trees, a sequence $(\boldsymbol{\tau}^{(i)})_{i\in\mathbb{N}}$ satisfying $\boldsymbol{\tau}^{(i)}_X = Y(\mathcal{D}^i_X)$ (Lemma 6.10), and we prove that it is a Newton sequence (Lemma 6.11). As the Newton sequence is unique by Proposition 5.6, we have $\boldsymbol{\tau}^{(i)} = \boldsymbol{\nu}^{(i)}$ and Theorem 6.8 follows.

We need the following definition.

Definition 6.9. A tree t is *proper* if $d(t) > d(t')$ for every child t' of t. For every $i \geq 0$, let P^i be the set of proper trees of dimension i. Define the sequence $(\boldsymbol{\tau}^{(i)})_{i\in\mathbb{N}}$ as follows:

$$\begin{aligned} \boldsymbol{\tau}^{(0)} &= \boldsymbol{f}(\mathbf{0}) \\ \boldsymbol{\tau}^{(i+1)} &= \boldsymbol{\tau}^{(i)} + D\boldsymbol{f}|^*_{\boldsymbol{\tau}^{(i)}}(\boldsymbol{\delta}^{(i)})\,, \end{aligned}$$

where $\boldsymbol{\delta}^{(i)}_X = Y(P^{i+1}_X)$ for all $X \in \mathcal{X}$.

LEMMA 6.10. *For every variable $X \in \mathcal{X}$ and every $i \geq 0$: $\boldsymbol{\tau}^{(i)}_X = Y(\mathcal{D}^i_X)$.*

LEMMA 6.11. *The sequence $(\boldsymbol{\tau}^{(i)})_{i\in\mathbb{N}}$ is a Newton sequence as defined in Definition* 3.6*, that is, the $\boldsymbol{\delta}^{(i)}$ of Definition* 6.9 *satisfy $\boldsymbol{f}(\boldsymbol{\tau}^{(i)}) = \boldsymbol{\tau}^{(i)} + \boldsymbol{\delta}^{(i)}$.*

The proofs of Lemma 6.10 and Lemma 6.11 can be found in Appendix A.

Example 6.12. Let us recall our example from the introduction (cf. Figure 1) with the equations

$$\begin{aligned} X &= a \cdot X \cdot Y + b \\ Y &= c \cdot Y \cdot Z + d \cdot Y \cdot X + e \\ Z &= g \cdot X \cdot h + i. \end{aligned}$$

Using our characterizations of $\boldsymbol{\kappa}^{(i)}$ and $\boldsymbol{\nu}^{(i)}$ by means of derivation trees we see that (a) every derivation tree t represents a terminating run of the procedure $\lambda(t)$, and, thus, (b) while $\boldsymbol{\kappa}^{(i)}$ only corresponds to a finite set of trees (runs), for $i > 0$ every $\boldsymbol{\nu}^{(i)}$ corresponds to an infinite set of runs. Hence, it is not very surprising that in general the Newton approximants give a better approximation of the (abstract) semantics of a program than the Kleene approximants.

7. *Idempotent Semirings*

Recall that in the algebraic structure underlying the framework of Sharir and Pnueli [1981] the summation operator is given by the join of a semilattice and, thus, summation is idempotent. We therefore study in this section the properties of our generalized Newton's method for this special case of ω-continuous semirings satisfying the additional axiom of idempotent addition. We simply call such semirings *idempotent* ω-continuous semirings, or just idempotent semirings in the following. In idempotent semirings, the natural order can be characterized as follows: $a \sqsubseteq b$ holds if and only if $a + b = b$. This is because $a \sqsubseteq b$ means by definition that there is a c such that $a + c = b$. Then, we have $a + b = a + a + c = a + c = b$. This extends analogously to vectors.

We start by showing that in the idempotent case the definition of the Newton sequence $(\boldsymbol{\nu}^{(i)})_{i\in\mathbb{N}}$ can be simplified.

PROPOSITION 7.1. *Let $\boldsymbol{f}$ be a vector of power series over an idempotent semiring. Let $(\boldsymbol{\nu}^{(i)})_{i\in\mathbb{N}}$ denote the Newton sequence of $\boldsymbol{f}$. It satisfies the following equations for all $i \in \mathbb{N}$:*

(a) $\boldsymbol{\nu}^{(i+1)} = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)}))$

(b) $\boldsymbol{\nu}^{(i+1)} = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)})$

(c) $\boldsymbol{\nu}^{(i+1)} = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0}))$

PROOF. We first show (a). By Theorem 3.9, we have $\boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)})$, hence with idempotence $\boldsymbol{\nu}^{(i)} + \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) = \boldsymbol{f}(\boldsymbol{\nu}^{(i)})$. So we can choose $\boldsymbol{\delta}^{(i)} = \boldsymbol{f}(\boldsymbol{\nu}^{(i)})$ and have $\boldsymbol{\nu}^{(i+1)} = \boldsymbol{\nu}^{(i)} + D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)})) = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)}))$, because $\boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)}))$. So (a) is shown.

Again by Theorem 3.9, we have $\boldsymbol{f}(\mathbf{0}) = \boldsymbol{\nu}^{(0)} \sqsubseteq \boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)})$. So we have, $D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0})) \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)}))$. Hence, for (b) and (c), it remains to show $D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)})) \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)})$ and $D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0}))$, respectively. For (b), we have:

$$\begin{aligned}
& D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\boldsymbol{\nu}^{(i)})) \\
& \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0}) + D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)})) && \text{(Lemma 5.4)} \\
& = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0})) + D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)})) \\
& \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) + D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)})) && (\boldsymbol{f}(\mathbf{0}) \sqsubseteq \boldsymbol{\nu}^{(i)}) \\
& \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) + D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) && \text{(Lemma 5.3)} \\
& = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) && \text{(idempotence)}
\end{aligned}$$

So (b) is shown.

For (c) it remains to show $D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0}))$. We proceed by induction on i. The base case $i = 0$ is easy because $\boldsymbol{\nu}^{(0)} = \boldsymbol{f}(\mathbf{0})$. Let $i \geq 1$. We have:

$$\begin{aligned}
& D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)}) \\
& = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i-1)}}(\boldsymbol{\nu}^{(i-1)})) && \text{(by (b))} \\
& \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i-1)}}(\boldsymbol{f}(\mathbf{0}))) && \text{(by induction)} \\
& \sqsubseteq D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0}))) && \text{(Theorem 3.9: } \boldsymbol{\nu}^{(i-1)} \sqsubseteq \boldsymbol{\nu}^{(i)}) \\
& = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{f}(\mathbf{0})) && \text{(see explanation below)}
\end{aligned}$$

For the last step we used that in the idempotent case we have $\boldsymbol{g}^*(\boldsymbol{g}^*(\boldsymbol{x})) = \boldsymbol{g}^*(\boldsymbol{x})$ for any linear map $\boldsymbol{g} : V \to V$. Recall that Remark 3.2.1 states that $D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}$ is linear.

$$\begin{aligned}
\boldsymbol{g}^*(\boldsymbol{g}^*(\boldsymbol{x})) &= \sum_{j\in\mathbb{N}} \boldsymbol{g}^j \left(\sum_{k\in\mathbb{N}} \boldsymbol{g}^k(\boldsymbol{x}) \right) && \text{(Definition 5.2)} \\
&= \sum_{j\in\mathbb{N}} \sum_{k\in\mathbb{N}} \boldsymbol{g}^j(\boldsymbol{g}^k(\boldsymbol{x})) && \text{(linearity)} \\
&= \sum_{l\in\mathbb{N}} \boldsymbol{g}^l(\boldsymbol{x}) && \text{(idempotence)} \\
&= \boldsymbol{g}^*(\boldsymbol{x}) && \text{(Definition 5.2)}
\end{aligned}$$

This concludes the proof. □

In the rest of the section we study *commutative* idempotent semirings. where not only addition is idempotent, but multiplication is commutative. We will use the abbreviation *ci-semirings* for such ω-continuous semirings in the following.

An instance of the Newton sequence in a ci-semiring has already been presented in the counting semiring example on page 15. We show another one here.

Example 7.2. Let $\langle 2^{\{a\}^*}, +, \cdot, 0, 1\rangle$ denote the ci-semiring $\langle 2^{\{a\}^*}, \cup, \cdot, \emptyset, \{\varepsilon\}\rangle$. The multiplication $\cdot$ is meant to be commutative. For simplicity, we write a^i instead of $\{a^i\}$. Consider $\boldsymbol{f}(X_1, X_2) = (X_2^2 + a,\ X_1^2)$. We have:

$$D\boldsymbol{f}|_{(v_1,v_2)}(X_1, X_2) = \big(v_2 X_2,\ v_1 X_1\big)$$

and

$$D\boldsymbol{f}|^*_{(v_1,v_2)}(X_1, X_2) = (v_1 v_2)^*\big(X_1 + v_2 X_2,\ v_1 X_1 + X_2\big).$$

The first three elements of the Newton sequence are:

$$\boldsymbol{\nu}^{(0)} = (a, 0), \quad \boldsymbol{\nu}^{(1)} = (a, a^2), \quad \boldsymbol{\nu}^{(2)} = (a^3)^*(a, a^2).$$

It is easy to check that $\boldsymbol{\nu}^{(2)}$ is a fixed point of $\boldsymbol{f}$. Hence we have $\boldsymbol{\nu}^{(2)} = \mu\boldsymbol{f}$, as $\boldsymbol{\nu}^{(2)} \sqsubseteq \mu\boldsymbol{f}$ by Theorem 3.9. □

In the case of ci-semirings the behaviors of the Kleene and Newton sequence differ very much: while the Kleene sequence may still need infinitely many steps, the Newton sequence *always* reaches $\mu\boldsymbol{f}$ after finitely many. This was first shown by Hopkins and Kozen in 7.3. Hopkins and Kozen defined the sequence $(\boldsymbol{\nu}^{(i)})_{i\in\mathbb{N}}$ directly through the equations $\boldsymbol{\nu}^{(0)} = \boldsymbol{f}(\boldsymbol{0})$ and $\boldsymbol{\nu}^{(i+1)} = D\boldsymbol{f}|^*_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\nu}^{(i)})$ from Proposition 7.1(b), without noticing the connection to Newton's method (which is not surprising, since in the idempotent case the original equations get masked). They proved the following result, which gives a $O(3^n)$ upper bound for the number of Newton iterations required for a system of n equations:

THEOREM 7.3 ([HOPKINS AND KOZEN 1999]). *Let $\boldsymbol{f}$ be a vector of power series over a ci-semiring and a set $\mathcal{X}$ of variables with $|\mathcal{X}| = n$. There is a function $P : \mathbb{N} \to \mathbb{N}$ with $P(n) \in \mathcal{O}(3^n)$ such that $\boldsymbol{\nu}^{(P(n))} = \mu\boldsymbol{f}$.*

In Section 7.1, we improve Theorem 7.3 by showing that it holds with $P(n) = n$. This is achieved through our characterisation of the Newton approximants in terms of derivation trees.

7.1. ANALYSIS OF THE CONVERGENCE SPEED. We analyze how many steps the Newton iteration and, equivalently, the Hopkins-Kozen iteration need to reach $\mu\boldsymbol{f}$ when we consider ci-semirings.

Recall from Section 6 the concept of derivation trees (short: trees). A tree t has a height $h(t)$, a dimension $d(t)$, and a yield $Y(t)$. We define yet another tree property.

Definition 7.4. A tree t is *compact* if $d(t) \leq L(t)$, where $L(t)$ denotes the number of distinct λ_v-labels in t.

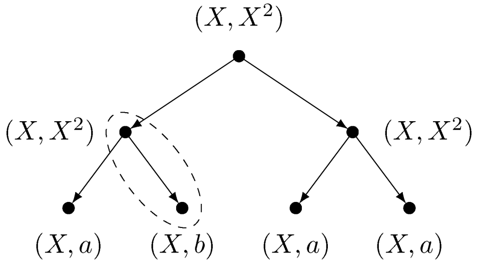
Now we are ready to prove the key lemma of this section, which states that any tree can be made compact.

LEMMA 7.5. *For each tree t there is a compact tree t' with $\lambda_v(t) = \lambda_v(t')$ and $Y(t) = Y(t')$.*

Example 7.6. We first sketch the proof of the lemma by means of an example. Consider the following univariate polynomial equation system:

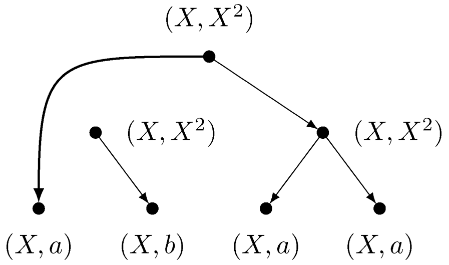
$$X = f(X) := X^2 + a + b.$$

Consider now the following tree $t \in \mathcal{T}_X$.[6]

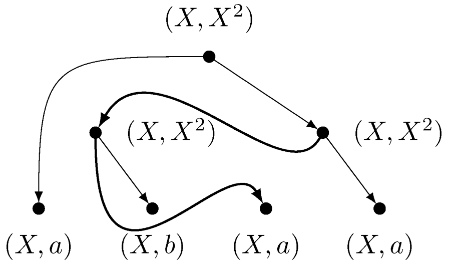


This tree has dimension 2 and is therefore not compact by definition. In order to make it compact, we have to transform it into a derivation tree of f which is of dimension 1 without changing its yield nor the variable-label of the root.

The idea is to reduce the left subtree to a tree of dimension 0 by reallocating "pump trees" (encircled in the above figure) into the right subtree; after that, we deal recursively with the right subtree.[7] We first remove such a pump tree from the rest of the tree by deleting the connecting edges and connecting the remaining parts as depicted here:



Note that we can introduce the new edge because the roots of the pump tree and the remaining subtree, in our example the leftmost leaf, are labeled by the same variable. Next, we reallocate the detached pump tree into the right subtree, e.g. as shown here:



It is easy to check that this new tree is indeed a derivation tree of f, and has the

[6] To improve readability in the following illustrations, we replace the node labels $(X, 1)$, $(X, 2)$, $(X, 3)$ by (X, X^2), (X, a), (X, b), respectively.

[7] Here, with "pump tree" we refer to partial derivation trees one adds or removes in the proof of the pumping lemma for context-free grammars.

same yield as the original one. Further, this tree is already compact. In general, we would have to proceed recursively in order to make the right subtree compact.

Note that, as we assume multiplication to be commutative, it is not important where we insert the pump tree into the right subtree. In the following proof, we show that we can always find such pump trees and relocate them, that is find insertion points, if the tree under consideration is not compact. □

We now give a formal proof of Lemma 7.5:

PROOF. We write $t = t_1 \cdot t_2$ to denote that t is combined from t_1 and t_2 in the following way: The tree t_1 is a "partial" derivation tree, i.e., a regular derivation tree except for one leaf l missing its children. The tree t_2 is a derivation tree with $\lambda_v(t_2) = \lambda_v(l)$. The tree t is obtained from t_1 and t_2 by replacing the leaf l of t_1 by the tree t_2.

We proceed by induction on the number of nodes. In the base case, t has just one node, so $d(t) = 0$, hence t is compact, and we are done. In the following, assume that t has more than one node and $d(t) > L(t)$ holds. We show how to construct a compact tree from t.

Let without loss of generality $s_1, s_2, \ldots, s_r$ be the children of t with $d(t) \geq d(s_1) \geq d(s_2) \geq \cdots \geq d(s_r)$. By induction we can make every child compact, that is, $d(s_i) \leq L(s_i)$. We then have by definition of dimension

$$L(t) + 1 \leq d(t) \leq d(s_1) + 1 \leq L(s_1) + 1 \leq L(t) + 1.$$

Hence, we have $d(t) = d(s_1)+1$ which, by definition of dimension and compactness, implies $d(s_1) = d(s_2) = L(t) = L(s_1) = L(s_2)$. As $h(s_2) \geq d(s_2) = L(s_2)$ by the remark after Definition 6.7, we find a path in s_2 from the root to a leaf which passes through at least two nodes with the same λ_v-label, say X_j. In other words, we may factor s_2 into $t_1^b \cdot (t_2^b \cdot t_3^b)$ such that $\lambda_v(t_2^b) = \lambda_v(t_3^b) = X_j$. As $L(t) = L(s_1) = L(s_2)$, we also find a node of s_1 labelled by X_j which allows us to write $s_1 = t_1^a \cdot t_2^a$ with $\lambda_v(t_2^a) = X_j$.

Now we move the middle part of s_2 to s_1, i.e., let $s_1' = t_1^a \cdot (t_2^b \cdot t_3^a)$ and let $s_2' = t_1^b \cdot t_3^b$. We then have $L(s_1') = L(s_1) = L(s_2) \geq L(s_2')$. By induction, s_1' and s_2' can be made compact, so $d(s_1') \leq d(s_1) = d(s_2) \geq d(s_2')$. Consider the tree t' obtained from t by replacing s_1 by s_1' and s_2 by s_2'. By commutativity, t and t' have the same yield. If $d(s_2') < d(s_2)$ then $d(t') \leq d(t) - 1 = L(t) = L(t')$ and we are done. Otherwise we iterate the described procedure.

This procedure terminates, because the number of nodes of (the current) s_2 strictly decreases in every iteration, and the number of nodes is an upper bound for $h(s_2)$ and, therefore, for $d(s_2)$. □

Now we can prove the main theorem of this section.

THEOREM 7.7. *Let $\boldsymbol{f}$ be a vector of power series over a ci-semiring $\mathcal{S}$ given in the set $\mathcal{X}$ of variables with $|\mathcal{X}| = n$. Then $\boldsymbol{\nu}^{(n)} = \mu \boldsymbol{f}$.*

PROOF. We have for all $X \in \mathcal{X}$:

$$\begin{aligned}(\mu \boldsymbol{f})_X &= \sum_{\text{trees } t \text{ with } \lambda_v(t)=X} Y(t) && \text{(Corollary 6.6)}\\ &= \sum_{\substack{\text{trees } t \text{ with } \lambda_v(t)=X\\ \text{and } d(t)\leq n}} Y(t) && \text{(Lemma 7.5)}\\ &= (\boldsymbol{\nu}^{(n)})_X && \text{(Theorem 6.8)} \qquad \square\end{aligned}$$

Remark 7.8 The bound of this theorem is tight, as shown by the following example: If $\boldsymbol{f}(X_1, \ldots, X_n) = (X_2^2 + a, X_3^2, \ldots, X_n^2, X_1^2)$, then $(\boldsymbol{\nu}^{(k)})_{X_1} = a$ for $k < n$, but $a^{2^n} \leq (\boldsymbol{\nu}^{(n)})_{X_1} = (\mu \boldsymbol{f})_{X_1}$.

8. *Non-Distributive Program Analyses*

In this article, we have focused on *distributive* program analyses, which allows us to use semirings as algebraic structure. Recall that semirings are distributive, that is, all semiring elements a, b, c satisfy $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(a+b) \cdot c = a \cdot c + b \cdot c$.

Distributive intraprocedural analyses (i.e., for programs without procedures) were considered first in Kildall [1973]. This seminal paper showed that, given a program and the distributive transfer functions of a program analysis, one can construct a vector $\boldsymbol{f}$ of polynomials such that, for every program point p, the p-component of the least fixed point $\mu \boldsymbol{f}$ coincides with the *JOP*$_0$*-value*, that is, the join over all valid paths where for every procedure call there is a matching return (as described in the introduction 1.1).

The framework of Kildall [1973] was generalized to nondistributive transfer functions in Kam and Ullman [1977]. Nondistributivity means, in our terms, that only *subdistributivity* holds: $a \cdot (b+c) \sqsupseteq a \cdot b + a \cdot c$ and $(a+b) \cdot c \sqsupseteq a \cdot c + b \cdot c$.[8] There are interesting program analyses, such as constant propagation, which are nondistributive, see for example, Kam and Ullman [1977], and Nielson et al. [1999]. In those cases, the least fixed point does not necessarily coincide with the JOP$_0$-value, but rather safely approximates ("overapproximates") it.

Sharir and Pnueli [1981] extended the work of Kildall [1973] to the interprocedural case. The generalization to nondistributive analyses was done by Knoop and Steffen [1992], who proved that, as in the intraprocedural case, the least fixed point is an overapproximation of the JOP$_0$-value.

We define the JOP$_0$-value as the vector $\boldsymbol{M}$ with $\boldsymbol{M}_p = Y(\mathcal{T}_p)$, where $\mathcal{T}_p$ is the set of trees labeled with p. Notice that a depth-first traversal of a tree labeled with p precisely corresponds to an interprocedural path from the beginning of the procedure of p to the program point p, that is, the JOP$_0$-value $\boldsymbol{M}_p = Y(\mathcal{T}_p)$ is indeed the sum of the dataflow values of all paths to p. Corollary 6.6 states that $\boldsymbol{M} = \mu \boldsymbol{f}$ holds in the distributive case. Proposition 2.4 and Theorem 3.9 show that the Kleene and Newton sequences converge to this value.

[8] If addition is idempotent (as for lattice joins) this condition is equivalent to the monotonicity of multiplication, or, in traditional terms, to the monotonicity of the transfer functions [Kam and Ullman 1977]. The stricter distributivity condition, on the other hand, amounts to requiring the transfer functions to be homomorphisms.

For the nondistributive case, the least fixed point overapproximates the JOP$_0$-value, that is, $\boldsymbol{M} \sqsubseteq \mu \boldsymbol{f}$, cf. Knoop and Steffen [1992].

In the following, we show that Newton's method is still welldefined in "subdistributive semirings", and that the Kleene and Newton sequences both converge to overapproximations of $\boldsymbol{M}$, more precisely, we show $\boldsymbol{M} \sqsubseteq \sup_{i\in\mathbb{N}} \boldsymbol{\kappa}^{(i)} \sqsubseteq \sup_{i\in\mathbb{N}} \boldsymbol{\nu}^{(i)}$.

For this, we first define *subdistributive (ω-complete) semirings*[9]:

Definition 8.1. A subdistributive semiring is a tuple $\langle S, +, \cdot, 0, 1\rangle$ satisfying the following properties:

(1) $\langle S, +, 0\rangle$ is a commutative monoid.
(2) $\langle S, \cdot, 1\rangle$ is a monoid.
(3) $0 \cdot a = a \cdot 0 = 0$ for all $a \in S$.
(4) $a \cdot (b + c) \sqsupseteq a \cdot b + a \cdot c$ and $(a + b) \cdot c \sqsupseteq a \cdot c + b \cdot c$ for all $a, b, c \in S$.
(5) The relation $\sqsubseteq := \{(a, b) \in S \times S \mid \exists d \in S : a + d = b\}$ is a partial order.
(6) For all *ω-chains* $(a_i)_{i\in\mathbb{N}}$ (i.e., $a_0 \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq \ldots$ with $a_i \in S$) $\sup^{\sqsubseteq}_{i\in\mathbb{N}} a_i$ exists. For *any* sequence $(b_i)_{i\in\mathbb{N}}$ define $\sum_{i\in\mathbb{N}} b_i := \sup^{\sqsubseteq}\{b_0 + b_1 + \cdots + b_i \mid i \in \mathbb{N}\}$.

Remark 8.2. We obtain the definition of subdistributive semiring from the definition of ω-continuous semiring by removing (7), and replacing distributivity with subdistributivity (see (4)).

In the rest of the section $\langle S, +, \cdot, 0, 1\rangle$ denotes a subdistributive semiring. Polynomials, vectors, differential, etc. are defined as in the distributive setting.

Note that the following inequalities still hold for all sequences $(a_i)_{i\in\mathbb{N}}$, $c \in S$, and partitions $(I_j)_{j\in J}$ of $\mathbb{N}$:

$$c\cdot\left(\sum_{i\in\mathbb{N}} a_i\right) \sqsupseteq \sum_{i\in\mathbb{N}}(c\cdot a_i), \quad \left(\sum_{i\in\mathbb{N}} a_i\right)\cdot c \sqsupseteq \sum_{i\in\mathbb{N}}(a_i\cdot c), \quad \sum_{j\in J}\left(\sum_{i\in I_j} a_j\right) \sqsupseteq \sum_{i\in\mathbb{N}} a_i\,.$$

Thus, any polynomial p is still monotone, although not necessarily ω-continuous. For any sequence $(\boldsymbol{v}_i)_{i\in\mathbb{N}}$ (of vectors) we still have $p(\sum_{i\in\mathbb{N}} \boldsymbol{v}_i) \sqsupseteq \sum_{i\in\mathbb{N}} p(\boldsymbol{v}_i)$. Hence, the Kleene sequence of a polynomial system $\boldsymbol{f}$ still converges, but not necessarily to the least fixed point of $\boldsymbol{f}$:

COROLLARY 8.3. *For any system $\boldsymbol{f}$ of polynomials, the Kleene sequence $(\boldsymbol{\kappa}^{(i)})_{i\in\mathbb{N}}$ is an ω-chain. Moreover, if $\boldsymbol{f}$ has a least solution $\mu\boldsymbol{f}$, then $\sup_{i\in\mathbb{N}} \boldsymbol{\kappa}^{(i)} \sqsubseteq \mu\boldsymbol{f}$.*

Since the Kleene sequence is still an ω-chain, its limit exists and is a safe approximation of the JOP$_0$-value:

PROPOSITION 8.4. *For any polynomial system $\boldsymbol{f}$ we have $(\boldsymbol{\kappa}^{(i)})_X \sqsupseteq Y(\mathcal{H}^i_X)$, and, hence, $(\sup_{i\in\mathbb{N}} \boldsymbol{\kappa}^{(i)})_X \sqsupseteq Y(\mathcal{T}_X)$ where $\mathcal{T}_X$ is the set of trees labeled with X.*

We skip the proof of this proposition as it is almost identical to the one of Proposition 6.4. The only difference is that when expanding the components of $\boldsymbol{\kappa}^{(i)}$ into a sum of products of coefficients, subdistributivity only guarantees that $\boldsymbol{\kappa}^{(i)}$ is

[9] We drop *ω-complete* in the following.

an upper bound, but not equality anymore. Similarly, subdistributivity only allows us to generalize the lower bound from Lemma 5.4, that is we have

$$\boldsymbol{f}(\boldsymbol{u}) + D\boldsymbol{f}|_{\boldsymbol{u}}(\boldsymbol{v}) \sqsubseteq \boldsymbol{f}(\boldsymbol{u} + \boldsymbol{v})$$

for a polynomial system $\boldsymbol{f}$ and vectors $\boldsymbol{u}, \boldsymbol{v}$.

We now turn to the definition of *Newton sequence*.

Definition 8.5. For $\boldsymbol{f}$ a polynomial system in the variables $\boldsymbol{X}$, and $\boldsymbol{a}, \boldsymbol{b}$ vectors we set

$$L_{\boldsymbol{f};\boldsymbol{a};\boldsymbol{b}}(\boldsymbol{X}) := \boldsymbol{b} + D\boldsymbol{f}|_{\boldsymbol{a}}(\boldsymbol{X}).$$

Definition 8.6. Let $\boldsymbol{f}$ be a polynomial system.

—Let $i \in \mathbb{N}$. An i-th *Newton approximant* $\boldsymbol{\nu}^{(i)}$ is inductively defined by

$$\boldsymbol{\nu}^{(0)} = \boldsymbol{f}(\boldsymbol{0}) \quad \text{and} \quad \boldsymbol{\nu}^{(i+1)} = \boldsymbol{\nu}^{(i)} + \boldsymbol{\Delta}^{(i)},$$

where $\boldsymbol{\Delta}^{(i)}$ has to satisfy $\sum_{k\in\mathbb{N}} D\boldsymbol{f}|^{k}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}) \sqsubseteq \boldsymbol{\Delta}^{(i)} \sqsubseteq L_{\boldsymbol{f};\boldsymbol{\nu}^{(i)};\boldsymbol{\delta}^{(i)}}\left(\boldsymbol{\Delta}^{(i)}\right)$.

—Any such sequence $(\boldsymbol{\nu}^{(i)})_{i\in\mathbb{N}}$ of Newton approximants is called *Newton sequence*.

Remark 8.7. If $\boldsymbol{\delta}^{(i)}$ exists, then possible choices for $\boldsymbol{\Delta}^{(i)}$ are

$$\sum_{k\in\mathbb{N}} D\boldsymbol{f}|^{k}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}),\ \sup_{k\in\mathbb{N}} L^{k}_{\boldsymbol{f};\boldsymbol{\nu}^{(i)};\boldsymbol{\delta}^{(i)}}(\boldsymbol{0}) \text{ or (if it exists) } \mu L_{\boldsymbol{f};\boldsymbol{\nu}^{(i)};\boldsymbol{\delta}^{(i)}}.$$

Note that in the distributive setting all three values coincide.

PROPOSITION 8.8. *Let $\boldsymbol{f}\colon V \to V$ be a vector of power series.*

—For every Newton approximant $\boldsymbol{\nu}^{(i)}$ there exists a vector $\boldsymbol{\delta}^{(i)}$ such that $\boldsymbol{f}(\boldsymbol{\nu}^{(i)}) = \boldsymbol{\nu}^{(i)} + \boldsymbol{\delta}^{(i)}$. So there is at least one Newton sequence.

—Every Newton sequence $\boldsymbol{\nu}^{(i)}$ satisfies $\boldsymbol{\kappa}^{(i)} \sqsubseteq \boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) \sqsubseteq \boldsymbol{\nu}^{(i+1)}$ for all $i \in \mathbb{N}$.

PROOF. First we prove for all $i \in \mathbb{N}$ that a suitable $\boldsymbol{\delta}^{(i)}$ exists and, at the same time, that the inequality $\boldsymbol{\kappa}^{(i)} \sqsubseteq \boldsymbol{\nu}^{(i)} \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)})$ holds. We proceed by induction on i. For the base case $i = 0$ we have:

$$\boldsymbol{\nu}^{(0)} = \boldsymbol{f}(\boldsymbol{0}) = \boldsymbol{\kappa}^{(0)} \sqsubseteq \boldsymbol{\kappa}^{(1)} = \boldsymbol{f}(\boldsymbol{\kappa}^{(0)}) = \boldsymbol{f}(\boldsymbol{\nu}^{(0)}).$$

So, there exists a $\boldsymbol{\delta}^{(0)}$ with $\boldsymbol{\nu}^{(0)} + \boldsymbol{\delta}^{(0)} = \boldsymbol{f}(\boldsymbol{\nu}^{(0)})$, and hence we have:

$$\boldsymbol{\nu}^{(1)} = \boldsymbol{\nu}^{(0)} + \boldsymbol{\Delta}^{(0)} \sqsupseteq \boldsymbol{\nu}^{(0)} + \sum_{k\in\mathbb{N}} D\boldsymbol{f}|^{k}_{\boldsymbol{\nu}^{(0)}}(\boldsymbol{\delta}^{(0)}) \sqsupseteq \boldsymbol{\nu}^{(0)} + \boldsymbol{\delta}^{(0)} = \boldsymbol{f}(\boldsymbol{\nu}^{(0)}).$$

For the induction step, let $i \geq 0$.

$$\boldsymbol{\kappa}^{(i+1)} = \boldsymbol{f}(\boldsymbol{\kappa}^{(i)}) \sqsubseteq \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) = \boldsymbol{\nu}^{(i)} + \boldsymbol{\delta}^{(i)} \sqsubseteq \boldsymbol{\nu}^{(i)} + \sum_{k\in\mathbb{N}} D\boldsymbol{f}|^{k}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}).$$

As we require that $\sum_{k\in\mathbb{N}} D\boldsymbol{f}|^{k}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)}) \sqsubseteq \boldsymbol{\Delta}^{(i)}$, it now immediately follows that

$$\boldsymbol{\kappa}^{(i+1)} \sqsubseteq \boldsymbol{\nu}^{(i)} + \boldsymbol{\Delta}^{(i)} = \boldsymbol{\nu}^{(i+1)}.$$

By definition of $\boldsymbol{\Delta}^{(i)}$ we have $\boldsymbol{\Delta}^{(i)} \sqsubseteq L_{\boldsymbol{f};\boldsymbol{\nu}^{(i)};\boldsymbol{\delta}^{(i)}}(\boldsymbol{\Delta}^{(i)})$, it therefore follows:

$$\begin{aligned}\boldsymbol{\nu}^{(i+1)} = \boldsymbol{\nu}^{(i)} + \boldsymbol{\Delta}^{(i)} &\sqsubseteq \boldsymbol{\nu}^{(i)} + \boldsymbol{\delta}^{(i)} + D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}\left(\boldsymbol{\Delta}^{(i)}\right) \\ &= \boldsymbol{f}(\boldsymbol{\nu}^{(i)}) + D\boldsymbol{f}|_{\boldsymbol{\nu}^{(i)}}\left(\boldsymbol{\Delta}^{(i)}\right) \sqsubseteq \boldsymbol{f}\left(\boldsymbol{\nu}^{(i)} + \boldsymbol{\Delta}^{(i)}\right) = \boldsymbol{f}(\boldsymbol{\nu}^{(i+1)}).\end{aligned}$$

We complete our proof by

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{\nu}^{(i+1)}) = \boldsymbol{\nu}^{(i+1)} + \boldsymbol{\delta}^{(i+1)} &\sqsubseteq \boldsymbol{\nu}^{(i+1)} + \sum_{k\in\mathbb{N}} D\boldsymbol{f}|^{k}_{\boldsymbol{\nu}^{(i+1)}}(\boldsymbol{\delta}^{(i+1)}) \\ &\sqsubseteq \boldsymbol{\nu}^{(i+1)} + \boldsymbol{\Delta}^{(i+1)} = \boldsymbol{\nu}^{(i+2)}. \quad \square\end{aligned}$$

PROPOSITION 8.9. *Let $\boldsymbol{M}$ be the JOP$_0$-value, that is, the vector $\boldsymbol{M}$ with $\boldsymbol{M}_X = Y(\mathcal{T}_X)$. Then, $\boldsymbol{M} \sqsubseteq \sup_{i\in\mathbb{N}} \boldsymbol{\kappa}^{(i)} \sqsubseteq \sup_{i\in\mathbb{N}} \boldsymbol{\nu}^{(i)}$.*

PROOF. Follows directly from Propositions 8.4 and 8.8. □

PROPOSITION 8.10. *For $\boldsymbol{\Delta}^{(i)} = \sum_{k\in\mathbb{N}} D\boldsymbol{f}|^{k}_{\boldsymbol{\nu}^{(i)}}(\boldsymbol{\delta}^{(i)})$, we have $\sup_{i\in\mathbb{N}} \boldsymbol{\nu}^{(i)} \sqsubseteq \mu\boldsymbol{f}$, if $\mu\boldsymbol{f}$ exists.*

PROOF. The proof is almost identical to the one of Proposition 5.1. Note that the proof of Lemma 5.5 does not use distributivity. □

THEOREM 8.11 (TREE CHARACTERIZATION OF THE NEWTON SEQUENCE). *Let $(\boldsymbol{\nu}^{(i)})_{i\in\mathbb{N}}$ be a Newton sequence of $\boldsymbol{f}$. For every $X \in \mathcal{X}$ and every $i \geq 0$ we have $\left(\boldsymbol{\nu}^{(i)}\right)_X \sqsupseteq Y(\mathcal{D}^i_X)$, that is, the X-component of the ith Newton approximant is a safe approximation of the yield of $\mathcal{D}^i_X$.*

PROOF. In the distributive setting, we proved this theorem via induction where we expanded the the terms we obtained using distributivity. In the subdistributive case, the same proof still guarantees that $\left(\boldsymbol{\nu}^{(i)}\right)_X \sqsupseteq Y(\mathcal{D}^i_X)$. □

9. *Conclusions*

Since its inception, the theory of program analysis has been based on two fundamental observations:

—Analysis problems can be reduced (using abstract interpretation [Cousot and Cousot 1977]) to the mathematical problem of computing the least solution of a system of equations over a semilattice.

—Such systems of equations can be solved using Kleene's fixed-point theorem as basic algorithm scheme.

In this article, we have contributed to both of these points. On the one hand, we generalize the algebraic setting from semilattices to arbitrary semirings (a generalization to idempotent semirings was already present in the work of Reps et al. [2005] on pushdown systems for program analysis). On the other hand, we obtain a new method for solving the dataflow equations by generalizing Newton's method to semirings.

The conceptually simple step from semilattices to semirings leads to a common algebraic setting for "qualitative" analyses (which, loosely speaking, explore the existence of execution paths satisfying a given property) and "quantitative" analyses (in which paths are assigned a numerical weight, and one is interested in the sum of the weights of all paths satisfying the property). Classical examples of qualitative analyses are live variables, constant propagation, or alias analysis, while examples of quantitative analysis arise in the study of probabilistic programs: probability of termination, expected execution time or, in the interprocedural case, expected stack height (for the latter, see Esparza et al. [2005] and Brázdil et al. [2005]). The common setting allows us to compare the algorithmic schemes used in the qualitative and quantitative case, and examine if a transfer of techniques is possible. We have shown that Newton's method can be generalized to the abstract setting. In particular, it can be applied to qualitative analysis problems.

We have explored Newton's method for idempotent semirings, that is, for the semirings corresponding to qualitative analyses. We have shown that the beautiful algebraic algorithm of Hopkins and Kozen [1999] for solving systems of equations over commutative Kleene algebras is a particular instance of Newton's method. Moreover, we have proved that the algorithm requires at most n iterations for a system of n equations, a tight bound that improves on the $\mathcal{O}(3^n)$ bound presented in Hopkins and Kozen [1999]. From a theoretical point of view, giving a purely algebraic proof of this fact along the lines of Hopkins and Kozen [1999] and Aceto et al. [2001] is an interesting challenge.

While this article imports notions of calculus and numerical mathematics into program analysis, our work also has some consequences pointing in the opposite direction. Quantitative analyses lead to systems of equations over the real semiring, a particular case of the systems over the real field. Surprisingly, the performance of Newton's method in this special case seems not to have received much attention from numerical mathematicians. The method turns out to have much better properties than in the general case. A consequence of our main result (which was already proved, in a slightly more restricted form, by Etessami and Yannakakis [2009]), is that on the real semiring Newton's method always converges to the least fixed point starting from zero. This is not so in the real field, where it may not converge or converge only locally, that is, when started sufficiently close to the zero (see, e.g., [Ortega 1972; Ortega and Rheinboldt 1970]). In related work, we have shown that the convergence order of the method is at least linear, meaning that the number of accurate bits of the Newton approximants grows at least linearly with the number of iterations [Kiefer et al. 2007; Esparza et al. 2008, 2010]

Appendix

A. *Proofs of Section 6*

To avoid typographical clutter in the following proofs, we use the following notation. Given some class of objects (e.g., derivation trees t) and a predicate $P(t)$, we write

$$\sum_t Y(t) : P(t)$$

instead of

$$\sum_{t \text{ such that } P(t) \text{ holds}} Y(t).$$

PROPOSITION 6.4. *$\left(\boldsymbol{\kappa}^{(i)}\right)_X = Y(\mathcal{H}^i_X)$, that is, the X-component of the ith Kleene approximant $\boldsymbol{\kappa}^{(i)}$ is equal to the yield of $\mathcal{H}^i_X$.*

PROOF. By induction on i. The base case $i = 0$ is easy. Induction step ($i \geq 0$):

$$
\begin{aligned}
&\left(\boldsymbol{\kappa}^{(i+1)}\right)_X \\
&\quad = \boldsymbol{f}_X(\boldsymbol{\kappa}^{(i)}) \\
&\quad = \sum_{j \in J} m_{X,j}(\boldsymbol{\kappa}^{(i)}) \\
&\quad = \sum_{j \in J} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots X_k a_{k+1} \\ y = a_1 \boldsymbol{\kappa}^{(i)}_{X_1} \cdots \boldsymbol{\kappa}^{(i)}_{X_k} a_{k+1} \end{cases}
\end{aligned}
$$

by induction:

$$
\begin{aligned}
&\quad = \sum_{j \in J} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots X_k a_{k+1} \\ y = a_1 Y(\mathcal{H}^i_{X_1}) \cdots Y(\mathcal{H}^i_{X_k}) a_{k+1} \end{cases} \\
&\quad = \sum_{\substack{j \in J \\ t_1, \ldots, t_k}} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots X_k a_{k+1} \\ t_1, \ldots, t_k \text{ trees with } h(t_r) \leq i, \lambda_v(t_r) = X_r \quad (1 \leq r \leq k) \\ y = a_1 Y(t_1) \cdots Y(t_k) a_{k+1} \end{cases} \\
&\quad = \sum_{j \in J, t} Y(t) : t \text{ is a tree with } h(t) \leq i+1,\ \lambda(t) = (X, j) \\
&\quad = Y(\mathcal{H}^i_X)
\end{aligned}
$$

□

The following definition of *fine dimension* is analogous to Definition 6.7, but adds a second component, which measures the length of the path from the root to the lowest node with the same dimension as the root:

Definition A.1 (*Fine Dimension*). The *fine dimension* $dl(t) = (d(t), l(t))$ of a tree t is inductively defined as follows:

(1) If t has no children, then $dl(t) = (0, 0)$.
(2) If t has exactly one child t_1, then $dl(t) = (d(t_1), l(t_1) + 1)$.
(3) If t has at least two children, let t_1, t_2 be two distinct children of t such that $d(t_1) \geq d(t_2)$ and $d(t_2) \geq d(t')$ for every child $t' \neq t_1$. Let $d_1 = d(t_1)$ and $d_2 = d(t_2)$. Then

$$
dl(t) = \begin{cases} (d_1 + 1, 0) & \text{if } d_1 = d_2 \\ (d_1, l(t_1) + 1) & \text{if } d_1 > d_2. \end{cases}
$$

Remark A.2. Notice that, by Definition 6.9, a tree t is proper if and only if $l(t) = 0$. So we have:

$$
Y(P^i_X) = \sum_t Y(t) : t \text{ tree with } \lambda_v(t) = X,\ dl(t) = (i, 0)
$$

Now we can prove the remaining lemmata from Section 6.

LEMMA 6.10. *For every variable $X \in \mathcal{X}$ and every $i \geq 0$: $\boldsymbol{\tau}_X^{(i)} = Y(\mathcal{D}_X^i)$.*

PROOF. By induction on i. Induction base ($i = 0$):

$$\begin{aligned}\boldsymbol{\tau}_X^{(0)} = \boldsymbol{f}_X(\mathbf{0}) &= \sum_t Y(t) : \lambda_v(t) = X, h(t) = 0 \\ &= \sum_t Y(t) : \lambda_v(t) = X, d(t) = 0 \\ &= Y(\mathcal{D}_X^0)\end{aligned}$$

Induction step ($i + 1 > 0$): We need to show that $D\boldsymbol{f}|^*_{\boldsymbol{\tau}^{(i)}}(\boldsymbol{\delta}^{(i)})$ equals exactly the yield of all trees of dimension $i + 1$, that is, that for all $X \in \mathcal{X}$

$$\left(D\boldsymbol{f}|^*_{\boldsymbol{\tau}^{(i)}}(\boldsymbol{\delta}^{(i)})\right)_X = \sum_t Y(t) : \lambda_v(t) = X,\ d(t) = i + 1.$$

We prove the following stronger claim by induction on p:

$$\left(D\boldsymbol{f}|^p_{\boldsymbol{\tau}^{(i)}}(\boldsymbol{\delta}^{(i)})\right)_X = \sum_t Y(t) : \lambda_v(t) = X,\ dl(t) = (i + 1, p)$$

The claim holds for $p = 0$ by Remark A.2. For the induction step, let $p \geq 0$. Then we have for all $X \in \mathcal{X}$:

$$\begin{aligned}&\left(D\boldsymbol{f}|^{p+1}_{\boldsymbol{\tau}^{(i)}}(\boldsymbol{\delta}^{(i)})\right)_X \\ &\quad = \left(D\boldsymbol{f}|_{\boldsymbol{\tau}^{(i)}} \circ D\boldsymbol{f}|^p_{\boldsymbol{\tau}^{(i)}}(\boldsymbol{\delta}^{(i)})\right)_X \\ &\quad = D\boldsymbol{f}_X|_{\boldsymbol{\tau}^{(i)}} \circ D\boldsymbol{f}|^p_{\boldsymbol{\tau}^{(i)}}(\boldsymbol{\delta}^{(i)})\end{aligned}$$

Define the vector $\widetilde{\boldsymbol{Y}}$ by $\widetilde{\boldsymbol{Y}}_{X_0} = \sum_t Y(t) : \lambda_v(t) = X_0, dl(t) = (i + 1, p)$. Then, by induction hypothesis (on p), above expression equals

$$\begin{aligned}&= D\boldsymbol{f}_X|_{\boldsymbol{\tau}^{(i)}}(\widetilde{\boldsymbol{Y}}) \\ &= \sum_{j \in J} Dm_{X,j}|_{\boldsymbol{\tau}^{(i)}}(\widetilde{\boldsymbol{Y}}) : m_{X,j} = a_1 X_1 \cdots a_k X_k a_{k+1} \\ &= \sum_{j \in J, r} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots a_k X_k a_{k+1} \\ 1 \leq r \leq k \\ y = a_1 \boldsymbol{\tau}_{X_1}^{(i)} \cdots a_r \widetilde{\boldsymbol{Y}}_{X_r} a_{r+1} \boldsymbol{\tau}_{X_{r+1}}^{(i)} \cdots a_k \boldsymbol{\tau}_{X_k}^{(i)} a_{k+1} \end{cases}\end{aligned}$$

by induction on i:

$$= \sum_{\substack{j\in J,r,\\ t_1,\ldots,t_k}} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots a_k X_k a_{k+1} \\ 1 \le r \le k \\ t_1, \ldots, t_k \text{ trees with } \lambda_v(t_s) = X_s \quad (1 \le s \le k) \\ \qquad dl(t_r) = (i+1, p), \\ \qquad d(t_s) \le i \quad (1 \le s \le k,\ s \ne r) \\ y = a_1 Y(t_1) \cdots a_r Y(t_r) \cdots a_k Y(t_k) a_{k+1} \end{cases}$$

$$= \sum_{j\in J,t} Y(t) : t \text{ tree with } \lambda(t) = (X, j),\ dl(t) = (i+1, p+1)$$

$$= \sum_{t} Y(t) : t \text{ tree with } \lambda_v(t) = X,\ dl(t) = (i+1, p+1) \qquad \square$$

LEMMA 6.11. *The sequence* $(\boldsymbol{\tau}^{(i)})_{i\in\mathbb{N}}$ *is a Newton sequence as defined in Definition* 3.6*, that is, the* $\boldsymbol{\delta}^{(i)}$ *of Definition* 6.9 *satisfy* $\boldsymbol{f}(\boldsymbol{\tau}^{(i)}) = \boldsymbol{\tau}^{(i)} + \boldsymbol{\delta}^{(i)}$.

PROOF.

$$\boldsymbol{f}_X(\boldsymbol{\tau}^{(i)}) = \sum_{j\in J} m_{X,j}(\boldsymbol{\tau}^{(i)})$$

$$= \sum_{j\in J} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots a_k X_k a_{k+1} \\ y = a_1 \boldsymbol{\tau}^{(i)}_{X_1} \cdots a_k \boldsymbol{\tau}^{(i)}_{X_k} a_{k+1} \end{cases}$$

by Lemma 6.10:

$$= \sum_{\substack{j\in J\\ t_1,\ldots,t_k}} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots a_k X_k a_{k+1} \\ t_1, \ldots, t_k \text{ trees with } \lambda_v(t_r) = X_r,\ d(t_r) \le i, \quad (1 \le r \le k) \\ y = a_1 Y(t_1) \cdots a_k Y(t_k) a_{k+1} \end{cases}$$

$$= \sum_{\substack{j\in J\\ t_1,\ldots,t_k}} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots a_k X_k a_{k+1} \\ t_1, \ldots, t_k \text{ trees with } \lambda_v(t_r) = X_r,\ d(t_r) \le i, \quad (1 \le r \le k) \\ \qquad \text{such that at most one of the } t_r \text{ with } d(t_r) = i \\ y = a_1 Y(t_1) \cdots a_k Y(t_k) a_{k+1} \end{cases}$$

$$+ \sum_{\substack{j\in J\\ t_1,\ldots,t_k}} y : \begin{cases} m_{X,j} = a_1 X_1 \cdots a_k X_k a_{k+1} \\ t_1, \ldots, t_k \text{ trees with } \lambda_v(t_r) = X_r,\ d(t_r) \le i, \quad (1 \le r \le k) \\ \qquad \text{such that at least two of the } t_r \text{ with } d(t_r) = i \\ y = a_1 Y(t_1) \cdots a_k Y(t_k) a_{k+1} \end{cases}$$

$$= \sum_{t} Y(t) : t \text{ tree with } \lambda_v(t) = X,\ d(t) \le i$$

$$+ \sum_{t} Y(t) : t \text{ tree with } \lambda_v(t) = X,\ dl(t) = (i+1, 0)$$

by Lemma 6.10 respectively, Remark A.2:

$$
\begin{aligned}
&= \boldsymbol{\tau}_X^{(i)} + Y(P_X^{i+1}) \\
&= \boldsymbol{\tau}_X^{(i)} + \boldsymbol{\delta}_X^{(i)} \qquad \square
\end{aligned}
$$

ACKNOWLEDGMENT. We thank Helmut Seidl and the anonymous referees for helpful suggestions and remarks.

REFERENCES

ACETO, L., ÉSIK, Z., AND INGÓLFSDÓTTIR, A. 2001. A fully equational proof of Parikh's theorem. *RAIRO, Theoretical Informatics and Applications 36*, 200–2.

BRÁZDIL, T., ESPARZA, J., AND KUCERA, A. 2005. Analysis and prediction of the long-run behavior of probabilistic sequential programs with recursion. In *Proceedings of Symposium on Foundations of Computer Science*. IEEE, 521–530.

BRZOZOWSKI, J. A. 1964. Derivatives of regular expressions. *J. ACM 11,* 4, 481–494.

COUSOT, P., AND COUSOT, R. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of Symposium on Principles of Programming Languages*. ACM, 238–252.

DEUTSCH, A. 1994. Interprocedural may-alias analysis for pointers: Beyond *k*-limiting. In *Proceedings of Conference on Programming Language Design and Implementation*. 230–241.

ESPARZA, J., KIEFER, S., AND LUTTENBERGER, M. 2007a. An extension of Newton's method to ω-continuous semirings. In *Proceedings of Conference on Developments in Language Theory*. LNCS 4588. Springer, 157–168.

ESPARZA, J., KIEFER, S., AND LUTTENBERGER, M. 2007b. On fixed point equations over commutative semirings. In *Proceedings of Symposium on Theoretical Aspects of Computer Science*. LNCS 4397. Springer, 296–307.

ESPARZA, J., KIEFER, S., AND LUTTENBERGER, M. 2008. Convergence thresholds of Newton's method for monotone polynomial equations. In *Proceedings of Symposium on Theoretical Aspects of Computer Science*. 289–300.

ESPARZA, J., KIEFER, S., AND LUTTENBERGER, M. 2010. Computing the least fixed point of positive polynomial systems. *SIAM J. Comput.*. To appear.

ESPARZA, J., KUČERA, A., AND MAYR, R. 2005. Quantitative analysis of probabilistic pushdown automata: Expectations and variances. In *Proceedings of Symposium on Logic in Computer Science*. IEEE, 117–126.

ESPARZA, J., KUČERA, A., AND MAYR, R. 2004. Model checking probabilistic pushdown automata. In *Proceedings of Symposium on Logic in Computer Science*. IEEE, 12–21.

ETESSAMI, K., AND YANNAKAKIS, M. 2009. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM 56,* 1, 1–66.

HOPKINS, M. W., AND KOZEN, D. 1999. Parikh's theorem in commutative Kleene algebra. In *Proceedings of Symposium on Logic in Computer Science*. 394–401.

JONES, N., AND MUCHNICK, S. 1982. A flexible approach to interprocedural data flow analysis and programs with recursive data structures. In *Proceedings of Symposium on Principles of programming Languages*. ACM, 66–74.

KAM, J. B., AND ULLMAN, J. D. 1977. Monotone data flow analysis frameworks. *Acta Inf. 7*, 305–317.

KIEFER, S., LUTTENBERGER, M., AND ESPARZA, J. 2007. On the convergence of Newton's method for monotone systems of polynomial equations. In *Proceedings of Symposium on Theory of Computing*. ACM, 217–226.

KILDALL, G. A. 1973. A unified approach to global program optimization. In *Proceedings of Symposium on Principles of Programming Languages*. ACM, 194–206.

KNOOP, J. AND STEFFEN, B. 1992. The interprocedural coincidence theorem. In *Proceedings of the International Conference on Compiler Construction*. LNCS, vol. 641. Springer-Verlag, 125–140.

KUICH, W. 1997. *Handbook of Formal Languages*. Vol. 1. Springer, Chapter 9: Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata, 609–677.

NIELSON, F., NIELSON, H., AND HANKIN, C. 1999. *Principles of Program Analysis*. Springer.

ORTEGA, J. 1972. *Numerical Analysis: A Second Course*. Academic Press, New York.

ORTEGA, J., AND RHEINBOLDT, W. 1970. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press.

REPS, T., HORWITZ, S., AND SAGIV, M. 1995. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of Symposium on Principles of Programming Languages*. ACM, 49–61.

REPS, T., SCHWOON, S., JHA, S., AND MELSKI, D. 2005. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Prog. 58,* 1–2 (October), 206–263. Special Issue on the Static Analysis Symposium 2003.

SAGIV, S., REPS, T. W., AND HORWITZ, S. 1996. Precise interprocedural dataflow analysis with applications to constant propagation. *Theoret. Comput. Sci. 167,* 1&2, 131–170.

SEIDL, H., AND FECHT, C. 2000. Interprocedural analyses: A comparison. *J. Log. Prog. 43*, 123–156.

SHARIR, M., AND PNUELI, A. 1981. *Program Flow Analysis: Theory and Applications*. Prentice-Hall, Chapter 7: Two Approaches to Interprocedural Data Flow Analysis, 189–233.

RECEIVED NOVEMBER 2008; REVISED SEPTEMBER 2009 AND MARCH 2010; ACCEPTED MARCH 2010