Newtonian Program Analysis via Tensor Product *

Thomas Reps[†,‡], Emma Turetsky[‡], and Prathmesh Prabhu[§]

[†]Univ. of Wisconsin; Madison, WI; USA
[‡]GrammaTech, Inc.; Ithaca, NY; USA
[§]Google, Inc.; Mountain View, CA; USA

Abstract

Recently, Esparza et al. generalized Newton's method—a numerical-analysis algorithm for finding roots of real-valued functions—to a method for finding fixed-points of systems of equations over semirings. Their method provides a new way to solve interprocedural dataflow-analysis problems. As in its real-valued counterpart, each iteration of their method solves a simpler "linearized" problem.

One of the reasons this advance is exciting is that some numerical analysts have claimed that "'all' effective and fast iterative [numerical] methods are forms (perhaps very disguised) of Newton's method." However, there is an important difference between the dataflow-analysis and numerical-analysis contexts: when Newton's method is used on numerical-analysis problems, multiplicative commutativity is relied on to rearrange expressions of the form "$c * X + X * d$" into "$(c + d) * X$." Such equations correspond to path problems described by regular languages. In contrast, when Newton's method is used for interprocedural dataflow analysis, the "multiplication" operation involves function composition, and hence is non-commutative: "$c * X + X * d$" cannot be rearranged into "$(c + d) * X$." Such equations correspond to path problems described by linear context-free languages (LCFLs).

In this paper, we present an improved technique for solving the LCFL sub-problems produced during successive rounds of Newton's method. Our method applies to predicate abstraction, on which most of today's software model checkers rely.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification—*Formal methods*; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—*Program analysis*; F.4.2 [*Mathematical Logic and Formal Languages*]: Grammars and Other Rewriting Systems—Grammar types; F.4.3 [*Mathematical Logic and Formal Languages*]: Formal Languages—*Algebraic language theory*

General Terms Algorithms, Languages, Theory, Verification

Keywords Newton's method, polynomial fixed-point equation, interprocedural program analysis, semiring, regular expression, tensor product

1. Introduction

Many interprocedural dataflow-analysis problems can be formulated as the problem of finding the least fixed-point of a system of equations $\vec{X} = \vec{f}(\vec{X})$ over a semiring [2, 23, 24]. Standard methods for obtaining the solution to such an equation system are based on ***Kleene iteration***, a successive-approximation method defined as follows:

$$\boxed{\begin{array}{rcl}\vec{\kappa}^{(0)} & = & \vec{\bot} \\ \vec{\kappa}^{(i+i)} & = & \vec{f}(\vec{\kappa}^{(i)})\end{array}} \tag{1}$$

Recently, Esparza et al. [6, 7] generalized Newton's method—a numerical-analysis algorithm for finding roots of real-valued functions—to a method for finding fixed-points of systems of equations over semirings. Their method, ***Newtonian Program Analysis (NPA)***, is also an iterative successive-approximation method, but uses the following scheme:[1]

$$\boxed{\begin{array}{rcl}\vec{\nu}^{(0)} & = & \vec{\bot} \\ \vec{\nu}^{(i+1)} & = & \vec{f}(\vec{\nu}^{(i)}) \sqcup \text{LinearCorrectionTerm}(\vec{f}, \vec{\nu}^{(i)})\end{array}} \tag{2}$$

where $\text{LinearCorrectionTerm}(\vec{f}, \vec{\nu}^{(i)})$ is a correction term—a function of $\vec{f}$ and the current approximation $\vec{\nu}^{(i)}$—that nudges the next approximation $\vec{\nu}^{(i+1)}$ in the right direction at each step. The sense in which the correction term is "linear" will be discussed in §2, but it is that linearity property that makes it proper to say that Eqn. (2) is a form of Newton's method.

NPA holds considerable promise for creating faster solvers for interprocedural dataflow analysis. Most dataflow-analysis algorithms use classical fixed-point iteration (typically worklist-based

* Supported, in part, by a gift from Rajiv and Ritu Batra; by NSF under grant CCF-0904371; by ONR under grants N00014-{09-1-0510, 11-C-0447}; by DARPA under cooperative agreement HR0011-12-2-0012; by ARL under grant W911NF-09-1-0413; by AFRL under grant FA9550-09-1-0279, DARPA CRASH award FA8650-10-C-7088, DARPA MUSE award FA8750-14-2-0270, and DARPA STAC award FA8750-15-C-0082; and by the UW-Madison Office of the Vice Chancellor for Research and Graduate Education with funding from the Wisconsin Alumni Research Foundation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the sponsoring agencies. T. Reps has an ownership interest in GrammaTech, Inc., which has licensed elements of the technology reported in this publication. When the research reported in the paper was carried out, E. Turetsky was affiliated with the Univ. of Wisconsin.

POPL'16, January 20–22, 2016, St. Petersburg, FL, USA
© 2016 ACM. 978-1-4503-3549-2/16/01...$15.00
http://dx.doi.org/10.1145/2837614.2837659

[1] For reasons that are immaterial to this discussion, Esparza et al. start the iteration via $\vec{\nu}^{(0)} = \langle f_1(\bot), \ldots, f_n(\bot)\rangle$ rather than $\vec{\nu}^{(0)} = \vec{\bot}$. Our goal here is to bring out the essential similarities between Eqns. (1) and (2).

"chaotic-iteration"). In contrast, the workhorse for fast numerical-analysis algorithms is Newton's method, which usually converges much faster than classical fixed-point iteration.[2] In fact, Tapia and Dennis [28] have claimed that

> 'All' effective and fast iterative [numerical] methods are forms (perhaps very disguised) of Newton's method.

Can a similar claim be made about methods for solving equations over semirings? As a first step toward an answer, it is important to discover the best approaches for creating NPA-based solvers.

Like its real-valued counterpart, NPA is an iterative method: each iteration solves a simpler "linearized" problem that is generated from the original equation system. At first glance, one might think that solving each linearized problem corresponds to solving an ***intra***procedural dataflow-analysis problem—a topic that has a fifty-year history [9, 11, 13, 29, 32]. Unfortunately, this idea does not hold up to closer scrutiny. In particular, ***the sub-problems generated by NPA lie outside the class of problems that an intraprocedural dataflow analyzer handles***, for a reason we now explain.

When Newton's method is used in numerical-analysis problems, ***commutativity of multiplication*** is relied on to rearrange an expression of the form "$c * X + X * d$" in the linearized problem into one of the form "$c * X + d * X$," which equals "$(c + d) * X$." In contrast, in interprocedural dataflow analysis, a dataflow value is typically an abstract transformer (i.e., it represents a function from sets of states to sets of states) [5, 26]. Consequently, the "multiplication" operation is typically the reversal of function composition—$v_1 * v_2 \stackrel{\text{def}}{=} v_2 \circ v_1$—which is not a commutative operation. When NPA is used with a non-commutative semiring, an expression "$c*X+X*d$" in the linearized problem cannot be rearranged: coefficients can appear on both sides of variables.

From a formal-languages perspective, the linearized equation systems that arise in numerical analysis correspond to path problems described by ***regular languages***. However, when expressions of the form "$c * X + X * d$" cannot be rearranged, the linearized equation systems correspond to path problems described by ***linear context-free languages (LCFLs)***. Conventional intraprocedural dataflow-analysis algorithms solve only regular-language path problems, and hence cannot, in general, be applied to the linearized equation systems considered on each round of NPA. Consequently, we are stuck performing classical fixed-point iteration on the LCFL equation systems. (Applying NPA's linearization transformation to one of the LCFL equation systems just results in the same LCFL equation system, and so one would not make any progress.)

A preliminary study that we did indicated that (i) NPA was not an improvement over conventional methods for interprocedural dataflow analysis, and (ii) 98% of the time was spent performing classical fixed-point iteration to solve the LCFL equation systems. If only we could apply a fast intraprocedural solver! In particular, Tarjan's path-expression method [30] finds a regular expression for each of the variables in a set of mutually recursive left-linear equations. The regular expressions are then evaluated using an appropriate interpretation of the regular operators $+$, $\cdot$, and *.

On the face of it, it seems impossible that our wish could be fulfilled. Formal-language theory tells us that LCFL $\supsetneq$ Regular. In particular, the canonical example of a non-regular language, $\{b^i c^i \mid i \in \mathbb{N}\}$, is an LCFL. ***However, despite this obstacle—and this is where the surprise value of our work lies—there are non-commutative semirings for which we can transform the problem so that Tarjan's method applies*** (§4.5). Moreover, as discussed in §5, one of the families of semirings for which our transformation applies is the set of ***predicate-abstraction domains*** [8], which are the foundation of most of today's software model checkers.[3]

Contributions. The paper's contributions include the following:

- We show how to improve the performance of NPA for certain classes of interprocedural dataflow-analysis problems. The paper presents Newtonian Program Analysis via Tensor Products (NPA-TP), a procedure for solving systems of mutually recursive equations over certain classes of non-commutative semirings (§4.5 and §7).
- NPA-TP sidesteps the issue "LCFL $\supsetneq$ Regular" as follows (§4):
  - We require semiring $\mathcal{S}$ to possess a ***tensor-product operation*** (Defn. 4.1). The special properties of this operation allow each LCFL problem to be transformed into a ***left-linear***—and hence ***regular***—system of equations over a ***different*** semiring $\mathcal{S}_\mathcal{T}$ (§4.2).
  - The $\mathcal{S}_\mathcal{T}$ equation system can be solved quickly using a fast intraprocedural solver—in particular, Tarjan's method for finding and evaluating path expressions.
  - The desired $\mathcal{S}$ answer can be read out of the $\mathcal{S}_\mathcal{T}$ answer.
  - This sequence of steps does not create any loss of precision.
- We describe how to apply NPA-TP to predicate-abstraction problems (§5).
- We describe a new way for loops to be handled in NPA and NPA-TP (§6).
- We describe how to extend NPA and NPA-TP to analyze programs with local variables (§8).
- We present the results of experiments with an implementation of NPA-TP for sequential Boolean programs (§9).

§10 discusses related work. §11 draws some conclusions.

2. Background

Semirings.

DEFINITION 2.1. *A **semiring** $\mathcal{S} = (D, \oplus, \otimes, \underline{0}, \underline{1})$ consists of a set of **elements** D equipped with two binary operations: **combine** ($\oplus$) and **extend** ($\otimes$). $\oplus$ and $\otimes$ are associative, and have identity elements $\underline{0}$ and $\underline{1}$, respectively. $\oplus$ is commutative, and $\otimes$ distributes over $\oplus$. (A semiring is sometimes called a **weight domain**, in which case elements are called **weights**.)*

*An ω-**continuous semiring** is a semiring with the following additional properties:*

1. *The relation $\sqsubseteq \stackrel{\text{def}}{=} \{(a, b) \in D \times D \mid \exists d : a \oplus d = b\}$ is a partial order.*
2. *Every ω-chain $(a_i)_{i\in\mathbb{N}}$ (i.e., for all $i \in \mathbb{N}$ $a_i \sqsubseteq a_{i+1}$) has a supremum with respect to $\sqsubseteq$, denoted by $\sup_{i\in\mathbb{N}} a_i$.*
3. *Given an arbitrary sequence $(c_i)_{i\in\mathbb{N}}$, define*

$$\bigoplus_{i\in\mathbb{N}} c_i \stackrel{\text{def}}{=} \sup\{c_0 \oplus c_1 \oplus \ldots \oplus c_i \mid i \in \mathbb{N}\}.$$

The supremum exists by (2) above. Then, for every sequence $(a_i)_{i\in\mathbb{N}}$, for every $b \in \mathcal{S}$, and every partition $(I_j)_{j\in J}$ of $\mathbb{N}$, the following properties all hold:

$$b \otimes \left(\bigoplus_{i\in\mathbb{N}} a_i\right) = \bigoplus_{i\in\mathbb{N}} (b \otimes a_i)$$

$$\left(\bigoplus_{i\in\mathbb{N}} a_i\right) \otimes b = \bigoplus_{i\in\mathbb{N}} (a_i \otimes b)$$

$$\bigoplus_{j\in J} \left(\bigoplus_{i\in I_j} a_i\right) = \bigoplus_{i\in\mathbb{N}} a_i$$

[2] For some inputs, Newton's method may converge slowly, converge only when started at a point close to the desired root, or not converge at all; however, when it does converge to a solution, it usually converges much faster than classical fixed-point iteration.

[3] Two other classes of semirings for which the transformation applies are based on abstract domains of affine relations [21, 22]; see [18, §6.2].

*The notation a^i denotes the i^{th} term in the sequence in which $a^0 = \underline{1}$ and $a^{i+1} = a^i \otimes a$. An ω-continuous semiring has a **Kleene-star operator** $^*: D \to D$ defined as follows: $a^* = \bigoplus_{i \in \mathbb{N}} a^i$.*

The set of all binary relations on a given finite set forms a semiring, and allows each predicate-abstraction domain to be formalized as a semiring.

DEFINITION 2.2. *If A is a finite set, then the **relational weight domain** on A is defined as $(2^{A \times A}, \cup, ;, \emptyset, id)$: weights are binary relations on A, $\oplus$ is union, $\otimes$ is relational composition, $\underline{0}$ is the empty relation, and $\underline{1}$ is the identity relation on A. The Kleene-star operation is reflexive transitive closure.*

A Boolean program is a program whose only datatype is Boolean. A Boolean program can be used as an abstraction of a real-world program [1] using predicate abstraction [8]. By instantiating A to be the set of global states of a Boolean program P, we obtain a semiring that can encode the state-transformers of P: the semiring value associated with an assignment or assume statement `st` of P is the binary relation on A that represents the effect of `st` on the global state of P.

In this paper, the focus is on semirings in which $\oplus$ is idempotent (i.e., for all $a \in D$, $a \oplus a = a$). In an idempotent semiring, the order on elements is defined by $a \sqsubseteq b$ iff $a \oplus b = b$. (Idempotence would be expected in the context of dataflow analysis because an idempotent semiring is a join semilattice $(D, \oplus)$ in which the join operation is $\oplus$.)

A semiring is ***commutative*** if for all $a, b \in D$, $a \otimes b = b \otimes a$. We work with ***non-commutative*** semirings, and henceforth use the term "semiring"—and symbol $\mathcal{S}$—to mean an ***idempotent, non-commutative, ω-continuous semiring***.

To simplify notation, we sometimes abbreviate $a \otimes b$ as ab, and we assume the following precedences for operators: $^* > \otimes > \oplus$. We also sometimes use $a \in \mathcal{S}$ rather than $a \in D$.

Remark. In general, we do not make a typographical distinction between uses of *, $\otimes$, and $\oplus$ as *syntactic* symbols in expressions that are constructed, and their *semantic* counterparts. The semantic operators are interpreted in $\mathcal{S}$, and must possess the various properties given in Defn. 2.1 and the text above. In one place it is useful to make such a distinction (Defn. 4.5), and there we denote the semantic operators by $(\!|^*|\!)$, $(\!|\otimes|\!)$, and $(\!|\oplus|\!)$, respectively. □

Newtonian Program Analysis (NPA). Esparza et al. [6, 7] have given a generalization of Newton's method that finds the least fixed-point of a system of equations over a semiring. In this section, we summarize their NPA method for the case of idempotent, non-commutative, ω-continuous semirings.

EXAMPLE 2.3. *Consider the following program scheme, where X_1 represents the main procedure, X_2 represents a subroutine, and s_a, s_b, s_c, and s_d represent four program statements:*

```
X1() {                X2() {
  sa;                   if (⋆) sd
  X2()                  else {
}                         sb; X2(); X2(); sc
                        }
                      }
```

Suppose that we have a semiring that captures a suitable abstraction of the program's actions (such as the relational weight domain). Let a, b, c, and d denote the semiring elements that abstract statements s_a, s_b, s_c, and s_d, respectively. The (abstract) actions of procedures X_1 and X_2 can be expressed as the following set of recursive equations:

$$X_1 = a \otimes X_2 \qquad X_2 = d \oplus b \otimes X_2 \otimes X_2 \otimes c. \tag{3}$$

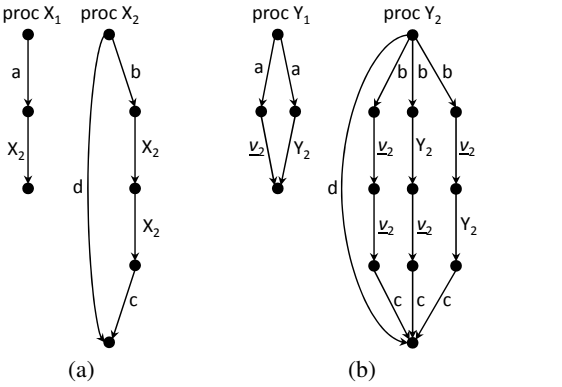


Figure 1. (a) Graphical depiction of the equation system given in Eqn. (3) as an interprocedural control-flow graph. The three edges labeled "X_2" represent calls to procedure X_2. (b) Linearized equation system over $\vec{Y}$ obtained from Eqn. (3) via Eqn. (5).

An equation system can also be viewed as a representation of a program's interprocedural control-flow graph (CFG). See Fig. 1(a).

In general, let $\mathcal{S} = (D, \oplus, \otimes, \underline{0}, \underline{1})$ be a semiring and $a_1, \ldots, a_{k+1} \in D$ be semiring elements. Let $\mathcal{X}$ be a finite set of variables $X_1, \ldots, X_k$. A ***monomial*** is a finite expression $a_1 X_1 a_2 \ldots a_k X_k a_{k+1}$, where $k \geq 0$. Monomials of the form $X_1 a_2$, $a_1 X_1$, and $a_1 X_1 a_2$ are ***left-linear***, ***right-linear***, and ***linear***, respectively. (A semiring constant a_1 is considered to be left-linear, right-linear, and linear.) A ***polynomial*** is a finite expression of the form $m_1 \oplus \ldots \oplus m_p$, where $p \geq 1$ and $m_1, \ldots, m_p$ are monomials. A system of polynomial equations has the form

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

or equivalently, $\vec{X} = \vec{f}(\vec{X})$, where $\vec{X} = \langle X_1, \ldots, X_n \rangle$ and $\vec{f} = \lambda \vec{X}.\langle f_1(\vec{X}), \ldots, f_n(\vec{X}) \rangle$. For instance, for Eqn. (3), $\vec{f} \stackrel{\text{def}}{=} \lambda \vec{X}.\langle a \otimes X_2, d \oplus b \otimes X_2 \otimes X_2 \otimes c \rangle$.

Kleene iteration is the well-known technique for finding the least fixed-point of $\vec{X} = \vec{f}(\vec{X})$ via the sequence $\vec{\kappa}^{(0)} = \underline{0}$; $\vec{\kappa}^{(i+i)} = \vec{f}(\vec{\kappa}^{(i)})$. Esparza et al. [6, 7] devised an alternative method, called NPA, for finding the least fixed-point of $\vec{X} = \vec{f}(\vec{X})$. With NPA, one solves the following sequence of problems for $\vec{\nu}$:

$$\boxed{\begin{array}{l} \vec{\nu}^{(0)} = (f_1(\underline{0}), \ldots, f_n(\underline{0})) \\ \vec{\nu}^{(i+1)} = \vec{Y}^{(i)} \end{array}} \tag{4}$$

where $\vec{Y}^{(i)}$ is the value of $\vec{Y}$ in the least solution of

$$\boxed{\vec{Y} = \vec{f}(\vec{\nu}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y})} \tag{5}$$

and $\mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y})$ is the ***multivariate differential*** of $\vec{f}$ at $\vec{\nu}^{(i)}$, defined below (see Defn. 2.4). As discussed in §1, Eqns. (4) and (5) resemble Kleene iteration, except that on each iteration $\vec{f}(\vec{\nu}^{(i)})$ is "corrected" by the amount $\mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y})$.[4]

[4] Esparza et al. also show that if Eqn. (5) is changed to

$$\vec{Y} = \vec{f}(\underline{0}) \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y}), \tag{6}$$

the combinations Eqns. (4) and (5) and Eqns. (4) and (6) produce the same set of iterates $\vec{\nu}^{(0)}, \vec{\nu}^{(1)}, \ldots, \vec{\nu}^{(i)}, \ldots$ [7, Prop. 7.1]. Eqn. (5) has the benefit of presenting NPA as a Kleene-like iteration, during which a linear correction is performed on each round, which provides better intuition about the connections with Newton's method for numerical analysis. Our implementation, however, is based on Eqn. (6).

There is a close analogy between NPA and the use of Newton's method in numerical analysis to solve a system of polynomial equations $\vec{f}(\vec{X}) = \vec{0}$. In both cases, one creates a linear approximation of $\vec{f}$ around the point $(\vec{\nu}^{(i)}, \vec{f}(\vec{\nu}^{(i)}))$, and then uses the solution of the linear system in the next approximation of $\vec{X}$. The sequence $\vec{\nu}^{(0)}, \vec{\nu}^{(1)}, \ldots, \vec{\nu}^{(i)}, \ldots$ is called the ***Newton sequence*** for $\vec{X} = \vec{f}(\vec{X})$. The process of solving Eqns. (4) and (5) for $\vec{\nu}^{(i+1)}$, given $\vec{\nu}^{(i)}$, is called a ***Newton step*** or one ***Newton round***. For polynomial equations over a semiring, the linear approximation of $\vec{f}$ is created as follows:

DEFINITION 2.4. *[6, 7] Let $f_i(\vec{X})$ be a component function of $\vec{f}(\vec{X})$. The **differential** of $f_i(\vec{X})$ with respect to X_j at $\underline{\vec{\nu}}$, denoted by $\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y})$, is defined as follows:*

$$\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y}) = \begin{cases} \underline{0} & \text{if } f_i = s \in \mathcal{S} \\ \underline{0} & \text{if } f_i = X_k \text{ and } k \neq j \\ Y_j & \text{if } f_i = X_j \\ \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \oplus \mathcal{D}_{X_j} h|_{\underline{\vec{\nu}}}(\vec{Y}) & \text{if } f_i = g \oplus h \\ \begin{pmatrix} \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\ g(\underline{\vec{\nu}}) \otimes \mathcal{D}_{X_j} h|_{\underline{\vec{\nu}}}(\vec{Y}) \end{pmatrix} & \text{if } f_i = g \otimes h \end{cases} \tag{7}$$

*Let $\vec{f}$ be a multivariate polynomial function defined by $\vec{f} \stackrel{\text{def}}{=} \lambda \vec{X}.(f_1(\vec{X}), \ldots, f_n(\vec{X}))$. The **multivariate differential** of $\vec{f}$ at $\underline{\vec{\nu}}$, denoted by $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$, is defined as follows:*

$$\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y}) = \left\langle \begin{matrix} \mathcal{D}_{X_1} f_1|_{\underline{\vec{\nu}}}(\vec{Y}) \oplus \ldots \oplus \mathcal{D}_{X_n} f_1|_{\underline{\vec{\nu}}}(\vec{Y}), \\ \vdots \\ \mathcal{D}_{X_1} f_n|_{\underline{\vec{\nu}}}(\vec{Y}) \oplus \ldots \oplus \mathcal{D}_{X_n} f_n|_{\underline{\vec{\nu}}}(\vec{Y}) \end{matrix} \right\rangle$$

$\mathcal{D} f_i|_{\underline{\vec{\nu}}}(\vec{Y})$ denotes the i^{th} component of $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$.

Note how the case for "$g \otimes h$" in Eqn. (7) resembles the product rule from differential calculus

$$\frac{d}{dx}(g * h) = \frac{dg}{dx} * h + g * \frac{dg}{dx},$$

and in particular the differential form of the product rule:

$$d(g * h) = dg * h + g * dh.$$

We refer to the creation of Eqn. (5) from $\vec{X} = \vec{f}(\vec{X})$ as the ***NPA linearizing transformation***.

EXAMPLE 2.5. *For Eqn. (3), the multivariate differential of $\vec{f}$ at the value $\underline{\vec{\nu}} = \langle \underline{\nu}_1, \underline{\nu}_2 \rangle$ is*

$$\begin{aligned} \mathcal{D}\vec{f}|_{(\underline{\nu}_1, \underline{\nu}_2)}(\vec{Y}) &= \left\langle \begin{matrix} \mathcal{D}_{X_1} f_1|_{(\underline{\nu}_1, \underline{\nu}_2)}(\vec{Y}) \oplus \mathcal{D}_{X_2} f_1|_{(\underline{\nu}_1, \underline{\nu}_2)}(\vec{Y}), \\ \mathcal{D}_{X_1} f_2|_{(\underline{\nu}_1, \underline{\nu}_2)}(\vec{Y}) \oplus \mathcal{D}_{X_2} f_2|_{(\underline{\nu}_1, \underline{\nu}_2)}(\vec{Y}) \end{matrix} \right\rangle \\ &= \left\langle \underline{0} \oplus a \otimes Y_2, \underline{0} \oplus \begin{pmatrix} b \otimes Y_2 \otimes \underline{\nu}_2 \otimes c \\ \oplus\ b \otimes \underline{\nu}_2 \otimes Y_2 \otimes c \end{pmatrix} \right\rangle \\ &= \left\langle a \otimes Y_2, \begin{pmatrix} b \otimes Y_2 \otimes \underline{\nu}_2 \otimes c \\ \oplus\ b \otimes \underline{\nu}_2 \otimes Y_2 \otimes c \end{pmatrix} \right\rangle \end{aligned} \tag{8}$$

From Eqn. (5), we then obtain the following linearized system of equations, which is also depicted graphically in Fig. 1(b):

$$\langle Y_1, Y_2 \rangle = \left\langle \begin{pmatrix} a \otimes \underline{\nu}_2 \\ \oplus\ a \otimes Y_2 \end{pmatrix}, \begin{pmatrix} d \\ \oplus\ b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c \\ \oplus\ b \otimes Y_2 \otimes \underline{\nu}_2 \otimes c \\ \oplus\ b \otimes \underline{\nu}_2 \otimes Y_2 \otimes c \end{pmatrix} \right\rangle \tag{9}$$

On the $i+1^{st}$ Newton round, we need to solve Eqn. (9) for $\langle Y_1, Y_2 \rangle$ with $\langle \underline{\nu}_1, \underline{\nu}_2 \rangle$ set to the value $\langle \nu_1^{(i)}, \nu_2^{(i)} \rangle$ obtained on the i^{th} round, and then perform the assignment $\langle \nu_1^{(i+1)}, \nu_2^{(i+1)} \rangle \leftarrow \langle Y_1, Y_2 \rangle$.

Kleene Iteration and Other Conventional Methods. Esparza et al. obtained several results that compare NPA against Kleene iteration—in particular, for interprocedural dataflow analysis, the Newton iteration-sequence is never worse than the Kleene iteration-sequence [7, Thm. 3.9]. However, in practice, interprocedural solvers do not perform Kleene iteration. Kleene iteration is like a ***fair scheduler***: each variable is considered on each round, no matter which components of $\vec{\kappa}^{(i)}$ changed value on the previous round. More commonly, solvers use ***chaotic iteration***, which uses a worklist to consider a variable Y_i only when there has been a change to the value of a variable Y_j on which Y_i depends. For ***intra***procedural problems, there are other techniques, such as elimination methods [4, 9, 31] and Tarjan's path-expression method [29, 30].

3. Overview

This section motivates our main improvement to the NPA method of Esparza et al. by illustrating some of its key points on a simple problem (§3.1 and §3.2). The method presented here is a simplification of our actual method. As shown in §3.3, the simplified method returns a conservative solution to an equation system, but not, in general, the least solution. This issue motivates the additional technical aspects needed to obtain the least solution (see §4.5).

3.1 Linear, Non-Regular, Equation Systems

We will concentrate on the (recursive) equation for Y_2:

$$Y_2 = \begin{pmatrix} d \\ \oplus\ b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c \\ \oplus\ b \otimes Y_2 \otimes \underline{\nu}_2 \otimes c \\ \oplus\ b \otimes \underline{\nu}_2 \otimes Y_2 \otimes c \end{pmatrix} \tag{10}$$

Each monomial in Eqn. (10) is ***linear***. In contrast, the equation for X_2 in the original equation system (Eqn. (3)), $X_2 = d \oplus b \otimes X_2 \otimes X_2 \otimes c$, involves a monomial that is ***quadratic***. In general, as in the example above, NPA reduces the problem of solving a polynomial equation system to solving a sequence of linear equation systems.

Note that the third and fourth monomials in Eqn. (10) each extend Y_2 by nontrivial quantities on both the left and the right. Thus, we are truly working with a linear equation system—***not one that is left-linear or right-linear***.

One can also consider Eqn. (10) as defining the following linear context-free grammar over the set of nonterminals $\{Y_2\}$ and the set of terminals $\{b, c, d, \underline{\nu}_2\}$:

$$Y_2 ::= d \mid b\ \underline{\nu}_2\ \underline{\nu}_2\ c \mid b\ Y_2\ \underline{\nu}_2\ c \mid b\ \underline{\nu}_2\ Y_2\ c \tag{11}$$

The linear context-free language (LCFL) generated by grammar (11) has a matching condition that its strings are all of the form

$$(b[\underline{\nu}_2])^i (d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c)([\underline{\nu}_2]c)^i, \tag{12}$$

where $\#\underline{\nu}_2 + 2\#d = i + 2$ and "$[\underline{\nu}_2]$" denotes an optional occurrence of $\underline{\nu}_2$. Moreover, except for a matched pair in the "center" of the form $b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c$, in each matched pair $\ldots b[\underline{\nu}_2] \ldots [\underline{\nu}_2]c \ldots$, there is an occurrence of $\underline{\nu}_2$ on the left side or the right side, but not both.

DEFINITION 3.1. *An equation system over semiring $\mathcal{S}$ is an **LCFL equation system** if each equation has the form*

$$Y_j = c_j \oplus \bigoplus_{i,k} (a_{i,j,k} \otimes Y_i \otimes b_{i,j,k}),$$

where $a_{i,j,k}, b_{i,j,k}, c_j \in \mathcal{S}$.

3.2 Problem Statement: "Regularizing" an LCFL Equation System

As mentioned earlier, NPA performs a Kleene-like iteration, during which a linear correction is applied on each round. Defn. 3.1 allows

us to be more precise: the correction value used on each round is the solution to an LCFL equation system. Our first contribution to NPA is to address the following problem:

> Given an LCFL equation system L, devise an efficient method for finding the least solution of L.

DEFINITION 3.2. *An equation system over semiring $\mathcal{S}$ is a **left-linear equation system** if each equation has the form*

$$Z_j = c_j \oplus \bigoplus_{i,k} (Z_i \otimes b_{i,j,k}),$$

where $b_{i,j,k}, c_j \in \mathcal{S}$.

In contrast to a general LCFL equation system, with a left-linear equation system one can always collect coefficients for a given Z_i—i.e., $d_{i,j} = \bigoplus_k b_{i,j,k}$—so that equations can always be put in a form in which Z_j has a single dependence on each Z_i:

$$Z_j = c_j \oplus \bigoplus_{i} (Z_i \otimes d_{i,j}),$$

where $d_{i,j}, c_j \in \mathcal{S}$.

A left-linear equation system corresponds to a left-linear grammar, and hence a regular language. The fact that Tarjan's path-expression method [30] provides a fast method for solving left-linear equation systems led us to pose the following question:

> Is it possible to "regularize" the LCFL equation system L that arises on each Newton round—i.e., transform L into a left-linear equation system L_{Reg}?

If the extend ($\otimes$) operation of the semiring is commutative, it is trivial to turn an LCFL equation system into a left-linear equation system. However, in dataflow-analysis problems, we rarely have a commutative extend operation; thus, our goal is to find a way to regularize a ***non-commutative*** LCFL equation system.

On the face of it, this line of attack seems unlikely to pan out; after all, Eqn. (12) resembles the language $\mathcal{L} = \{b^i c^i \mid i \in \mathbb{N}\}$, which is the canonical example of an LCFL that is not regular. $\mathcal{L}$ can be defined via the linear context-free grammar

$$S ::= \epsilon \mid b\ S\ c \tag{13}$$

in which the second production allows matching b's and c's to be accumulated on the left and right sides of nonterminal S. Moreover, if grammar (13) is extended to have K matching rules

$$S ::= \epsilon \mid b_j\ S\ c_j \quad 1 \le j \le K \tag{14}$$

the generated strings have bilateral symmetry, e.g.,

$$\ldots b_2 \underbrace{b_1 c_1}\ c_2 \ldots$$

Any solution to the problem of regularizing a non-commutative LCFL equation system has to accommodate such mirrored correlation patterns.

The challenge is to devise a way to accumulate matching quantities on both the left and right sides, whereas in a regular language, we can only accumulate values on one side. This observation suggests the strategy of using ***pairs*** in which left-side and right-side values are accumulated separately but concurrently, so that the desired correlation is maintained. Toward this end, we define extend and combine on pairs as follows:

$$(a_1, b_1) \otimes_p (a_2, b_2) = (a_2 \otimes a_1, b_1 \otimes b_2) \tag{15}$$

$$(a_1, b_1) \oplus_p (a_2, b_2) = (a_1 \oplus a_2, b_1 \oplus b_2) \tag{16}$$

Note the order-reversal in the first component of the right-hand side of Eqn. (15): "$a_2 \otimes a_1$."

Given a pair (a, b), we can read out a normal value via the operation $\mathcal{R}(a, b) \stackrel{\text{def}}{=} a \otimes b$. Because of the order-reversal in Eqn. (15), we have

$$\begin{aligned} \mathcal{R}((a_1, b_1) \otimes_p (a_2, b_2)) &= \mathcal{R}((a_2 \otimes a_1, b_1 \otimes b_2)) \\ &= \underbrace{a_2 \otimes \underbrace{a_1 \otimes b_1} \otimes b_2}\ . \end{aligned}$$

The braces highlight the fact that we have achieved the desired mirrored matching of (i) a_1 with b_1, and (ii) a_2 with b_2.

EXAMPLE 3.3. *Using $\otimes_p$ and $\oplus_p$, we can transform a linear equation (and more generally a set of linear equations) by pairing semiring values that appear to the left of a variable with the values that appear to the right of the variable, placing the pair to the variable's right. For instance, Eqn. (10) is transformed into*

$$Z_2 = \begin{pmatrix} & (\underline{1}, d) \\ \oplus_p & (\underline{1}, b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c) \\ \oplus_p & Z_2 \otimes_p (b, \underline{\nu}_2 \otimes c) \\ \oplus_p & Z_2 \otimes_p (b \otimes \underline{\nu}_2, c) \end{pmatrix} \tag{17}$$

*where Z_2 is now a variable that takes on **pairs** of semiring values. After collecting terms, we have an equation of the form*

$$Z_2 = A \oplus_p Z_2 \otimes_p B, \tag{18}$$

$$\textit{where}\quad A = (\underline{1}, d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c), \tag{19}$$

$$\textit{and}\quad B = (b \oplus b \otimes \underline{\nu}_2, \underline{\nu}_2 \otimes c \oplus c). \tag{20}$$

Eqn. (18) is similar to the equation over formal languages

$$Z_2 = A + (Z_2 \cdot B),$$

for which the regular expression $A \cdot B^*$ is a closed-form solution for Z_2. Similarly, the solution of Eqn. (18) for Z_2 over paired semiring values is given by

$$Z_2 = A \otimes_p B^{*_p}, \tag{21}$$

where B^{*_p} denotes $\bigoplus_{i \in \mathbb{N}}^{\text{p}} B^i$ (in which the repeated "multiplication" operation in B^i is $\otimes_p$). If the answer obtained for Z_2 is the pair (w_1, w_2), we can read out the value for Z_2 as $\mathcal{R}((w_1, w_2)) = w_1 \otimes w_2$.

The algorithm demonstrated above can be stated as follows:

ALGORITHM 3.4. *To solve a linear equation system L,*

1. *Convert L into a left-linear equation system L_{Reg} (with weights that consist of pairs of semiring values).*
2. *Find the least solution of equation system L_{Reg}.*
3. *Apply the readout operation $\mathcal{R}$ to the least solution of L_{Reg} to obtain a solution to L.*

In our example, for step (2) we expressed the least solution of Eqn. (18) in closed form, as a regular expression (Eqn. (21)), which means that the solution for Z_2 can be obtained merely by evaluating the regular expression. In general, when equation system L_{Reg} has a larger number of variables, for step (2) we can use Tarjan's path-expression method [30], which finds a regular expression for each of the variables in a set of mutually recursive left-linear equations.

This approach has a lot of promise for Newtonian program analysis because the structure of L_{Reg}—***and hence of the corresponding regular expressions***—remains fixed from round to round. Consequently, we only need to perform the expensive step of regular-expression construction via Tarjan's method once, before the first round. The actions taken for step (2) on each Newton round are as follows: (i) in each regular expression, replace the constant-valued leaves $\{\underline{\nu}_i\}$, which represent previous-round values, with updated constants, and (ii) reevaluate the regular expression.

In our example, the original linearized system of Eqn. (9), transformed to left-linear form, is

$$\langle Z_1, Z_2\rangle = \langle(\underline{1}, a \otimes \underline{\nu}_2) \oplus_p Z_2 \otimes_p (a, \underline{1}), A \oplus_p Z_2 \otimes_p B\rangle,$$

for which we have the closed-form solution

$$\langle Z_1, Z_2\rangle = \left\langle \begin{matrix}(\underline{1}, a \otimes \underline{\nu}_2) \oplus_p A \otimes_p B^{*_p} \otimes_p (a, \underline{1}),\\ A \otimes_p B^{*_p}\end{matrix} \right\rangle. \quad (22)$$

To solve the original system of equations given in Eqn. (3),

1. First, set $\underline{\nu}_2$ to $\underline{0}$ in Eqn. (22) and evaluate the right-hand side:

$$\langle Z_1, Z_2\rangle = \left\langle \begin{matrix}(\underline{1}, \underline{0}) \oplus_p (\underline{1}, d) \otimes_p (b, c)^{*_p} \otimes_p (a, \underline{1}),\\ (\underline{1}, d) \otimes_p (b, c)^{*_p}\end{matrix} \right\rangle. \quad (23)$$

2. Then, until convergence, repeat the following steps:
   (a) Apply $\mathcal{R}$ to the value obtained for Z_2 to obtain the value of $\underline{\nu}_2$ to use during the next round.
   (b) Use that value in Eqns. (19) and (20), and evaluate the right-hand side of Eqn. (22) to obtain new values for Z_1 and Z_2.

3.3 What Fails?

Unfortunately, the method given as Alg. 3.4 is not guaranteed to produce the desired least-fixed-point solution to an LCFL equation system L. The reason is that the read-out operation $\mathcal{R}$ does not, in general, distribute over $\oplus_p$. Consider the equation system

$$X_1 = \underline{1} \qquad X_2 = a_1 X_1 b_1 \oplus a_2 X_1 b_2.$$

This system corresponds to a graph with two paths. The least solution for X_2 is $a_1b_1 \oplus a_2b_2$, where a_1b_1 and a_2b_2 are the contributions from the two paths. However, when treated as a paired-semiring-value problem, we have

$$Z_1 = (\underline{1}, \underline{1}) \qquad Z_2 = Z_2 \otimes_p ((a_1, b_1) \oplus_p (a_2, b_2)).$$

The least solution for Z_2 is $(a_1, b_1) \oplus_p (a_2, b_2)$, whose readout value is $\mathcal{R}((a_1, b_1) \oplus_p (a_2, b_2))$. However, the latter does not equal $a_1b_1 \oplus a_2b_2$.

$$\begin{aligned}\mathcal{R}((a_1, b_1) \oplus_p (a_2, b_2)) &= \mathcal{R}((a_1 \oplus a_2, b_1 \oplus b_2))\\ &= (a_1 \oplus a_2) \otimes (b_1 \oplus b_2)\\ &= a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2\\ &\sqsupseteq a_1b_1 \oplus a_2b_2\\ &= \mathcal{R}((a_1, b_1)) \oplus \mathcal{R}((a_2, b_2)).\end{aligned} \quad (24)$$

In other words, using combines of pairs leads to cross-terms, such as a_2b_1 and a_1b_2, and consequently answers obtained by (i) solving Eqn. (18) over paired semiring values for the combine-over-all-values answer, and (ii) applying $\mathcal{R}$ to the result, could return an overapproximation ($\sqsupseteq$) of the least solution of the original LCFL equation system L.

In the case of Eqn. (18), $A = (\underline{1}, d \oplus b\underline{\nu}_2\underline{\nu}_2c)$ and $B = (b \oplus b\underline{\nu}_2, \underline{\nu}_2c \oplus c)$. One of the "strings" described by $A \otimes_p B^{*_p}$ is

$$\begin{aligned}AB &= (\underline{1}, d \oplus b\underline{\nu}_2\underline{\nu}_2c) \otimes_p (b \oplus b\underline{\nu}_2, \underline{\nu}_2c \oplus c)\\ &= (b \oplus b\underline{\nu}_2, (d \oplus b\underline{\nu}_2\underline{\nu}_2c)(\underline{\nu}_2c \oplus c))\\ &= (b \oplus b\underline{\nu}_2, d\underline{\nu}_2c \oplus dc \oplus b\underline{\nu}_2\underline{\nu}_2c\underline{\nu}_2c \oplus b\underline{\nu}_2\underline{\nu}_2cc),\end{aligned}$$

and hence,

	Eqn. (12)-term?	i	$\#\underline{\nu}_2 + 2\#d$
$\mathcal{R}(AB) = bd\underline{\nu}_2c$	✓	1	3
$\oplus\ bdc$	χ	n/a	2
$\oplus\ bb\underline{\nu}_2\underline{\nu}_2c\underline{\nu}_2c$	✓	1	3
$\oplus\ bb\underline{\nu}_2\underline{\nu}_2cc$	χ	n/a	2
$\oplus\ b\underline{\nu}_2d\underline{\nu}_2c$	χ	n/a	4
$\oplus\ b\underline{\nu}_2dc$	✓	1	3
$\oplus\ b\underline{\nu}_2b\underline{\nu}_2\underline{\nu}_2c\underline{\nu}_2c$	χ	n/a	4
$\oplus\ b\underline{\nu}_2b\underline{\nu}_2\underline{\nu}_2cc$	✓	1	3

(25)

Of the eight terms on the right-hand side of $\mathcal{R}(AB)$, only four meet the conditions of Eqn. (12): $bd\underline{\nu}_2c$, $bb\underline{\nu}_2\underline{\nu}_2c\underline{\nu}_2c$, $b\underline{\nu}_2dc$, and $b\underline{\nu}_2b\underline{\nu}_2\underline{\nu}_2cc$. The remaining four terms are undesired cross-terms that arise from the properties of $\mathcal{R}$, $\oplus_p$, $\otimes$, and $\oplus$.

Because of the presence of the four cross-terms, the answer computed by $\mathcal{R}(AB)$ is an overapproximation of what we would like it to contribute to the answer; similarly, $\mathcal{R}(A \otimes_p B^{*_p})$ is an overapproximation of the least-fixed-point solution of Eqn. (3).

4. "Regularizing" an LCFL Equation System Redux

In light of the example presented in §3, the prospects for harnessing Tarjan's path-expression method for use during NPA look rather bleak. However, there is still one glimmer of hope:

> A transformation of the linearized problem to left-linear form is not actually forced to use ***pairing***: given a "coupled value" $c = (a, b)$, we never need to recover from c the value of either a or b alone; we only need to be able to obtain the value $a \otimes b$.

Thus, by using some other binary operator to couple values together, it may still be possible to perform a transformation similar to the conversion of Eqn. (10) into Eqn. (17). Of course, the final answer read out of the solution to the left-linear problem must not have contributions from undesired cross-terms.

4.1 A Different Kind of Pairing

We define the desired "coupling" operation in terms of two primitives: ***transpose*** and ***tensor product***:

DEFINITION 4.1. *Let $\mathcal{S} = (D, \oplus, \otimes, \underline{0}, \underline{1})$ be a semiring. $\mathcal{S}$ has a **transpose** operation, denoted by $\cdot^t : D \to D$, if for all elements $a, a_1, a_2 \in D$ the following properties hold:*

$$(a_1 \otimes a_2)^t = a_2^t \otimes a_1^t \quad (26)$$

$$(a_1 \oplus a_2)^t = a_1^t \oplus a_2^t \quad (27)$$

$$(a^t)^t = a. \quad (28)$$

*A **tensor-product** semiring over $\mathcal{S}$ is defined to be another semiring $\mathcal{S}_\mathcal{T} = (D_\mathcal{T}, \oplus_\mathcal{T}, \otimes_\mathcal{T}, \underline{0}_\mathcal{T}, \underline{1}_\mathcal{T})$, where $\mathcal{S}$ and $\mathcal{S}_\mathcal{T}$ support a **tensor-product** operation, denoted by $\odot : D \times D \to D_\mathcal{T}$, such that for all $a, a_1, a_2, b_1, b_2, c_1, c_2 \in D$, the following properties hold:*

$$\underline{0} \odot a = a \odot \underline{0} \;=\; \underline{0}_\mathcal{T} \quad (29)$$

$$a_1 \odot (b_2 \oplus c_2) = (a_1 \odot b_2) \oplus_\mathcal{T} (a_1 \odot c_2) \quad (30)$$

$$(b_1 \oplus c_1) \odot a_2 = (b_1 \odot a_2) \oplus_\mathcal{T} (c_1 \odot a_2) \quad (31)$$

$$(a_1 \odot b_1) \otimes_\mathcal{T} (a_2 \odot b_2) = (a_1 \otimes a_2) \odot (b_1 \otimes b_2). \quad (32)$$

*A tensor-product semiring defined over a semiring with transpose has a **(sequential) detensor-transpose** operation, denoted by $\natural^{(t,\cdot)} : D_\mathcal{T} \to D$, if for all elements $a_1, a_2 \in D$ and $p_1, p_2 \in D_\mathcal{T}$ the following properties hold:*

$$\natural^{(t,\cdot)}(a_1 \odot a_2) = (a_1^t \otimes a_2) \quad (33)$$

$$\natural^{(t,\cdot)}(p_1 \oplus_\mathcal{T} p_2) = \natural^{(t,\cdot)}(p_1) \oplus \natural^{(t,\cdot)}(p_2). \quad (34)$$

We assume that Eqns. (27), (30), (31), and (34) also hold for infinite combines.

*For brevity, we say that $\mathcal{S}$ is an **admissible semiring** if (i) $\mathcal{S}$ has a transpose operation, (ii) $\mathcal{S}$ has an associated tensor-product semiring $\mathcal{S}_\mathcal{T}$, and (iii) $\mathcal{S}_\mathcal{T}$ has a sequential detensor-transpose operation. Henceforth, we consider only admissible semirings.*

*The operation to **couple** pairs of values from an admissible semiring, denoted by $\mathcal{C} : D \times D \to D_\mathcal{T}$, is defined as follows:*

$$\mathcal{C}(a, b) \stackrel{\text{def}}{=} (a^t \odot b).$$

Note that by Eqns. (26) and (32),

$$\begin{aligned}\mathcal{C}(a_1,b_1)\otimes_{\mathcal{T}}\mathcal{C}(a_2,b_2) &= (a_1^t\odot b_1)\otimes_{\mathcal{T}}(a_2^t\odot b_2)\\ &= (a_1^t\otimes a_2^t)\odot(b_1\otimes b_2)\\ &= (a_2\otimes a_1)^t\odot(b_1\otimes b_2)\\ &= \mathcal{C}(a_2\otimes a_1, b_1\otimes b_2)\end{aligned}\tag{35}$$

The order-reversal vis à vis $\otimes_{\mathcal{T}}$ and $\otimes$ in Eqn. (35) will substitute for the order-reversal vis à vis $\otimes_p$ and $\otimes$ in Eqn. (15).

The operator that plays the role of $\mathcal{R}$ is $\lightning^{(t,\cdot)}$. The superscript in $\lightning^{(t,\cdot)}$ serves as a reminder that Eqn. (33) performs an additional transpose on the first argument of a coupled value $(a^t\odot b)$, so that $\lightning^{(t,\cdot)}(a^t\odot b)$ becomes $(a^t)^t\otimes b = a\otimes b$. Consequently,

$$\begin{aligned}\lightning^{(t,\cdot)}(\mathcal{C}(a_2\otimes a_1, b_1\otimes b_2)) &= \lightning^{(t,\cdot)}((a_2\otimes a_1)^t\odot(b_1\otimes b_2))\\ &= ((a_2\otimes a_1)^t)^t\otimes(b_1\otimes b_2)\\ &= \underbrace{a_2\otimes\underbrace{a_1\otimes b_1}\otimes b_2}\end{aligned}$$

which has the desired matching of a_1 with b_1 and a_2 with b_2. Moreover, by Eqn. (34), $\lightning^{(t,\cdot)}$ does not produce cross-terms:

$$\begin{aligned}\lightning^{(t,\cdot)}((a_1^t\odot b_1)\oplus_{\mathcal{T}}(a_2^t\odot b_2)) &= \lightning^{(t,\cdot)}(a_1^t\odot b_1)\oplus_{\mathcal{T}}\lightning^{(t,\cdot)}(a_2^t\odot b_2)\\ &= a_1b_1\oplus a_2b_2.\end{aligned}$$

4.2 The Regularizing Transformation

DEFINITION 4.2. *Given an LCFL equation system L over admissible semiring $\mathcal{S}$, the* ***regularizing transformation*** *τ_{Reg} creates a left-linear equation system $L_{\mathcal{T}} = \tau_{Reg}(L)$ over $\mathcal{S}_{\mathcal{T}}$ by transforming each equation of L as follows:*

$$\frac{Y_j = c_j\oplus\bigoplus_{i,k}(a_{i,j,k}\otimes Y_i\otimes b_{i,j,k})}{Z_j = (\underline{1}\odot c_j)\oplus_{\mathcal{T}}\bigoplus_{i,k}{}_{\mathcal{T}}(Z_i\otimes_{\mathcal{T}}(a_{i,j,k}\odot b_{i,j,k}))}\ \tau_{\text{REG}}$$

where Z_i and Z_j are variables that take on values from tensor-product semiring $\mathcal{S}_{\mathcal{T}}$.

We also use τ_{Reg} as a function on right-hand-side terms:

$$\begin{aligned}&\tau_{Reg}(c_j\oplus\bigoplus_{i,k}(a_{i,j,k}\otimes Y_i\otimes b_{i,j,k}))\\ &\qquad\stackrel{\text{def}}{=}(\underline{1}\odot c_j)\oplus_{\mathcal{T}}\bigoplus_{i,k}{}_{\mathcal{T}}(Z_i\otimes_{\mathcal{T}}(a_{i,j,k}\odot b_{i,j,k})).\end{aligned}\tag{36}$$

We use $Coeff_i(\cdot)$ to select Z_i's coefficient in Eqn. (36):

$$Coeff_i(\tau_{Reg}(c_j\oplus\bigoplus_{i,k}(a_{i,j,k}\otimes Y_i\otimes b_{i,j,k}))) \stackrel{\text{def}}{=} \bigoplus_{k}{}_{\mathcal{T}}(a_{i,j,k}\odot b_{i,j,k}).$$

Finally, we extend τ_{Reg} to operate component-wise on vectors:

$$\tau_{Reg}(\vec{E}) \stackrel{\text{def}}{=} \langle\tau_{Reg}(E_1),\ldots,\tau_{Reg}(E_n)\rangle.$$

EXAMPLE 4.3. *Using τ_{Reg}, Eqn. (10) would be transformed into*

$$Z_2 = \begin{pmatrix}(\underline{1}^t\odot(d\oplus b\otimes\underline{\nu}_2\otimes\underline{\nu}_2\otimes c))\\ \oplus_{\mathcal{T}}\ Z_2\otimes_{\mathcal{T}}(b^t\odot(\underline{\nu}_2\otimes c))\\ \oplus_{\mathcal{T}}\ Z_2\otimes_{\mathcal{T}}((b\otimes\underline{\nu}_2)^t\odot c)\end{pmatrix}\tag{37}$$

which is depicted in Fig. 2. After collecting terms, we have

$$Z_2 = A\oplus_{\mathcal{T}}(Z_2\otimes_{\mathcal{T}}B),\tag{38}$$

$$\textit{where}\quad A = (\underline{1}^t\odot(d\oplus b\otimes\underline{\nu}_2\otimes\underline{\nu}_2\otimes c))$$

$$\textit{and}\quad B = (b^t\odot(\underline{\nu}_2\otimes c))\oplus_{\mathcal{T}}((b\otimes\underline{\nu}_2)^t\odot c)\tag{39}$$

4.3 Solving an LCFL Equation System

We can now harness Tarjan's path-expression algorithm to solve an LCFL equation system.

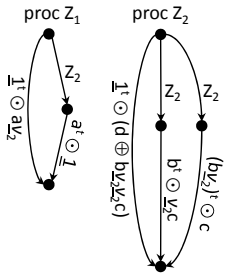


Figure 2. Graphical representation of the linearized equation system over $\vec{Z}$ obtained from Eqn. (3) via Defn. 4.2.

ALGORITHM 4.4. *To solve an LCFL equation system L over admissible semiring $\mathcal{S}$,*

1. *Apply τ_{Reg} to L to create the left-linear equation system $L_{\mathcal{T}}$ over the tensor-product semiring $\mathcal{S}_{\mathcal{T}}$.*[5]
2. *Use Tarjan's path-expression algorithm to find a regular expression Reg_i for each variable Z_i in $L_{\mathcal{T}}$.*
3. *Obtain $\vec{Z}$, the least solution to $L_{\mathcal{T}}$: for each variable Z_i, evaluate Reg_i; i.e., $Z_i \leftarrow [\![Reg_i]\!]_{\mathcal{T}}$, where $[\![\cdot]\!]_{\mathcal{T}}$ denotes the interpretation of the regular-expression operators in $\mathcal{S}_{\mathcal{T}}$.*
4. *Apply $\lightning^{(t,\cdot)}$ to each component of $\vec{Z}$ to obtain the solution to the original LCFL equation system L; i.e., $Y_i \leftarrow \lightning^{(t,\cdot)}(Z_i)$.*

The regular expressions created in step 2 are actually *generalized regular expressions* that involve (i) $\oplus_{\mathcal{T}}$, $\otimes_{\mathcal{T}}$, and ${}^{*_{\mathcal{T}}}$, which are interpreted in $\mathcal{S}_{\mathcal{T}}$; (ii) $\oplus$, $\otimes$, *, and ^{t}, which are interpreted in $\mathcal{S}$; (iii) $\odot$, which is interpreted in $\mathcal{S}$ to create a value in $\mathcal{S}_{\mathcal{T}}$; (iv) the symbols $\{\underline{\nu}_i\}$, which are associated with values in $\mathcal{S}$; and (v) constants from the semirings $\mathcal{S}$ and $\mathcal{S}_{\mathcal{T}}$.

DEFINITION 4.5. ***Generalized regular expressions*** *are defined by the following grammar:*

$$\begin{array}{lll} exp_{\mathcal{T}} ::= a_{\mathcal{T}}\in\mathcal{S}_{\mathcal{T}} & expt ::= exp^t & exp ::= a\in\mathcal{S}\\ \quad\mid\ expt\odot exp & & \quad\mid\ \underline{\nu}_i\\ \quad\mid\ exp\oplus_{\mathcal{T}}exp & & \quad\mid\ exp\oplus exp\\ \quad\mid\ exp\otimes_{\mathcal{T}}exp & & \quad\mid\ exp\otimes exp\\ \quad\mid\ exp^{*_{\mathcal{T}}} & & \quad\mid\ exp^*\end{array}$$

Given a vector of values $\vec{\nu}$, a generalized regular expression is evaluated as follows, where $(\!|op|\!)$ denotes the interpretation of op in $\mathcal{S}$ or $\mathcal{S}_{\mathcal{T}}$, as appropriate:

$$[\![e]\!]_{\mathcal{T}}\vec{\nu} \stackrel{\text{def}}{=} \begin{cases} a_{\mathcal{T}} & \text{if } e = a_{\mathcal{T}}\in\mathcal{S}_{\mathcal{T}}\\ ([\![e_1]\!]\vec{\nu})^{(\!|t|\!)}(\!|\odot|\!)\,[\![e_2]\!]\vec{\nu} & \text{if } e = e_1^t\odot e_2\\ [\![e_1]\!]_{\mathcal{T}}\vec{\nu}\,(\!|\oplus_{\mathcal{T}}|\!)\,[\![e_2]\!]_{\mathcal{T}}\vec{\nu} & \text{if } e = e_1\oplus_{\mathcal{T}}e_2\\ [\![e_1]\!]_{\mathcal{T}}\vec{\nu}\,(\!|\otimes_{\mathcal{T}}|\!)\,[\![e_2]\!]_{\mathcal{T}}\vec{\nu} & \text{if } e = e_1\otimes_{\mathcal{T}}e_2\\ ([\![e_1]\!]_{\mathcal{T}}\vec{\nu})^{(\!|*_{\mathcal{T}}|\!)} & \text{if } e = (e_1)^{*_{\mathcal{T}}}\end{cases}$$

$$[\![e]\!]\vec{\nu} \stackrel{\text{def}}{=} \begin{cases} a & \text{if } e = a\in\mathcal{S}\\ (\vec{\nu})_i & \text{if } e = \underline{\nu}_i\\ [\![e_1]\!]\vec{\nu}\,(\!|\oplus|\!)\,[\![e_2]\!]\vec{\nu} & \text{if } e = e_1\oplus e_2\\ [\![e_1]\!]\vec{\nu}\,(\!|\otimes|\!)\,[\![e_2]\!]\vec{\nu} & \text{if } e = e_1\otimes e_2\\ ([\![e_1]\!]\vec{\nu})^{(\!|*|\!)} & \text{if } e = (e_1)^*\end{cases}$$

EXAMPLE 4.6. *In step 2, the regular expression that would be obtained for variable Z_2—defined in Eqn. (38)—is $Z_2 = A\otimes_{\mathcal{T}}B^{*_{\mathcal{T}}}$. In this expression, $B^{*_{\mathcal{T}}}$ denotes tensored Kleene-star: $B^{*_{\mathcal{T}}} = \bigoplus_{i\in\mathbb{N}}{}_{\mathcal{T}}\,B^i$, where the repeated multiplication operation in B^i is the operation $\otimes_{\mathcal{T}}$.*

[5] In essence, $L_{\mathcal{T}}$ corresponds to an intraprocedural dataflow-analysis problem over $\mathcal{S}_{\mathcal{T}}$.

In step 4, to obtain the value Y_2 that solves Eqn. (10), we would evaluate $\natural^{(t,\cdot)}(Z_2) = \natural^{(t,\cdot)}(A \otimes_{\mathcal{T}} B^{*\mathcal{T}})$.

THEOREM 4.7. *Given an LCFL equation system L over admissible semiring $\mathcal{S}$, Alg. 4.4 finds the least solution of L.*

4.4 Discussion

It is instructive to consider the contributions of the different powers of B to the value of $\natural^{(t,\cdot)}(A \otimes_{\mathcal{T}} B^{*\mathcal{T}})$.

$$\begin{aligned}\natural^{(t,\cdot)}(A \otimes_{\mathcal{T}} B^{*\mathcal{T}}) &= \natural^{(t,\cdot)}(A \oplus AB \oplus ABB \oplus \ldots) \\ &= \natural^{(t,\cdot)}(A) \oplus \natural^{(t,\cdot)}(AB) \oplus \natural^{(t,\cdot)}(ABB) \oplus \ldots\end{aligned}$$

To demonstrate why the use of tensor-products avoids the cross-terms that spoiled the approach described in §3, we focus on $\natural^{(t,\cdot)}(AB)$:

$$\begin{aligned}
&\natural^{(t,\cdot)}(AB) \\
&= \natural^{(t,\cdot)}\begin{pmatrix} (\underline{1}^t \odot (d \oplus b\underline{\nu}_2\underline{\nu}_2 c)) \\ \otimes_{\mathcal{T}} \ ((b^t \odot (\underline{\nu}_2 c)) \oplus_{\mathcal{T}} ((b\underline{\nu}_2)^t \odot c)) \end{pmatrix} \\
&= \natural^{(t,\cdot)}\begin{pmatrix} (\underline{1}^t \odot (d \oplus b\underline{\nu}_2\underline{\nu}_2 c)) \otimes_{\mathcal{T}} (b^t \odot (\underline{\nu}_2 c)) \\ \oplus_{\mathcal{T}} \ (\underline{1}^t \odot (d \oplus b\underline{\nu}_2\underline{\nu}_2 c)) \otimes_{\mathcal{T}} ((b\underline{\nu}_2)^t \odot c) \end{pmatrix} \quad (40) \\
&= \natural^{(t,\cdot)}\begin{pmatrix} ((\underline{1}^t b^t) \odot ((d \oplus b\underline{\nu}_2\underline{\nu}_2 c)(\underline{\nu}_2 c))) \\ \oplus_{\mathcal{T}} \ ((\underline{1}^t (b\underline{\nu}_2)^t) \odot ((d \oplus b\underline{\nu}_2\underline{\nu}_2 c)c)) \end{pmatrix} \quad (41) \\
&= \natural^{(t,\cdot)}\begin{pmatrix} (b^t \odot (d\underline{\nu}_2 c \oplus b\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c)) \\ \oplus_{\mathcal{T}} \ ((b\underline{\nu}_2)^t \odot (dc \oplus b\underline{\nu}_2\underline{\nu}_2 cc)) \end{pmatrix} \\
&= \natural^{(t,\cdot)}\begin{pmatrix} (b^t \odot d\underline{\nu}_2 c) \oplus_{\mathcal{T}} (b^t \odot b\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c) \\ \oplus_{\mathcal{T}} \ ((b\underline{\nu}_2)^t \odot dc) \oplus_{\mathcal{T}} ((b\underline{\nu}_2)^t \odot b\underline{\nu}_2\underline{\nu}_2 cc) \end{pmatrix} \\
&= \begin{pmatrix} \natural^{(t,\cdot)}(b^t \odot d\underline{\nu}_2 c) \oplus_{\mathcal{T}} \natural^{(t,\cdot)}(b^t \odot b\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c) \\ \oplus_{\mathcal{T}} \ \natural^{(t,\cdot)}((b\underline{\nu}_2)^t \odot dc) \oplus_{\mathcal{T}} \natural^{(t,\cdot)}((b\underline{\nu}_2)^t \odot b\underline{\nu}_2\underline{\nu}_2 cc) \end{pmatrix} \quad (42) \\
&= bd\underline{\nu}_2 c \oplus_{\mathcal{T}} bb\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c \oplus_{\mathcal{T}} b\underline{\nu}_2 dc \oplus_{\mathcal{T}} b\underline{\nu}_2 b\underline{\nu}_2\underline{\nu}_2 cc. \quad (43)
\end{aligned}$$

In contrast to the eight summands that arose in Eqn. (25), the four summands that appear in Eqn. (43) each meet the matching condition of Eqn. (12). Moreover, these four terms are exactly the ones marked with ✓ in Eqn. (25).

In general, $\natural^{(t,\cdot)}(A \otimes_{\mathcal{T}} B^k)$ contributes summands of the form $(b[\underline{\nu}_2])^k(d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c)([\underline{\nu}_2]c)^k$ that satisfy the matching condition of Eqn. (12) (e.g., $\#\underline{\nu}_2 + 2\#d = k+2$). Eqn. (43) shows the contribution of $\natural^{(t,\cdot)}(AB)$ (i.e., $k = 1$), and $\#\underline{\nu}_2 + 2\#d = 3$ holds for each summand.

Compared to the derivation leading up to Eqn. (25) in §3.2, the derivation above of the contribution of AB to $\natural^{(t,\cdot)}(A \otimes_{\mathcal{T}} B^{*\mathcal{T}})$ illustrates how the properties of transpose, tensor product, and detensor-transpose allow exactly the right pairings of semiring values b and c to arise in Eqn. (43). The two summands in Eqn. (39)—and hence the arguments on the right-hand sides of the two occurrences of $\otimes_{\mathcal{T}}$ in Eqn. (40)—are $b^t \odot (\underline{\nu}_2 \otimes c)$ and $(b \otimes \underline{\nu}_2)^t \odot c$. These terms capture the two recursive summands that define Y_2 in Eqn. (10): $b \otimes Y_2 \otimes \underline{\nu}_2 \otimes c$ and $b \otimes \underline{\nu}_2 \otimes Y_2 \otimes c$. In particular, in Eqn. (40) the position of "$\odot$" in $b^t \odot (\underline{\nu}_2 \otimes c)$ and $(b \otimes \underline{\nu}_2)^t \odot c$ can be viewed as marking the position of the recursive occurrences of Y_2 in $b \otimes Y_2 \otimes \underline{\nu}_2 \otimes c$ and $b \otimes \underline{\nu}_2 \otimes Y_2 \otimes c$, respectively. In effect, the derivation of Eqn. (41) from Eqn. (40) is where an LCFL-like "substitution" takes place in the "middle" of $b^t \odot (\underline{\nu}_2 \otimes c)$ and $(b \otimes \underline{\nu}_2)^t \odot c$.

4.5 Newtonian Program Analysis via Tensor Products

To sum up, Newtonian Program Analysis via Tensor Products (NPA-TP) is based on a way to find the least solution to a system of equations over a semiring $\mathcal{S}$. We use Eqns. (4) and (5) of Esparza et al. but apply Alg. 4.4 to solve Eqn. (5).

Our approach can also be restated as follows: we solve the following sequence of problems for $\vec{\nu}$:

$$\boxed{\begin{array}{rcl} \vec{\nu}^{(0)} & = & \langle f_1(\underline{\vec{0}}), \ldots, f_n(\underline{\vec{0}}) \rangle \\ \vec{\nu}^{(i+1)} & = & \langle \natural^{(t,\cdot)}(Z_1^{(i)}), \ldots, \natural^{(t,\cdot)}(Z_n^{(i)}) \rangle \end{array}} \quad (44)$$

where $\vec{Z}^{(i)} = \langle Z_1^{(i)}, \ldots, Z_n^{(i)} \rangle$ is the least solution of the following equation system over $\mathcal{S}^{\mathcal{T}}$:

$$\boxed{\tau_{\text{Reg}}(\vec{Y} = \vec{f}(\vec{\nu}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y}))} \quad (45)$$

(Recall that τ_{Reg} replaces Y's with Z's.)

In practice, the LCFL equation systems that arise on successive rounds have a great deal of structure in common, and it is possible to arrange to call Tarjan's path-expression algorithm only a single time to create parameterized regular expressions that can be used to solve Eqn. (45) on each round. (See the discussion of step 4 of Alg. 7.1.)

5. NPA-TP for Predicate-Abstraction Domains

In this section, we explain how NPA-TP applies to predicate-abstraction domains—an instantiation denoted by NPA-TP[PA]. For a given predicate-abstraction domain, NPA-TP[PA] has the following ingredients:

Semiring: A predicate-abstraction domain over predicate set P is a relational weight domain $(2^{A \times A}, \cup, ;, \emptyset, id)$ (Defn. 2.2), where A is the set of *Boolean assignments* to P; that is, $A = P \to$ Bool. ($P \to$ Bool is isomorphic to 2^P.) Let N denote $|A|$. Each semiring element R can be thought of as an $N \times N$ Boolean matrix

$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,N} \\ \vdots & \ddots & \vdots \\ r_{N,1} & \cdots & r_{N,N} \end{bmatrix}.$$

We will write this as "$R(A, A')$" when we wish to introduce names for the index sets of the matrix.

Transpose: The transpose operation is matrix transpose. Semantically, transpose reverses a relation:

$$R^t = \{(a', a) \mid R^{-1}(a', a)\} = \{(a', a) \mid R(a, a')\}.$$

Tensor Product: The tensor-product operation is Kronecker product of Boolean matrices:

$$R \odot S = \begin{bmatrix} r_{1,1}S & \cdots & r_{1,N}S \\ \vdots & \ddots & \vdots \\ r_{N,1}S & \cdots & r_{N,N}S \end{bmatrix}$$

which is an $N^2 \times N^2$ binary matrix whose entries are

$$(R \odot S)[(a-1)N + b, (a'-1)N + b'] = R(a, a') \wedge S(b, b').$$

Semantically, tensor-product builds 4-ary relations:

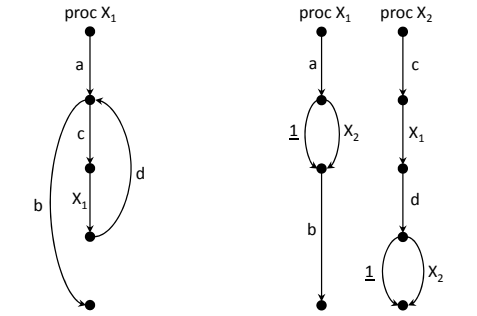
$$R \odot S = \{(a, b, a', b') \mid R(a, a') \wedge S(b, b')\}.$$

Coupling: The coupling of R and S is the tensor-transpose relation $R^t \odot S$; hence, $R^t \odot S = \{(a', b, a, b') \mid R(a, a') \wedge S(b, b')\}$,

$$R^t \odot S = \begin{bmatrix} r_{1,1}S & \cdots & r_{N,1}S \\ \vdots & \ddots & \vdots \\ r_{1,N}S & \cdots & r_{N,N}S \end{bmatrix}$$

and thus

$$(R^t \odot S)[(a'-1)N + b, (a-1)N + b'] = R(a, a') \wedge S(b, b').$$



$X_1 = a(cX_1d)^*b$ (a)

$X_1 = a(\underline{1} \oplus X_2)b$
$X_2 = cX_1d(\underline{1} \oplus X_2)$ (b)

Figure 3. Two equation systems and their graphical representations. (a) A recursive program that contains a loop. (b) "Loop-free" variant in which the loop is encoded by recursive procedure X_2.

Detensor Transpose: If T is a tensor-transpose relation,

$$\not{t}^{(t,\cdot)}(T(A', B, A, B')) \stackrel{\text{def}}{=} \exists A', B : T(A', B, A, B') \wedge A' = B. \tag{46}$$

THEOREM 5.1. *The transpose, tensor-product, and detensor-transpose operations defined above satisfy Eqns. (26)–(34).*

6. Loops

In this section, we summarize how programs with loops can be handled in the method of Esparza et al., and then present an alternative method for handling loops.

6.1 Loops for Esparza et al.

As presented by Esparza et al., NPA applies to a system of equations in which each right-hand-side expression is a ***polynomial***: semiring expressions consist of semiring constants, variables, extend, and combine (where each occurrence of a variable corresponds to a procedure call). The restriction to polynomials means that each procedure must consist of loop-free code. Recursive equations are permitted, and thus a program whose (original) procedures contain loops can be handled by systematically replacing each loop with a call to an appropriate recursive, loop-free procedure.

EXAMPLE 6.1. *Consider program (i) below, which is shown in graphical form in Fig. 3(a).*

```
X1() {
  sa;
  while (⋆) {
    sc;
    X1();
    sd
  }
  sb
}
```

(i)

```
X1() {
  sa;
  if (⋆) X2()
  sb
}
```

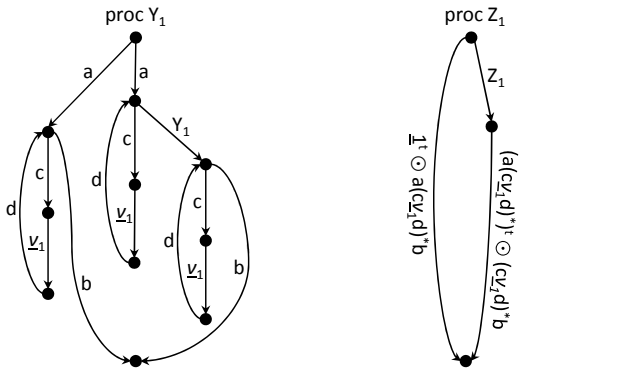
```
X2() {
  sc;
  X1();
  sd;
  if (⋆) X2()
}
```

(ii)

Program (ii) shows one possible transformation of program (i) to put it in loop-free form. Fig. 3(b) shows program (ii) in graphical form, and also as an equation system.

6.2 An Alternative Approach to Handling Loops

We now show how to extend NPA and NPA-TP to handle programs with loops in a different way. Our approach involves introducing a Kleene-star operator, and allowing the right-hand side of each equation to be a *regular expression*:



$$Y_1 = \begin{array}{l} a(c\underline{\nu}_1 d)^* b \\ \oplus\, a(c\underline{\nu}_1 d)^* Y_1 (c\underline{\nu}_1 d)^* b \end{array} \quad (a)$$

$$Z_1 = \begin{array}{l} (\underline{1}^t \odot a(c\underline{\nu}_1 d)^* b) \\ \oplus_{\mathcal{T}} Z_1 \otimes_{\mathcal{T}} \begin{pmatrix} (a(c\underline{\nu}_1 d)^*)^t \\ \odot\, (c\underline{\nu}_1 d)^* b \end{pmatrix} \end{array} \quad (b)$$

Figure 4. (a) NPA and (b) NPA-TP equation systems that result from the equation "$X_1 = a(cX_1d)^*b$" (from Fig. 3(a)), when both NPA and NPA-TP are extended to handle Kleene-star.

DEFINITION 6.2. *Let $\mathcal{S}$ be an ω-continuous semiring and $\mathcal{X}$ a finite set of variables. The following grammar defines an* ***equation system over $\mathcal{S}$ and $\mathcal{X}$, with regular right-hand sides****:*

$$\begin{array}{rcl} \textit{equation system} & ::= & \textit{set of equation} \\ \textit{equation} & ::= & \textit{var} = \textit{exp} \\ \textit{exp} & ::= & a \in \mathcal{S} \mid \textit{var} \in \mathcal{X} \mid \textit{exp} \oplus \textit{exp} \\ & & \mid \textit{exp} \otimes \textit{exp} \mid \textit{exp}^* \end{array}$$

Fig. 3(a) shows the recursive equation over $\mathcal{S}$ and $\{X_1\}$, with regular right-hand side "$a(cX_1d)^*b$," that corresponds to program (i) from Ex. 6.1.

Given a program, there may be some massaging required to create the corresponding system of equations with regular right-hand sides. However, this transformation can be performed automatically by applying Tarjan's path-expression algorithm to the CFG of each procedure of the program.[6] The result of this pre-processing step is a system of equations with regular right-hand sides.

The Differential of a Regular Expression. Because equation right-hand sides can now include occurrences of Kleene-star, we need to be able to obtain the differential of an expression of the form $(g(\vec{X}))^*$.

THEOREM 6.3. *Let $f(\vec{X}) = (g(\vec{X}))^*$, then*

$$\mathcal{D}_{X_j} f|_{\underline{\vec{\nu}}}(\vec{Y}) = (g(\underline{\vec{\nu}}))^* \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^* \tag{47}$$

Thm. 6.3 implies that the differential of a component function $f_i(\vec{X})$ in an equation system with regular right-hand sides can be obtained by the rule given in Defn. 2.4, extended with one more case for Kleene-star:

$$\boxed{\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y}) = (g(\underline{\vec{\nu}}))^* \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^* \quad \text{if } f_i = g^*}$$

This rule, like the others given in Defn. 2.4, produces a linear term. Consequently, when Defn. 2.4 is augmented with the above rule, the NPA linearizing transformation is still guaranteed to create an LCFL equation system over $\mathcal{S}$. Therefore, for NPA-TP we can still

[6] This application of Tarjan's path-expression algorithm should not be confused with the later use of the path-expression method to create parameterized regular expressions that are used to solve Eqn. (45) on each round of NPA-TP. See steps 1 and 4 of Alg. 7.1.

create a left-linear equation system over $\mathcal{S}_{\mathcal{T}}$ by applying τ_{Reg} to the LCFL equation system. Fig. 4(a) and Fig. 4(b) show the LCFL and left-linear equations for Y_1 and Z_1 obtained from Fig. 3(a) by these transformations.

7. Algorithm Pragmatics

NPA-TP can be implemented in a straightforward manner using Eqns. (44) and (45). However, as mentioned in §4.5, the LCFL equation systems that arise on successive rounds have a great deal of structure in common. To exploit these commonalities, our implementation of NPA-TP implements Eqns. (44) and (45) as decribed below.

In steps 4 and 5 of the algorithm, we work with regular expressions over an alphabet whose symbols have the form $\langle k, j\rangle$. We use the notation $\mathcal{R}[\langle k, j\rangle \leftarrow E]$ to denote $\mathcal{R}$ with regular expression E substituted in for all occurrences of $\langle k, j\rangle$.

ALGORITHM 7.1 (NPA-TP). *The input is an interprocedural dataflow-analysis problem over admissible semiring $\mathcal{S}$. Let $\vec{X}$ denote the set of n procedures of the program.*

1. *Apply Tarjan's path-expression algorithm to the CFG of each procedure in $\vec{X}$ to create a system of recursive equations $\mathcal{E}$ in which*
   - *each variable corresponds to one of the procedures in $\vec{X}$*
   - *the right-hand side of each equation is a regular expression over variables in $\vec{X}$ and constants in $\mathcal{S}$.*

   That is, $\mathcal{E} = \{X_j = \mathit{Rhs}_j(\vec{X}) \mid X_j \in \vec{X}\}$.
2. *For each equation $X_j = \mathit{Rhs}_j(\vec{X}) \in \mathcal{E}$, create the left-linear equation for Z_j over variables in $\vec{Z}$ and coefficients that are generalized regular expressions.*

$$Z_j = \tau_{\mathit{Reg}}(\mathcal{D}\,\mathit{Rhs}_j|_{\underline{\vec{\nu}}}(\vec{Y})).$$

   (Recall that τ_{Reg} replaces Y's with Z's.)
3. *Create a dependence graph $\mathcal{G}$ for the equation system created in step 2.*
   - *$\mathcal{G}$ contains an edge $Z_k \to Z_j$ labeled $\langle k, j\rangle$ if the equation for Z_j contains an occurrence of Z_k on the right-hand side.*
   - *In addition, $\mathcal{G}$ contains a dummy vertex Λ, and for each Z_j, an edge $\Lambda \to Z_j$ labeled $\langle 0, j\rangle$.*
4. *Apply Tarjan's path-expression algorithm to $\mathcal{G}$ (with entry vertex Λ) to create, for each variable $Z_i \in \vec{Z}$, a regular expression $\mathcal{R}_i$ (with tensored operators) over the alphabet $\{\langle k, j\rangle \mid 0 \le k \le n, 1 \le j \le n\}$ (i.e., $[0..n] \times [1..n]$).*
5. *Create the map m, in which variable Z_i, $1 \le i \le n$, is mapped to the regular expression*

$$\begin{aligned}&\mathcal{R}_i[\langle 0, j\rangle \leftarrow (\underline{1}^t \odot \mathit{Rhs}_j(\vec{\nu}))]\\ &\quad[\langle 1, j\rangle \leftarrow \mathit{Coeff}_1(\tau_{\mathit{Reg}}(\mathcal{D}_{X_1}\mathit{Rhs}_j|_{\vec{\nu}}(\vec{Y})))]\\ &\quad\cdots\\ &\quad[\langle n, j\rangle \leftarrow \mathit{Coeff}_n(\tau_{\mathit{Reg}}(\mathcal{D}_{X_n}\mathit{Rhs}_j|_{\vec{\nu}}(\vec{Y})))]\end{aligned}$$

6. $i \leftarrow 0; \vec{\mu} \leftarrow \vec{f}(\vec{\underline{0}})$
7. *Repeat*
   (a) $\vec{\nu}^{(i)} = \vec{\mu}$
   (b) $\vec{\mu} = \langle \mathit{f}^{(t,\cdot)}([\![m(Z_j)]\!]_{\mathcal{T}}\vec{\nu}^{(i)}) \mid Z_j \in \vec{Z}\rangle$
   (c) $i \leftarrow i + 1$
   until $(\vec{\nu}^{(i-1)} = \vec{\mu})$
8. *Return $\vec{\mu}$*

Steps 6 and 7 create the Newton iterates. There are a few aspects of Alg. 7.1 that are worth commenting on.

- Tarjan's algorithm has two separate roles:
  1. In step 1, it is applied to each CFG of the program to create an equation system with regular right-hand sides, which is the input to step 2. Because this equation system can contain occurrences of Kleene-star, it was necessary for us to extend the NPA linearizing transformation, as described in §6.2.
  2. In step 4, it is applied to dependence graph $\mathcal{G}$. If you think of the symbols $\langle k, j\rangle$ on $\mathcal{G}$'s edges as proxies for the regular expressions that replace the symbols in step 5, $\mathcal{G}$ is a relatively straightforward encoding of Eqn. (45): (i) The values $(\underline{1}^t \odot \text{Rhs}_j(\vec{\nu}))$ associated with edge-labels of the form $\langle 0, j\rangle$ represent the (tensored) "seed values" $(\underline{1}^t \odot f_j(\vec{\nu}))$ from the first summand "$\vec{f}(\vec{\nu}^{(i)}) \oplus \ldots$" of Eqn. (45). (ii) The remaining edges of $\mathcal{G}$ encode the regular structure of the recursive portion of Eqn. (45), $\tau_{\text{Reg}}(\vec{Y} = \ldots \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y}))$.
- Because of the calls to Coeff_i in the substitutions performed in step 5, each alphabet symbol $\langle k, j\rangle$ is replaced by a generalized regular expression. Note that a generalized regular expression does not have any occurrences of a variable Z_k. Thus, the only variable-like quantities in each generalized regular expression $m(Z_j)$ are occurrences of symbols, such as $\underline{\nu}_k$. These values are "constants" from $\mathcal{S}$ during a given Newton round, but change value from round to round. In step 7b, rather than explicitly substituting the value $(\vec{\nu}^{(i)})_k$—i.e., the k^{th} component of $\vec{\nu}^{(i)}$—for $\underline{\nu}_k$ in $m(Z_j)$ as a constant-valued leaf, we merely fetch $(\vec{\nu}^{(i)})_k$ by look-up during regular-expression evaluation (Defn. 4.5).
- In the implementation, identical subexpressions of regular expressions are shared. We use a variant of Defn. 4.5 that implements function caching to avoid redundant evaluations in step 7b.

EXAMPLE 7.2. *Consider Eqn. (37) and the corresponding graphical depiction in Fig. 2. The regular expression created for Z_2 is $\langle 0, 2\rangle \oplus_{\mathcal{T}} \langle 0, 2\rangle(\langle 2, 2\rangle)^{*_{\mathcal{T}}}$. After step 5, $m(Z_2)$ is*

$$m(Z_2) = \oplus_{\mathcal{T}} \begin{pmatrix} (\underline{1}^t \odot (d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c)) \\ \otimes_{\mathcal{T}} \left((b^t \odot (\underline{\nu}_2 \otimes c)) \oplus_{\mathcal{T}} ((b \otimes \underline{\nu}_2)^t \odot c)\right)^{*_{\mathcal{T}}} \end{pmatrix}$$

Similarly, the regular expression created for Z_1 is $\langle 0, 1\rangle \oplus_{\mathcal{T}} (\langle 0, 2\rangle \oplus_{\mathcal{T}} \langle 0, 2\rangle(\langle 2, 2\rangle)^{*_{\mathcal{T}}})\langle 2, 1\rangle$, and $m(Z_1)$ is

$$m(Z_1) = (\underline{1} \odot a\underline{\nu}_2) \oplus_{\mathcal{T}} m(Z_2) \otimes_{\mathcal{T}} (a^t \odot \underline{1}).$$

Steps 6 and 7 then repeat the following actions until convergence:

- *Evaluate $m(Z_1)$ and $m(Z_2)$ with respect to the current value of $\underline{\vec{\nu}} = \langle \underline{\nu}_1, \underline{\nu}_2\rangle$ to obtain, say, $w_1, w_2 \in \mathcal{S}_{\mathcal{T}}$, respectively.*
- *Set $\langle \underline{\nu}_1, \underline{\nu}_2\rangle$ to $\langle \mathit{f}^{(t,\cdot)}(w_1), \mathit{f}^{(t,\cdot)}(w_2)\rangle$.*

THEOREM 7.3. *Given an interprocedural dataflow-analysis problem over admissible semiring $\mathcal{S}$, Alg. 7.1 finds the least solution.*

8. Local Variables

This section discusses how to extend NPA and NPA-TP to handle programs with local variables. We adopt the approach introduced by Knoop and Steffen [14]. At a call site at which procedure P calls procedure Q, the local variables of P are modeled as if the current incarnations of P's locals are stored in locations that are inaccessible to Q and to procedures transitively called by Q—consequently, the contents of P's locals cannot be affected by the call to Q; we use special merge functions to combine them with the value returned by Q to create the state after Q returns. (Other work using merge functions includes [16, 21].)

DEFINITION 8.1 (Merge function for a semiring [16]). *Given semiring $\mathcal{S} = (D, \oplus, \otimes, \underline{0}, \underline{1})$, a binary function $M : D \times D \to D$ is an **acceptable merge function for** $\mathcal{S}$ if M obeys the following properties:*

1. *($\underline{0}$-strictness) For all $a, b \in D$, $M(a, \underline{0}) = \underline{0}$ and $M(\underline{0}, b) = \underline{0}$.*

2. *(Distributivity) M distributes over finite and infinite combines in both argument positions; e.g., for all $a, b, c \in D$,*

$$\begin{aligned} M(a \oplus b, c) &= M(a, c) \oplus M(b, c) \\ M(a, b \oplus c) &= M(a, b) \oplus M(a, c) \end{aligned}$$

3. *(Path extension) For all $a, b, c \in D$, $M(a \otimes b, c) = a \otimes M(b, c)$.*

EXAMPLE 8.2. *Consider the following equation system:*

$$\begin{aligned} X_1 &= a \oplus M(M(b, X_2), X_1) \\ X_2 &= c \oplus M(M(d, X_3), X_2) \\ X_3 &= g \oplus (M(e, X_2) \otimes f). \end{aligned} \tag{48}$$

By the path-extension property (Defn. 8.1(3)), Eqn. (48) can be rewritten as follows:

$$\begin{aligned} X_1 &= a \oplus (b \otimes M(\underline{1}, X_2) \otimes M(\underline{1}, X_1)) \\ X_2 &= c \oplus (d \otimes M(\underline{1}, X_3) \otimes M(\underline{1}, X_2)) \\ X_3 &= g \oplus (e \otimes M(\underline{1}, X_2) \otimes f). \end{aligned} \tag{49}$$

Note that by setting $b = \underline{1}$, the path-extension property becomes

$$\text{For all } a, c \in D, M(a, c) = a \otimes M(\underline{1}, c). \tag{50}$$

In an interprocedural-dataflow analysis problem, a corresponds to the abstract value at the call-site in the caller, and c corresponds to the abstract value at the exit-site in the callee. Eqn. (50) shows that for a given procedure Q, much of the work needed for the merge operation for different call-sites on Q can be factored out as $m = M(\underline{1}, c)$. The merge needed at the i^{th} call-site on Q can then be completed by performing $a_i \otimes m$.

We extend our language of regular expressions with the unary operator Project$(\cdot)$, whose semantics is $[\![\text{Project}(e)]\!]\vec{\nu} = M(\underline{1}, [\![e]\!]\vec{\nu})$. Eqn. (49) can be rewritten using Project as follows:

$$\begin{aligned} X_1 &= a \oplus (b \otimes \text{Project}(X_2) \otimes \text{Project}(X_1)) \\ X_2 &= c \oplus (d \otimes \text{Project}(X_3) \otimes \text{Project}(X_2)) \\ X_3 &= g \oplus (e \otimes \text{Project}(X_2) \otimes f). \end{aligned} \tag{51}$$

Merge/Project for a Relational Weight Domain. When the state of a Boolean program has contributions from both global states G and local states L, we use the relational weight domain on $G \times L$, defined as $((G \times L) \times (G \times L) \to \mathbb{B}, \cup, ;, \emptyset, \text{Id})$. A typical element will be denoted by $R(G, L, G', L')$. In this case, the following is an acceptable merge function:

$$\begin{aligned} M(R_1, R_2) &= R_1 \otimes M(\underline{1}, R_2) \\ &= R_1 \otimes \text{Project}(R_2) \\ \text{Project}(R(G, L, G', L')) &= (\exists L, L' : R) \wedge (L = L'). \end{aligned}$$

The Differential of Project. We extend the definition from §2 of the differential $\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{y})$ of a component function $f_i(\vec{x})$ with a case for Project:

$$\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{y}) = \text{Project}(\mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{y})) \qquad \text{if } f_i = \text{Project}(g).$$

EXAMPLE 8.3. *The application of the NPA linearizing transformation to Eqn. (51) creates the following equation system:*

$$\begin{aligned} Y_1 &= \begin{pmatrix} a \oplus (b \otimes \mathit{Project}(\underline{\nu_2}) \otimes \mathit{Project}(\underline{\nu_1})) \\ \oplus (b \otimes \mathit{Project}(\underline{\nu_2}) \otimes \mathit{Project}(Y_1)) \\ \oplus (b \otimes \mathit{Project}(Y_2) \otimes \mathit{Project}(\underline{\nu_1})) \end{pmatrix} \\ Y_2 &= \begin{pmatrix} c \oplus (d \otimes \mathit{Project}(\underline{\nu_3}) \otimes \mathit{Project}(\underline{\nu_2})) \\ \oplus (d \otimes \mathit{Project}(\underline{\nu_3}) \otimes \mathit{Project}(Y_2)) \\ \oplus (d \otimes \mathit{Project}(Y_3) \otimes \mathit{Project}(\underline{\nu_2})) \end{pmatrix} \\ Y_3 &= \begin{pmatrix} g \oplus (e \otimes \mathit{Project}(\underline{\nu_2}) \otimes f) \\ \oplus (e \otimes \mathit{Project}(Y_2) \otimes f). \end{pmatrix} \end{aligned} \tag{52}$$

Correctness. With the extension given here for local variables and in §6 for loops, the component functions of an equation system $L : \vec{X} = \vec{f}(\vec{X})$ can now contain both regular operators and occurrences of the operator Project. Let $\vec{X}^\star$ denote the least fixed-point of L. $\vec{X}^\star$ exists because we are working with an ω-continuous semiring. Our goal is to relate Kleene iterate $\vec{\kappa}^{(i)}$, Newton iterate $\vec{\nu}^{(i)}$, and $\vec{X}^\star$ as follows:

THEOREM 8.4. *For all i, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)} \sqsubseteq \vec{X}^\star$.*

Thm. 8.4 shows that each Newton iterate $\vec{\nu}^{(i)}$ is trapped between the corresponding Kleene iterate $\vec{\kappa}^{(i)}$ and the least solution $\vec{X}^\star$. Because successive Kleene iterates approach $\vec{X}^\star$, successive Newton iterates must also approach $\vec{X}^\star$.

Esparza et al. proved a similar theorem [7, Thm. 3.9] for an equation system over a general semiring, but without occurrences of Kleene-star and Project.

Merge Functions and NPA-TP[PA]. For NPA-TP[PA] domains with local variables, we will focus on a slightly different problem, which is that of computing a projection value for each variable in the original equation. Eqn. (49) now becomes

$$\begin{aligned} W_1 &= \text{Project}(X_1) = \text{Project}(a \oplus (b \otimes W_2 \otimes W_1)) \\ W_2 &= \text{Project}(X_2) = \text{Project}(c \oplus (d \otimes W_3 \otimes W_2)) \\ W_3 &= \text{Project}(X_3) = \text{Project}(g \oplus (e \otimes W_2 \otimes f)). \end{aligned} \tag{53}$$

In a program-analysis problem, the value of W_i serves as a summary of procedure X_i. Once the $\vec{W}$ values are in hand, one can obtain the $\vec{X}$ values by evaluating the right-hand side of the original equation.

The merge function for a tensor-transpose relational weight can be defined as follows:

$$\begin{aligned} M_{\mathcal{T}}(T_1, T_2) &= T_1 \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(T_2) \\ \text{Project}_{\mathcal{T}}(T(G_1', L_1', G_2, L_2, G_1, L_1, G_2', L_2')) & \\ = (\exists L_1', L_2, L_1, L_2' : T \wedge (L_1' = L_2)) & \\ \wedge (L1 = L1') \wedge (L2 = L2') & \end{aligned} \tag{54}$$

OBSERVATION 8.1. *The $\mathit{Project}_{\mathcal{T}}$ operation defined in Eqn. (54) has the properties*

$$\begin{aligned} \mathit{Project}_{\mathcal{T}}(\mathit{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} b) &= \mathit{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b) \\ \mathit{Project}_{\mathcal{T}}(a \oplus_{\mathcal{T}} b) &= \mathit{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b) \\ \mathit{Project}_{\mathcal{T}}(\mathit{Project}_{\mathcal{T}}(a)) &= \mathit{Project}_{\mathcal{T}}(a) \end{aligned}$$

The latter can be used to show that for X defined by $X = \mathit{Project}_{\mathcal{T}}(a \oplus_{\mathcal{T}} X \otimes_{\mathcal{T}} b)$, $\mathit{Project}_{\mathcal{T}}(X) = X$. These properties allow us to push occurrences of $\mathit{Project}_{\mathcal{T}}$ down to tensor-product-semiring constants, which we show by means of an example:

$$\begin{aligned} X &= \mathit{Project}_{\mathcal{T}}(a \oplus_{\mathcal{T}} X \otimes_{\mathcal{T}} b) \\ &= \mathit{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(X \otimes_{\mathcal{T}} b) \\ &= \mathit{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(\mathit{Project}_{\mathcal{T}}(X) \otimes_{\mathcal{T}} b) \\ &= \mathit{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(X) \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b) \\ &= \mathit{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} X \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b) \end{aligned}$$

□

The introduction of the Project$(\cdot)$ operator creates an impediment to applying Tarjan's algorithm, which is limited to equation systems over the standard regular operators. Fortunately, we are able to sidestep this difficulty because in an equation system like Eqn. (53) the locations of Project$(\cdot)$ are always associated with the bodies of procedures. Therefore, in steps 1 and 2 of Alg. 7.1, we work with an equation system with no occurrences of Project$(\cdot)$. After step 2, occurrences of $\text{Project}_{\mathcal{T}}$ can be introduced.

EXAMPLE 8.5. *For Eqn. (52), we would obtain the following equations:*

$$Z_1 = \begin{pmatrix} \mathit{Project}_{\mathcal{T}}(\underline{1} \odot (a \oplus (b \otimes \underline{\nu}_2 \otimes \underline{\nu}_1))) \\ \oplus_{\mathcal{T}} \; Z_1 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}((b \otimes \underline{\nu}_2)^t \odot \underline{1}) \\ \oplus_{\mathcal{T}} \; Z_2 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b^t \odot \underline{\nu}_1) \end{pmatrix}$$

$$Z_2 = \begin{pmatrix} \mathit{Project}_{\mathcal{T}}(\underline{1} \odot (c \oplus (d \otimes \underline{\nu}_3 \otimes \underline{\nu}_2))) \\ \oplus_{\mathcal{T}} \; Z_2 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}((d \otimes \underline{\nu}_3)^t \odot \underline{1}) \\ \oplus_{\mathcal{T}} \; Z_3 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(d^t \odot \underline{\nu}_2) \end{pmatrix}$$

$$Z_3 = \begin{pmatrix} \mathit{Project}_{\mathcal{T}}(\underline{1} \odot (g \oplus (e \otimes \underline{\nu}_2 \otimes f))) \\ \oplus_{\mathcal{T}} \; Z_2 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(e^t \odot f). \end{pmatrix}$$

Equivalently, one can wait until step 5 and use the following method to create map m: each variable Z_i, $1 \leq i \leq n$, is mapped to the regular expression

$$\begin{aligned} &\mathcal{R}_i[\langle 0, j\rangle \leftarrow \text{Project}_{\mathcal{T}}(\underline{1}^t \odot \text{Rhs}_j(\vec{\nu}))] \\ &\quad [\langle 1, j\rangle \leftarrow \text{Project}_{\mathcal{T}}(\text{Coeff}_1(\tau_{\text{Reg}}(\mathcal{D}_{X_1}\text{Rhs}_j|_{\vec{\nu}}(\vec{Y}))))] \\ &\quad \cdots \\ &\quad [\langle n, j\rangle \leftarrow \text{Project}_{\mathcal{T}}(\text{Coeff}_n(\tau_{\text{Reg}}(\mathcal{D}_{X_n}\text{Rhs}_j|_{\vec{\nu}}(\vec{Y}))))] \end{aligned}$$

EXAMPLE 8.6. *Consider again Ex. 7.2. The regular expression created for Z_2 in step 4 is $\langle 0, 2\rangle \oplus_{\mathcal{T}} \langle 0, 2\rangle(\langle 2, 2\rangle)^{*_{\mathcal{T}}}$. After step 5, $m(Z_2)$ becomes*

$$\begin{aligned} &\mathit{Project}_{\mathcal{T}}(\underline{1}^t \odot (d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c)) \\ \oplus_{\mathcal{T}} &\begin{pmatrix} \mathit{Project}_{\mathcal{T}}(\underline{1}^t \odot (d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c)) \\ \otimes_{\mathcal{T}} \left(\mathit{Project}_{\mathcal{T}}((b^t \odot (\underline{\nu}_2 \otimes c)) \oplus_{\mathcal{T}} ((b \otimes \underline{\nu}_2)^t \odot c))\right)^{*_{\mathcal{T}}} \end{pmatrix} \end{aligned}$$

Similarly, the regular expression created for Z_1 is $\langle 0, 1\rangle \oplus_{\mathcal{T}} (\langle 0, 2\rangle \oplus_{\mathcal{T}} \langle 0, 2\rangle(\langle 2, 2\rangle)^{*_{\mathcal{T}}})\langle 2, 1\rangle$, and $m(Z_1)$ is

$$\mathit{Project}_{\mathcal{T}}(\underline{1} \odot a\underline{\nu}_2) \oplus_{\mathcal{T}} m(Z_2) \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(a^t \odot \underline{1}).$$

9. Implementation and Experiments

The Implemented Solvers. We experimented with implementations of NPA and NPA-TP, along with two non-Newton solvers.

- One conventional solver used chaotic iteration (implemented using the post* algorithm for EWPDSs [16], followed by "path_summary" [23]). The other used an adaptation of Tarjan's path-expression algorithm [30] for interprocedural analysis (the post* algorithm for FWPDSs [15], followed by path_summary). We refer to these as "EWPDS" and "FWPDS," respectively.
- For the Newton solvers, we first applied Tarjan's path-expression algorithm to each CFG of the program to create a system of equations with regular right-hand sides. We then applied the differential operator (Defn. 2.4)—with the extensions presented in §6 and §8—and τ_{Reg} for the NPA-TP version. The NPA solver used FWPDS to solve each LCFL problem, whereas the NPA-TP solver used the steps given in Alg. 7.1.

EWPDS and FWPDS are standard solvers available in the Weighted Automaton Library (WALi) [12]; NPA and NPA-TP were implemented using primitives available in WALi.

OBDD Variable-Ordering Issues. The predicate-transformer relations of the predicate-abstraction domain are represented with Ordered Binary Decision Diagrams (OBDDs) [3]. As is well-known, the size of the OBDD for a Boolean function is sensitive to the order chosen for the Boolean variables.

Equation-Solving Experiments. Our experiments were designed to determine which method for solving a set of equations is the fastest. In particular, for solving predicate-abstraction problems,

1. How many Newton rounds do NPA and NPA-TP perform?
2. Is NPA faster than chaotic iteration?
3. Is NPA-TP faster than NPA?
4. Is NPA-TP faster than chaotic iteration?
5. What is the algorithm of choice?

	#Completed	#Timeouts	#Spaceouts	#Newton Rounds
EWPDS	495	16	73	N/A
FWPDS	483	32	69	N/A
NPA	290	142	152	3.38
NPA-TP	386	16	182	3.67

Table 1. Completion rates for the solvers, along with the average number of Newton rounds (completed runs only).

Our test suite consisted of 584 Boolean programs from the 3,366 Boolean programs distributed with Microsoft's Static Driver Verifier [27]. The test suite consisted of all of the programs for which any of the four analyzers took more than 1 second to run (prior to some optimizations implemented in the final week before submission). Timings were taken on a Dell OptiPlex 3020 with four Intel Core i5-4570 CPUs (3.20GHz), equipped with 16 GB of memory, running Windows 7 Enterprise 64-bit (6.1, Build 7601) SP1.

Results. Completion rates for the solvers are shown in Tab. 1. Note that NPA had significantly more timeouts, although somewhat fewer spaceouts than NPA-TP.

As one might expect, NPA and NPA-TP generally performed only a small number of Newton rounds: column 5 of Tab. 1 reports the average number of rounds (for completed runs only), including the final round needed to determine quiescence.

Figs. 5 and 6 present scatter plots that compare the times for running one solver against another. In each of the plots, the solver on the x-axis has better performance: there are more points in the upper-left triangle, and both geometric means are ≥ 1.

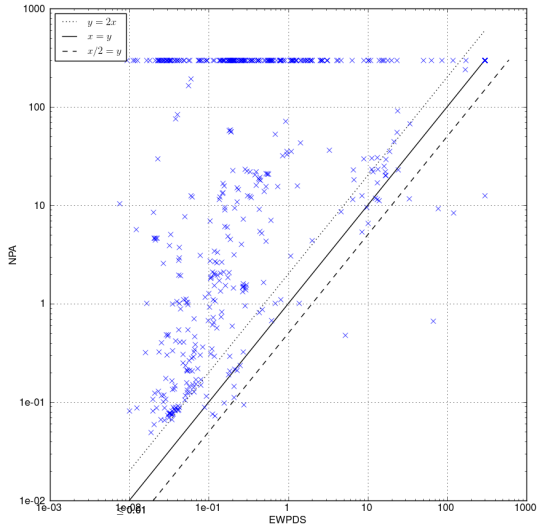
- Fig. 5(a) shows that chaotic iteration (EWPDS) performs far better than NPA (geometric means: $8.75 \rightarrow 31.6$). Thus, at least for this test suite of Boolean programs, these results answer Question 2 in the negative.
- Fig. 5(b) shows that the implementation of NPA-TP performs better than NPA (geometric means: $1.62 \rightarrow 4.61$). Thus, for this test suite, our results answer Question 3 in the positive: for Boolean programs, Alg. 7.1 succeeds in extending the capabilities of Newtonian Program Analysis.
- Fig. 5(c) shows that NPA-TP is still slower than chaotic iteration (geometric means: $5.20 \rightarrow 6.84$), which answers Question 4 in the negative. However, we see that NPA-TP did better against chaotic iteration than NPA did.
- Fig. 5(d) show that EWPDS is about 2x faster than FWPDS (geometric means: $2.22 \rightarrow 2.15$), although FWPDS is faster for some of the more compute-intensive problems.
- Fig. 6(a) shows that FWPDS is much faster than NPA (geometric means: $4.62 \rightarrow 14.7$).
- Fig. 6(b) shows that FWPDS is faster than NPA-TP (geometric means: $1.94 \rightarrow 3.19$), but again we see that NPA-TP did better against FWPDS than NPA did.

Overall, our results indicate that, among the four algorithms tested, the answer to Question 5 is that EWPDS is the algorithm of choice for predicate-abstraction problems.

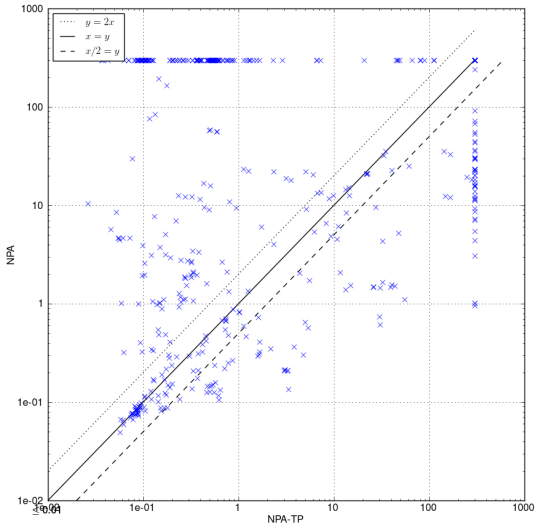
10. Related Work

To the best of our knowledge, this work is the first to consider the problem of solving LCFL equations on semirings. Yannakakis [33] considered the Boolean case (LCFL reachability). The technique of McNaughton and Yamada [20] for obtaining a regular expression that describes the paths in a finite labeled graph can be generalized from regular languages to LCFLs, but is much more costly than Tarjan's path-expression algorithm [30].

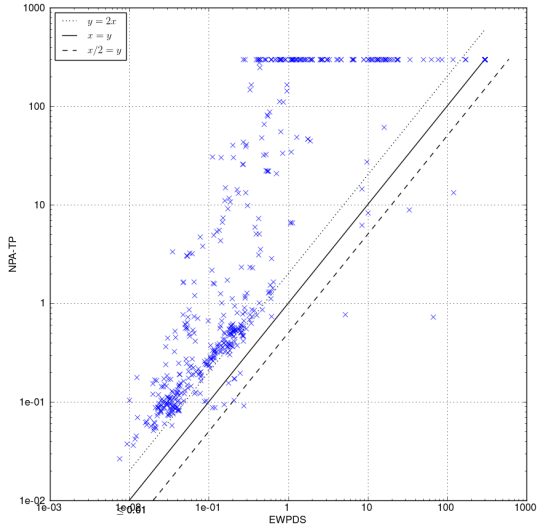
Tarjan's path-expression algorithm was used earlier by Lal and Reps [15] in a much more straightforward algorithm for interprocedural dataflow analysis. As in our method, they apply the path-



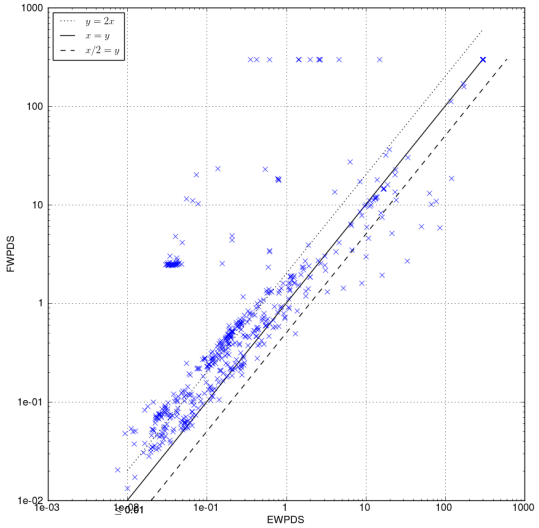
(a) EWPDS vs. NPA. Geometric means: $8.75 \rightarrow 31.6$



(b) NPA-TP vs. NPA. Geometric means: $1.62 \rightarrow 4.61$



(c) EWPDS vs. NPA-TP. Geometric means: $5.20 \rightarrow 6.84$



(d) EWPDS vs. FWPDS. Geometric means: $2.22 \rightarrow 2.15$

Figure 5. Log-log scatter plots of solver times on Boolean programs from SDV, with a 300-second timeout. (Spaceouts are also plotted at 300 seconds.) The solid diagonal line indicates equal performance; the dotted and dashed lines indicate 2x speedup/slowdown. For each plot, we report two geometric means of the Y/X values: (i) when Y and X both complete, and (ii) when non-completion counts as 300 seconds.

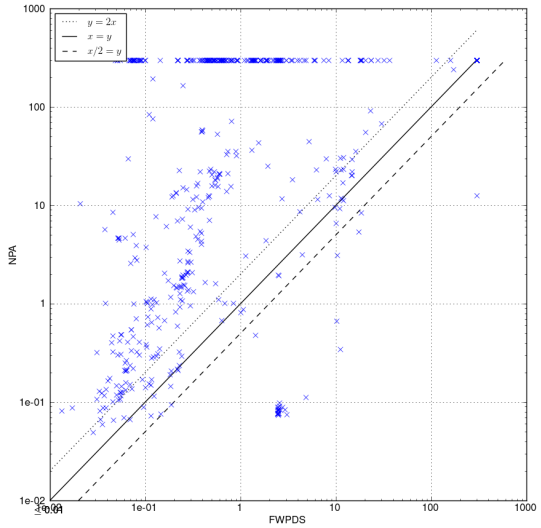
expression algorithm to each CFG of the program to create a system of recursive equations with regular right-hand sides. They then solve those equations directly via chaotic iteration. In contrast, NPA-TP converts the equation system to left-linear form, switching from $\mathcal{S}$ values to $\mathcal{S}_{\mathcal{T}}$ values in the process. Because the equation right-hand sides that are the input to this step can contain occurrences of Kleene-star, it was necessary for us to extend the NPA linearizing transformation, as described in §6.2. The resulting equation system is left-linear and Tarjan's algorithm is applied a second time to find a closed-form solution for each of the $\mathcal{S}_{\mathcal{T}}$-valued variables. The resulting regular expressions specify the computation that is performed on each Newton round.

The performance of Tarjan's path-expression algorithm can degenerate on non-reducible graphs. Although the graphs to which we apply the algorithm are not guaranteed to be reducible, in our experiments we found that the algorithm did not consume a significant amount of time.

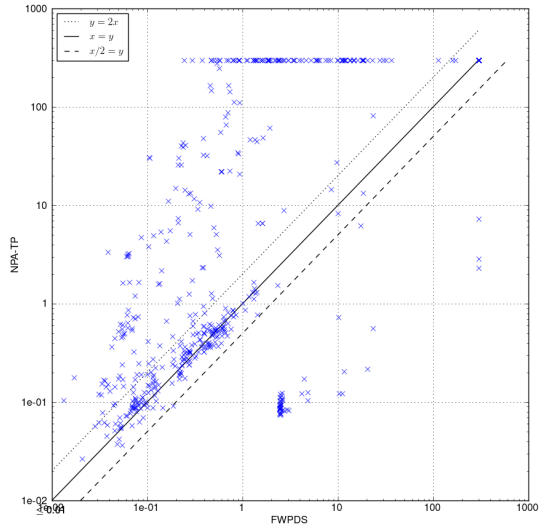
A different implementation of NPA is discussed in [25]. The two experiments reported were both with commutative semirings, for which NPA-TP is not needed.

Grathwohl et al. [10] developed an extension of Kleene algebra with tests to allow a finite amount of mutable state. They noted that one model of their extension could be represented using Kronecker products of 2×2 Boolean matrices, but did not make further use of that fact.

Tensor Product and Detensor-Transpose. Lal et al. [17] gave an algorithm for a variant of intersection of two weighted automata, which involved a side-condition on weight-products that can be formulated using an LCFL [17, §4]. The problem can be recast



(a) FWPDS vs. NPA. Geometric means: $4.62 \rightarrow 14.7$



(b) FWPDS vs. NPA-TP. Geometric means: $1.94 \rightarrow 3.19$

Figure 6. Experimental results (continued).

using an LCFL equation system to which the algorithm of the present paper can be applied.

Admissible semirings were used by Lal et al. [19] for context-bounded analysis of concurrent programs. Tensor product was used to support the intersection of weighted transducers. Analyses of different processes were performed independently, and the restructuring of values enabled by $\odot$ allowed the different analysis results to be stitched together. An operation similar to $\frac{1}{2}^{(t,\cdot)}$ was used to read out answers.

That work has a high-level point of similarity with our work, which might be termed the ***tensor-product principle***:

> Tensor products—plus an appropriate detensor operation—allow computations to be rearranged in certain ways; they can be used to delay performing every multiplication in a sequence of multiplications, which is useful if either (a) a value that is only obtainable at a later time needs to be placed in the middle of the sequence, or (b) a subsequence of values in the middle of the sequence needs to be adjusted in certain ways before contributing to the overall product.

In this paper, we use only one level of tensor products because that is all that is needed for "regularizing" an LCFL equation system. Lal et al. use $2k + 1$ levels of tensor products to capture $k + 1$ execution contexts and k context switches. Each execution context contributes a subsequence of values that must be reordered to compute the correct answer.

11. Conclusion

Our work attempted to unleash the promise of Newtonian program analysis. Our NPA-TP technique applies to equation systems over any semiring that meets the conditions of Defn. 4.1. The main technical result is a method to transform an LCFL equation system over semiring $\mathcal{S}$ into a left-linear—and hence regular—system of equations over a tensor-product semiring $\mathcal{S}_{\mathcal{T}}$. This transformation is both novel and surprising: formal-language theory tells us that LCFL $\supsetneq$ Regular, and the canonical example of a non-regular language, $\{b^i c^i \mid i \in \mathbb{N}\}$, is an LCFL. Nevertheless, we showed that there are non-commutative semirings for which we can apply such a transform with no loss of precision. We are not aware of any previous work that uses a similar "regularizing" transformation.

In addition, we showed how to extend Newtonian program analysis in two ways: (i) to handle loops via Kleene-star, and (ii) to handle local variables by means of merge functions.

The experiments, based on Boolean programs, show that NPA-TP is only a qualified success. Our work was motivated by the observation that standard NPA is slower than chaotic iteration (cf. Fig. 5(a)). Our goal of speeding up Newtonian program analysis was achieved (Fig. 5(b)); however, NPA-TP is still slower than EWPDS (Fig. 5(c)). NPA-TP is also slower than FWPDS (Fig. 6(b)), a more straightforward way of using Tarjan's algorithm for interprocedural dataflow analysis [15]. The head-to-head comparison of FWPDS with EWPDS shows that EWPDS is about 2x faster than FWPDS, although FWPDS is faster for some of the more compute-intensive problems (Fig. 5(d)). Overall, our results indicate that, among the four algorithms tested, EWPDS is the algorithm of choice for predicate-abstraction problems.

Acknowledgments

We thank Z. Kincaid for his help in finding an improved method for inserting Project$_{\mathcal{T}}$ operators when using NPA-TP to analyze programs with local variables (see §8); A. Lal for articulating what we have called the "tensor-product principle" (§10); and the anonymous reviewers for their feedback on the submission.

References

[1] T. Ball and S. Rajamani. Bebop: A symbolic model checker for Boolean programs. In *Spin Workshop*, 2000.

[2] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *POPL*, 2003.

[3] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, C-35(6):677–691, Aug. 1986.

[4] J. Cocke. Global common subexpression elimination. *Proc. Symp. on Compiler Optimization*, 1970.

[5] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In *Formal Descriptions of Programming Concepts*. North-Holland, 1978.

[6] J. Esparza, S. Kiefer, and M. Luttenberger. Newton's method for omega-continuous semirings. In *ICALP*, 2008.

[7] J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *J. ACM*, 57(6), 2010.

[8] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *CAV*, 1997.

[9] S. Graham and M. Wegman. A fast and usually linear algorithm for data flow analysis. *J. ACM*, 23(1):172–202, 1976.

[10] N. Grathwohl, D. Kozen, and K. Mamouras. KAT + B! In *CSL-LICS*, 2014.

[11] J. Kam and J. Ullman. Global data flow analysis and iterative algorithms. *J. ACM*, 23(1):158–171, 1976.

[12] N. Kidd, A. Lal, and T. Reps. WALi: The Weighted Automaton Library, 2007. www.cs.wisc.edu/wpis/wpds/download.php.

[13] G. Kildall. A unified approach to global program optimization. In *POPL*, 1973.

[14] J. Knoop and B. Steffen. The interprocedural coincidence theorem. In *CC*, 1992.

[15] A. Lal and T. Reps. Improving pushdown system model checking. In *CAV*, 2006.

[16] A. Lal, T. Reps, and G. Balakrishnan. Extended weighted pushdown systems. In *CAV*, 2005.

[17] A. Lal, N. Kidd, T. Reps, and T. Touili. Abstract error projection. In *Static Analysis Symp.*, 2007.

[18] A. Lal, T. Touili, N. Kidd, and T. Reps. Interprocedural analysis of concurrent programs under a context bound. Tech. Rep. TR-1598, Comp. Sci. Dept., Univ. of Wisconsin, Madison, WI, July 2007.

[19] A. Lal, T. Touili, N. Kidd, and T. Reps. Interprocedural analysis of concurrent programs under a context bound. In *TACAS*, 2008.

[20] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. on Elec. Computers*, 9:39–47, 1960.

[21] M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *POPL*, 2004.

[22] M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. In *ESOP*, 2005.

[23] T. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *SCP*, 58(1–2):206–263, Oct. 2005.

[24] T. Reps, A. Lal, and N. Kidd. Program analysis using weighted pushdown systems. In *FSTTCS*, 2007.

[25] M. Schlund, M. Terepeta, and M. Luttenberger. Putting Newton into practice: A solver for polynomial equations over semirings. In *LPAR*, 2013.

[26] M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Applications*. Prentice-Hall, 1981.

[27] Static Driver Verifier. Static driver verifier. msdn.microsoft.com/en-us/library/windows/hardware/ff552808(v=vs.85).aspx.

[28] R. Tapia. Inverse, shifted inverse, and Rayleigh quotient iteration as Newton's method, 2008. www.frequency.com/video/lecture-series-/18347021.

[29] R. Tarjan. A unified approach to path problems. *J. ACM*, 28(3):577–593, 1981.

[30] R. Tarjan. Fast algorithms for solving path problems. *J. ACM*, 28(3): 594–614, 1981.

[31] J. Ullman. Fast algorithms for the elimination of common subexpressions. *Acta Inf.*, 2:191–213, 1973.

[32] V. Vyssotsky and P. Wegner. A graph theoretical Fortran source language analyzer. Unpublished technical report, Bell Labs, Murray-Hill NJ (as cited in Aho et al., "Compilers: Principles, Techniques, and Tools", Addison-Wesley, 1986), 1963.

[33] M. Yannakakis. Graph-theoretic methods in database theory. In *PODS*, 1990.