Newtonian Program Analysis via Tensor Product

THOMAS REPS, University of Wisconsin and GrammaTech, Inc.
EMMA TURETSKY, University of Wisconsin
PRATHMESH PRABHU, Google, Inc.

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 in numerical-analysis problems, ***commutativity of multiplication*** is relied on to rearrange an expression of the form "$a{*}X{*}b{+}c{*}X{*}d$" into "$(a{*}b{+}c{*}d){*}X$." Equations with such expressions 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: "$a{*}X{*}b{+}c{*}X{*}d$" cannot be rearranged into "$(a{*}b{+}c{*}d){*}X$." Equations with such expressions 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

Additional Key Words and Phrases: Newton's method, polynomial fixed-point equation, interprocedural program analysis, semiring, regular expression, tensor product

E. Turetsky is currently affiliated with GrammaTech, Inc.
Portions of this work appeared in the 2016 ACM SIGPLAN-SIGACT Symposium on Principles of Program Analysis [Reps et al. 2016].
This work was 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.
Thomas Reps has an ownership interest in GrammaTech, Inc., which has licensed elements of the technology reported in this publication.
Corresponding author's address: Thomas Reps, Computer Sciences Department, University of Wisconsin, 1210 W. Dayton St., Madison, WI 53703, and GrammaTech, Inc., 440 Science Drive, Suite 303, Madison, WI 53711; e-mail: reps@cs.wisc.edu.

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 [Bouajjani et al. 2003; Reps et al. 2005; Reps et al. 2007]. Standard methods for obtaining the solution to such an equation system are based on ***Kleene iteration***, a successive-approximation method defined as follows:

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

Recently, Esparza et al. [2008], [2010] 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 a successive-approximation method, but uses the following iterative scheme:[1]

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

where $\mathrm{LinearCorrectionTerm}(\vec{f}, \vec{\nu}^{(i)})$ 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. In essence, the insight behind the work of Esparza et al. is that the high-level principle of Newton's method, namely,

> repeatedly, create a linear model of the function and use it to find a better approximation of the solution

can be applied to programs, too. The sense in which the correction term in Eqn. (2) 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 "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 [Tapia 2008] 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.

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

[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.

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 [Vyssotsky and Wegner 1963; Kildall 1973; Kam and Ullman 1976; Graham and Wegman 1976; Tarjan 1981b]. 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 "$a*X*b+c*X*d$" into "$a*b*X+c*d*X$," which equals "$(a*b+c*d)*X$." 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) [Cousot and Cousot 1978; Sharir and Pnueli 1981]. Consequently, the "multiplication" operation is typically the reversal of function composition—$v_1 * v_2 \overset{\text{def}}{=} v_2 \circ v_1$—which is not a commutative operation. When NPA is used with a non-commutative semiring, an expression of the form "$a * X * b + c * X * d$" in the linearized problem cannot be rearranged: coefficients remain on both sides of variable X.

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 "$a * X * b + c * X * d$" 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 [Tarjan 1981a] 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 *.[3]

[3]We will continue to refer to Tarjan's path-expression method in the remainder of the paper as a shorthand for "an algorithm to find a regular expression for each of the variables in a set of mutually recursive left-linear equations"—or, equivalently, "given a control-flow graph (CFG) G, find a collection of regular expressions $\{\mathcal{R}_n \mid n \in \text{Nodes}(G)\}$ such that each language $L(\mathcal{R}_n)$ consists of all paths in G from the entry node of G to node n." However, there are several other algorithms that could be used to create such a set of regular expressions, including generalized Floyd-Warshall [Droste et al. 2009], recursive splitting [Droste et al. 2009], and LDU decomposition [Litvinov et al. 2013] (see Schlund [2016, §4.1.3]).

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, $\{a^i b^i \mid i \in \mathbb{N}\}$, 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*** (§4.6). Moreover, as discussed in §5, one of the families of semirings for which our transformation applies is the set of ***predicate-abstraction domains*** [Graf and Saïdi 1997], which are the foundation of most of today's software model checkers.[4]

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 (§4.6 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).

§2 summarizes background material on which our work builds. §3 motivates the approach presented in the paper by demonstrating a version of our method on a simple problem. §10 discusses related work. §11 draws some conclusions. The proofs of correctness for our extensions of NPA and the different variants of NPA-TP discussed in the paper are found in Apps. A–C.

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 $\otimes$*

[4]Two other classes of semirings for which the transformation applies are based on abstract domains of affine relations [Müller-Olm and Seidl 2004; 2005]; see [Lal et al. 2007, §6.2].

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 $\sqsubseteq \overset{\text{def}}{=} \{(a,b) \in D \times D \mid \exists d \colon a \oplus d = b\}$ 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*

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

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:

$$\begin{aligned} b \otimes \left(\bigoplus_{i\in\mathbb{N}} a_i\right) &= \bigoplus_{i\in\mathbb{N}} (b \otimes a_i) \\ \left(\bigoplus_{i\in\mathbb{N}} a_i\right) \otimes b &= \bigoplus_{i\in\mathbb{N}} (a_i \otimes b) \\ \bigoplus_{j\in J}\left(\bigoplus_{i\in I_j} a_i\right) &= \bigoplus_{i\in\mathbb{N}} a_i \end{aligned}$$

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*** *$^* \colon D \to D$ 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 called the relational weight domain:

DEFINITION 2.2. *If A is a finite set, the* ***relational weight domain*** *on A is defined as $(\mathcal{P}(A \times A), \cup, ;, \emptyset, id)$. A weight $R \subseteq A \times A$ is a binary relation on A,*[5] *$\oplus$ is union ($\cup$), $\otimes$ 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.*

Defn. 2.2 gives us a way to formalize each predicate-abstraction domain as a semiring. A Boolean program is a program whose only datatype is Boolean. A Boolean program P can be used as an abstraction of a real-world program [Ball and Rajamani 2000] via predicate abstraction [Graf and Saïdi 1997]. For each predicate p, there is a variable $v_p \in \text{Var}$ in the Boolean program, which holds the value of p in states of the program being modeled. A state of the Boolean program is an assignment in $\text{Var} \to \text{Bool}$.

By instantiating A in Defn. 2.2 to be the set of assignments $\text{Var} \to \text{Bool}$, we obtain a semiring whose values can encode the state-transformers of P: the semiring value

[5]A weight can also be thought of as a Boolean matrix with dimensions $|A| \times |A|$.

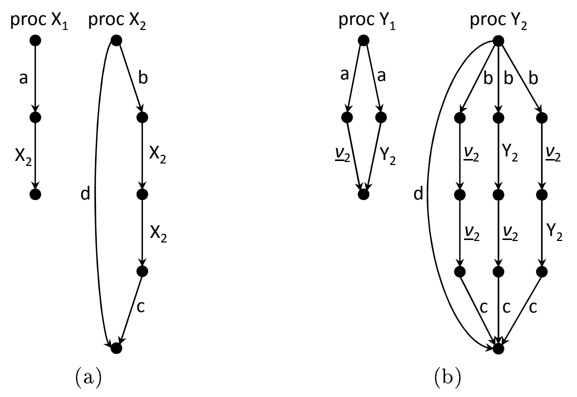


Fig. 1. (a) Graphical depiction of the equation system given in Eqn. (3) as an interprocedural control-flow graph. The three edges labeled "X_2" represent calls to procedure X_2. (b) Linearized equation system over $\vec{Y}$ obtained from Eqn. (3) via Eqn. (5); see Ex. 2.6.

associated with an assignment statement or assume statement `st` of P is the binary relation on A that represents the effect of `st` on the state of P. (See §5.)

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 2.3. *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. [2008], [2010] 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 2.4. *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:*

```
$X_1()$ {                    $X_2()$ {
    $s_a$;                       if ($\star$)  $s_d$
    $X_2()$                      else {
}                                    $s_b$; $X_2()$; $X_2()$; $s_c$
                                 }
                             }
```

Suppose that we have a semiring whose elements correspond to some abstraction of the state-transformers of the programming language. (An example of such a semiring is the relational weight domain for a predicate-abstraction domain.) Let a, b, c, and d denote the semiring elements that abstract the actions of 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:

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

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

Each unknown in an equation system represents an abstract transformer that serves as a ***procedure summary****, in the sense of Sharir and Pnueli [1981] and Reps et al. [1995]. A summary for procedure X_i overapproximates the behavior of X_i, including all procedures called transitively from X_i. Once procedure summaries have been obtained, one can use them to analyze each procedure X_j to obtain an abstract value $V_{j,p}$ for each program point p in X_j [Sharir and Pnueli 1981]. $V_{j,p}$ represents a superset of the states that can arise at p. To find potential bugs, for instance, one needs to determine if any bad states can arise at p, which can be done by checking whether any bad states are contained within the meaning of $V_{j,p}$—in abstract-interpretation terminology, by checking whether $(\gamma(V_{j,p}) \cap Bad) \neq \emptyset$.*

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 monomial that consists of a single 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 polynomial is linear if all of its monomials are linear. A system of polynomial equations has the form

$$\begin{aligned} X_1 &= f_1(X_1, \ldots, X_n) \\ &\cdots \\ X_n &= f_n(X_1, \ldots, X_n), \end{aligned}$$

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),

$$\vec{f} \overset{\text{def}}{=} \lambda \vec{X}.\langle a \otimes X_2, d \oplus b \otimes X_2 \otimes X_2 \otimes c \rangle.$$

The jumping-off point for Esparza et al. is the observation that in numerical problems the workhorse for successive-approximation algorithms is Newton's method. Fig. 2 shows how Newton's method can (sometimes) help identify where a root of a function lies. (Newton's method is not guaranteed to converge to a root.) The

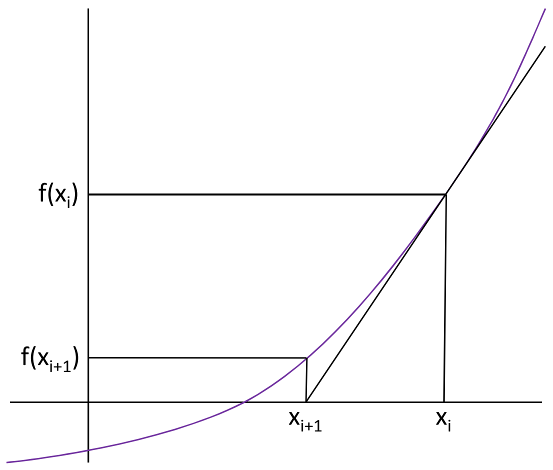


Fig. 2. The principle behind Newton's method for finding roots of real-valued functions.

general principle is to create a linear model of the function—in this case the tangent line—and solve the problem for the linear model to obtain the next approximation to the root.

Compared to the numerical setting, Esparza et al. had two points that they needed to finesse:

(1) With numerical functions, the linear model is defined using derivatives, which are defined in terms of limits. We have no analogues of such entities in semirings.

(2) Newton's method is for root-finding (i.e., find x such that $f(x) = 0$), whereas in program analysis we are interested in finding a fixed-point (i.e., find x such that $f(x) = x$). Although one can easily convert a fixed-point problem into a root-finding problem—find x such that $f(x) - x = 0$—this approach creates a new problem because there is no analogue of a subtraction operation in a semiring.

Kleene iteration is the well-known technique for finding the least fixed-point of $\vec{X} = \vec{f}(\vec{X})$ via the sequence $\vec{\kappa}^{(0)} = \underline{\vec{0}}$; $\vec{\kappa}^{(i+i)} = \vec{f}(\vec{\kappa}^{(i)})$. The NPA method of Esparza et al. [2008], [2010] provides an alternative method for finding the least fixed-point of $\vec{X} = \vec{f}(\vec{X})$. With NPA, one solves the following sequence of problems for $\vec{\nu}$:

$$\boxed{\begin{array}{rcl} \vec{\nu}^{(0)} & = & \vec{f}(\underline{\vec{0}}) \\ \vec{\nu}^{(i+1)} & = & \vec{Y}^{(i)} \end{array}} \tag{4}$$

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. 2.5). 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 $\mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y})$.[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 by the transformation given in Defn. 2.5. It converts a system of equations with polynomial right-hand sides into a new equation system in which each equation's right-hand side is linear.

Definition 2.5. *[Esparza et al. 2008; 2010] 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:*

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

*where $K \subseteq \mathbb{N}$ is some finite or infinite index set. Let $\vec{f}$ be a multivariate polynomial function defined by $\vec{f} \overset{\text{def}}{=} \lambda \vec{X}.\langle f_1(\vec{X}), \ldots, f_n(\vec{X})\rangle$. 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:*

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

$\mathcal{D} f_i|_{\underline{\vec{\nu}}}(\vec{Y}) \overset{\text{def}}{=} \bigoplus_{k=1}^{n} \mathcal{D}_{X_k} f_i|_{\underline{\vec{\nu}}}(\vec{Y})$ denotes the i^{th} component of $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$.

The fourth case in Eqn. (7) generalizes the differential of a binary combine, i.e.,

$$\mathcal{D}_{X_j} g_1|_{\underline{\vec{\nu}}}(\vec{Y}) \oplus \mathcal{D}_{X_j} g_2|_{\underline{\vec{\nu}}}(\vec{Y}) \qquad \text{if } f_i = g_1 \oplus g_2,$$

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

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

the combinations Eqns. (4) and (5) and Eqns. (4) and (6) produce the same set of iterates $\vec{\nu}^{(0)}, \vec{\nu}^{(1)}, \ldots, \vec{\nu}^{(i)}, \ldots$ [Esparza et al. 2010, 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), and we prove that the property of coinciding iteration sequences carries over to the extensions of NPA that we make in this paper—see Thm. 8.6.

to infinite combines. Note how the fifth case, for "$g \otimes h$", 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:

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

An inductive argument can be used to show that $\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y})$ yields a sum (combine) of terms, each of which is linear in $\vec{Y}$. The core of the argument is that in "$\mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes h(\underline{\vec{\nu}}) \oplus g(\underline{\vec{\nu}}) \otimes \mathcal{D}_{X_j} h|_{\underline{\vec{\nu}}}(\vec{Y})$," the introduction of $\underline{\vec{\nu}}$ in $h(\underline{\vec{\nu}})$ in the first summand ensures that no variables appear in the h multiplicand, and likewise for $g(\underline{\vec{\nu}})$ in the second summand. Consequently, the inductive hypothesis, applied to $\mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y})$ and $\mathcal{D}_{X_j} h|_{\underline{\vec{\nu}}}(\vec{Y})$, shows that the result of the $f_i = g \otimes h$ case is a sum of linear terms.

The multivariate differential defined in Defn. 2.5 is how Esparza et al. addressed the issue raised in item (1) above: the multivariate differential is a formal operator on a polynomial expression over a semiring, and does not involve any notion of "the limit as ΔX approaches 0."

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

EXAMPLE 2.6. *For Eqn. (3), the multivariate differential of $\vec{f}$ at the value $\underline{\vec{\nu}} = \langle \underline{\nu}_1, \underline{\nu}_2 \rangle$ is*

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

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 $\langle \nu_1^{(i+1)}, \nu_2^{(i+1)} \rangle \leftarrow \langle Y_1, Y_2 \rangle$.

NPA can be thought of as a kind of sampling method for the state space of the program. For instance, in Ex. 2.4, procedure X_2 has two call-sites. In the corresponding linearized program in Fig. 1(b), each path through Y_2 has at most one call site: the NPA linearizing transformation inserted the value $\underline{\nu}_2$ at various call-sites, *and left at most one variable in each summand. In essence, during a given Newton round the analyzer samples the state space of Y_2 by taking the $\oplus$ of various paths through Y_2. Along each such path, the abstract values for the call-sites encountered are held fixed at $\underline{\nu}_2$, except for possibly one call-site on the path, which is explored by visiting (the linearized version of) the called procedure. The abstract values for $\underline{\nu}_1$ and $\underline{\nu}_2$ are updated according to the results of this state-space exploration, and the algorithm proceeds to the next Newton round.*

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 [Esparza et al. 2010, 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 [Cocke 1970; Ullman 1973; Graham and Wegman 1976] and Tarjan's path-expression method [Tarjan 1981b; 1981a]. Tarjan's method has also been harnessed for ***inter***procedural analysis [Lal and Reps 2006].

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 §4.6).

3.1 Linear, Non-Regular, Equation Systems

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

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

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 non-trivial 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 $\mathcal{L}$, devise an efficient method for finding the least solution of $\mathcal{L}$.

Definition 3.2. *An LCFL 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 (Defn. 3.1), 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 [Tarjan 1981a] provides a fast method for solving left-linear equation systems led us to pose the following question:

$$\boxed{\text{Is it possible to "regularize" the LCFL equation system } \mathcal{L} \text{ that arises on each Newton round---i.e., transform } \mathcal{L} \text{ into a left-linear equation system } \mathcal{L}_{\mathrm{Reg}}\text{?}}$$

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 $L = \{b^i c^i \mid i \in \mathbb{N}\}$, which is the canonical example of an LCFL that is not regular. 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

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

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

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

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 $\mathcal{R}(a, b) \overset{\text{def}}{=} a \otimes b$. 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*

$$\begin{aligned} Z_2 &= A \oplus_p Z_2 \otimes_p B, & (18) \\ \textit{where} \quad A &= (\underline{1}, d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c), & (19) \\ \textit{and} \quad B &= (b \oplus b \otimes \underline{\nu}_2, \underline{\nu}_2 \otimes c \oplus c). & (20) \end{aligned}$$

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 $\bigoplus_{i \in \mathbb{N}}{}_{\mathrm{p}}\, B^i$ (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 $\mathcal{L}$,*

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

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 $\mathcal{L}_{\mathrm{Reg}}$ has a larger number of variables, for step (2) we can use Tarjan's path-expression method [Tarjan 1981a], 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 $\mathcal{L}_{\mathrm{Reg}}$—***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

$$\langle Z_1, Z_2 \rangle = \left\langle \begin{array}{l} (\underline{1}, a \otimes \underline{\nu}_2) \oplus_p Z_2 \otimes_p (a, \underline{1}), \\ A \oplus_p Z_2 \otimes_p B \end{array} \right\rangle,$$

for which we have the closed-form solution

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

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:

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

(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.

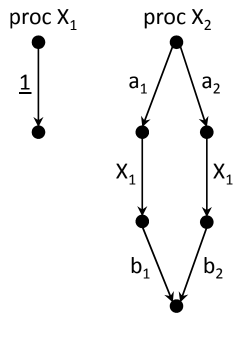
   (b) Use that value in Eqns. (19) and (20), and evaluate the right-hand side of Eqn. (22) to obtain new values for Z_1 and Z_2.

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 $\mathcal{L}$. 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 exactly two paths:



The least solution for X_2 is $a_1 b_1 \oplus a_2 b_2$, where $a_1 b_1$ and $a_2 b_2$ are the contributions from the two paths. However, when treated as a paired-semiring-value problem, we have

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

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 $a_1 b_1 \oplus a_2 b_2$.

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

In other words, using combines of pairs leads to cross-terms, such as $a_2 b_1$ and $a_1 b_2$, 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 $\mathcal{L}$.

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

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

and hence,

	Eqn. (12)-term?	i	$\#\underline{\nu}_2 + 2\#d$
$\mathcal{R}(AB) = bd\underline{\nu}_2 c$	✓	1	3
$\oplus\ bdc$	✗	n/a	2
$\oplus\ bb\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c$	✓	1	3
$\oplus\ bb\underline{\nu}_2\underline{\nu}_2 cc$	✗	n/a	2
$\oplus\ b\underline{\nu}_2 d\underline{\nu}_2 c$	✗	n/a	4
$\oplus\ b\underline{\nu}_2 dc$	✓	1	3
$\oplus\ b\underline{\nu}_2 b\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c$	✗	n/a	4
$\oplus\ b\underline{\nu}_2 b\underline{\nu}_2\underline{\nu}_2 cc$	✓	1	3

(25)

Of the eight terms on the right-hand side of $\mathcal{R}(AB)$, only four meet the conditions of Eqn. (12): $bd\underline{\nu}_2 c$, $bb\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c$, $b\underline{\nu}_2 dc$, and $b\underline{\nu}_2 b\underline{\nu}_2\underline{\nu}_2 cc$. 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).

Discussion. It is worthwhile reiterating what the example above demonstrates. The example involves a lot of expressions, but that is merely because we did not base the example on a specific semiring. A semiring $\mathcal{S} = (D, \oplus, \otimes, \underline{0}, \underline{1})$ supplies an interpretation for $\oplus$ and $\otimes$, and thus each (variable-free) expression in the example would actually be a specific semiring value. In particular, the closed-form solution for Z_2 given in Eqn. (21) would give a value for Z_2—say $Z_p \in D \times D$—from which we would read out the semiring value $\mathcal{R}(Z_p) \in D$.

We chose to present the derivation leading up to Eqn. (25) *symbolically*—rather than for a specific semiring—for the following reason: the expressions obtained for AB and $\mathcal{R}(A \otimes_p B)$ allow one to understand exactly how the properties of the operators that work on pairs contribute undesirable cross-terms that cause $\mathcal{R}(A \otimes_p B^{*_p})$ to overapproximate the least-fixed-point solution.

In terms of the steps of Alg. 3.4, the source of the difficulty is not so much the $\mathcal{R}$ operation of step (3) *per se*. As we will see, the key issue is actually the choice in step (1) to use weights that consist of *pairs* of semiring values.

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:*

$$(a_1 \oplus a_2)^t = a_1^t \oplus a_2^t \tag{26}$$
$$(a_1 \otimes a_2)^t = a_2^t \otimes a_1^t \tag{27}$$
$$(a^t)^t = a. \tag{28}$$

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:*

$$\underline{0} \odot a = a \odot \underline{0} \quad = \quad \underline{0}_\mathcal{T} \tag{29}$$
$$a_1 \odot (b_2 \oplus c_2) = (a_1 \odot b_2) \oplus_\mathcal{T} (a_1 \odot c_2) \tag{30}$$
$$(b_1 \oplus c_1) \odot a_2 = (b_1 \odot a_2) \oplus_\mathcal{T} (c_1 \odot a_2) \tag{31}$$
$$(a_1 \odot b_1) \otimes_\mathcal{T} (a_2 \odot b_2) = (a_1 \otimes a_2) \odot (b_1 \otimes b_2). \tag{32}$$

A tensor-product semiring defined over a semiring with transpose has a ***(sequential) detensor-transpose*** *operation, denoted by $\lightning^{(t,\cdot)} : D_\mathcal{T} \to D$, if for all ele-*

ments $a_1, a_2 \in D$ and $p_1, p_2 \in D_{\mathcal{T}}$ the following properties hold:

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

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

We assume that Eqns. (26), (30), (31), and (34) also hold for infinite combines. In particular, for $\lightning^{(t,\cdot)}$ we have

$$\lightning^{(t,\cdot)}\left(\bigoplus_{i \in I}{}_{\mathcal{T}}\, p_i\right) = \bigoplus_{i \in I} \lightning^{(t,\cdot)}(p_i). \tag{35}$$

*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. (27) and (32),

$$\begin{aligned} \mathcal{C}(a_1, b_1) \otimes_{\mathcal{T}} \mathcal{C}(a_2, b_2) &= (a_1^t \odot b_1) \otimes_{\mathcal{T}} (a_2^t \odot b_2) \\ &= (a_1^t \otimes a_2^t) \odot (b_1 \otimes b_2) \\ &= (a_2 \otimes a_1)^t \odot (b_1 \otimes b_2) \\ &= \mathcal{C}(a_2 \otimes a_1, b_1 \otimes b_2) \end{aligned} \tag{36}$$

The order-reversal vis à vis $\otimes_{\mathcal{T}}$ and $\otimes$ in Eqn. (36) 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, in contrast with Eqn. (24), $\lightning^{(t,\cdot)}$ does not produce cross-terms because of property (34):

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

In general, we require $\lightning^{(t,\cdot)}$ to distribute over infinite combines—cf. Eqn. (35). That property guarantees that the "readout" value produced by $\lightning^{(t,\cdot)}$ is the combine-over-all-LCFL-paths of the readout value of each individual path. (See also Remark 4.16.)

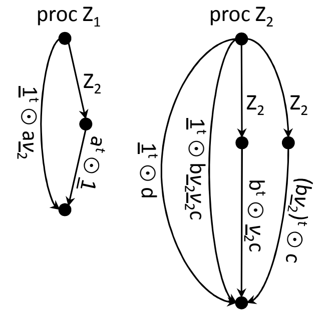


Fig. 3. Graphical representation of Eqn. (38), the linearized equation system over $\vec{Z}$ obtained from Eqn. (10) via Defn. 4.2.

4.2 The Regularizing Transformation

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

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

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:

$$\tau_{Reg}\left(c_j \oplus \bigoplus_{i,k} (a_{i,j,k} \otimes Y_i \otimes b_{i,j,k})\right) \overset{\text{def}}{=} (\underline{1}^t \odot c_j) \oplus_\mathcal{T} \bigoplus_{i,k}{}_\mathcal{T} (Z_i \otimes_\mathcal{T} (a^t_{i,j,k} \odot b_{i,j,k})). \tag{37}$$

We use $Coeff_i(\cdot)$ to select Z_i's coefficient in Eqn. (37):

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

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

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

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

$$Z_2 = \begin{pmatrix} & (\underline{1}^t \odot d) \\ \oplus_\mathcal{T} & (\underline{1}^t \odot b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c) \\ \oplus_\mathcal{T} & Z_2 \otimes_\mathcal{T} (b^t \odot (\underline{\nu}_2 \otimes c)) \\ \oplus_\mathcal{T} & Z_2 \otimes_\mathcal{T} ((b \otimes \underline{\nu}_2)^t \odot c) \end{pmatrix} \tag{38}$$

which is depicted in Fig. 3. After collecting terms, we have

$$Z_2 = A \oplus_\mathcal{T} (Z_2 \otimes_\mathcal{T} B), \tag{39}$$

$$\begin{aligned} \textit{where} \quad A &= \left(\underline{1}^t \odot (d \oplus b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c)\right) \\ \textit{and} \quad B &= \left(b^t \odot (\underline{\nu}_2 \otimes c)\right) \oplus_{\mathcal{T}} \left((b \otimes \underline{\nu}_2)^t \odot c\right) \end{aligned} \tag{40}$$

4.3 Solving an LCFL Equation System

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

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

(1) Apply τ_{Reg} to $\mathcal{L}$ to create the left-linear equation system $\mathcal{L}_{\mathcal{T}}$ over the tensor-product semiring $\mathcal{S}_{\mathcal{T}}$.[7]

(2) Use Tarjan's path-expression algorithm to find a regular expression Reg_i for each variable Z_i in $\mathcal{L}_{\mathcal{T}}$.

(3) Obtain $\vec{Z}$, the least solution to $\mathcal{L}_{\mathcal{T}}$: 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 $\between^{(t,\cdot)}$ to each component of $\vec{Z}$ to obtain the solution to the original LCFL equation system $\mathcal{L}$; i.e., $Y_i \leftarrow \between^{(t,\cdot)}(Z_i)$.

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:*[8]

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

Given a vector of values $\vec{\nu}$, the value of a generalized regular expression is obtained in the expected manner, shown below, where $(\!|op|\!)$ denotes the interpretation of op

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

[8]The case for $exp ::= exp^*$ is irrelevant in this section; however, it is needed for the version of NPA-TP discussed in §6.2.

in $\mathcal{S}$ or $\mathcal{S}_{\mathcal{T}}$, as appropriate:

$$
\llbracket e \rrbracket_{\mathcal{T}} \vec{\nu} \overset{\text{def}}{=} \begin{cases} a_{\mathcal{T}} & \textit{if } e = a_{\mathcal{T}} \in \mathcal{S}_{\mathcal{T}} \\ (\llbracket e_1 \rrbracket \vec{\nu})^{\llparenthesis t \rrparenthesis} \llparenthesis \odot \rrparenthesis \llbracket e_2 \rrbracket \vec{\nu} & \textit{if } e = e_1^t \odot e_2 \\ \llbracket e_1 \rrbracket_{\mathcal{T}} \vec{\nu} \llparenthesis \oplus_{\mathcal{T}} \rrparenthesis \llbracket e_2 \rrbracket_{\mathcal{T}} \vec{\nu} & \textit{if } e = e_1 \oplus_{\mathcal{T}} e_2 \\ \llbracket e_1 \rrbracket_{\mathcal{T}} \vec{\nu} \llparenthesis \otimes_{\mathcal{T}} \rrparenthesis \llbracket e_2 \rrbracket_{\mathcal{T}} \vec{\nu} & \textit{if } e = e_1 \otimes_{\mathcal{T}} e_2 \\ (\llbracket e_1 \rrbracket_{\mathcal{T}} \vec{\nu})^{\llparenthesis * _{\mathcal{T}} \rrparenthesis} & \textit{if } e = (e_1)^{*_{\mathcal{T}}} \end{cases}
$$

$$
\llbracket e \rrbracket \vec{\nu} \overset{\text{def}}{=} \begin{cases} a & \textit{if } e = a \in \mathcal{S} \\ (\vec{\nu})_i & \textit{if } e = \underline{\nu}_i \\ \llbracket e_1 \rrbracket \vec{\nu} \llparenthesis \oplus \rrparenthesis \llbracket e_2 \rrbracket \vec{\nu} & \textit{if } e = e_1 \oplus e_2 \\ \llbracket e_1 \rrbracket \vec{\nu} \llparenthesis \otimes \rrparenthesis \llbracket e_2 \rrbracket \vec{\nu} & \textit{if } e = e_1 \otimes e_2 \\ (\llbracket e_1 \rrbracket \vec{\nu})^{\llparenthesis * \rrparenthesis} & \textit{if } e = (e_1)^{*} \end{cases}
$$

4.4 Discussion

It is instructive to consider what happens when Alg. 4.4 is applied to the LCFL equation system that was the subject of Exs. 2.6 and 4.3.

—In step 2 of Alg. 4.4, the regular expression that would be obtained for variable Z_2—defined in Eqn. (39)—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 $\otimes_{\mathcal{T}}$.

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

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

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

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

$$
\begin{aligned}
\lightning^{(t,\cdot)}(AB) &= \lightning^{(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} \\
&= \lightning^{(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} && (41) \\
&= \lightning^{(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} && (42) \\
&= \lightning^{(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} \\
&= \lightning^{(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} & \lightning^{(t,\cdot)}(b^t \odot d\underline{\nu}_2 c) \oplus_{\mathcal{T}} \lightning^{(t,\cdot)}(b^t \odot b\underline{\nu}_2\underline{\nu}_2 c\underline{\nu}_2 c) \\ \oplus_{\mathcal{T}} & \lightning^{(t,\cdot)}((b\underline{\nu}_2)^t \odot dc) \oplus_{\mathcal{T}} \lightning^{(t,\cdot)}((b\underline{\nu}_2)^t \odot b\underline{\nu}_2\underline{\nu}_2 cc) \end{pmatrix} && (43) \\
&= 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. && (44)
\end{aligned}
$$

In contrast to the eight summands that arose in Eqn. (25), the four summands that appear in Eqn. (44) each meet the matching condition of Eqn. (12). Moreover, these four terms are exactly the ones marked with ✓ in Eqn. (25).

In general, $\lightning^{(t,\cdot)}(A \otimes_{\mathcal{T}} B^k)$ 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. (44) shows the contribution of $\lightning^{(t,\cdot)}(AB)$ (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 $\lightning^{(t,\cdot)}(A \otimes_{\mathcal{T}} B^{*_{\mathcal{T}}})$ 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. (44). The two summands in Eqn. (40)—and hence the arguments on the right-hand sides of the two occurrences of $\otimes_{\mathcal{T}}$ in Eqn. (41)—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. (41) 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. The derivation of Eqn. (42) from Eqn. (41) is where an LCFL-like "substitution" takes place in the "middle" of $b^t \odot (\underline{\nu}_2 \otimes c)$ and $(b \otimes \underline{\nu}_2)^t \odot c$. The net effect is equivalent to the replacement of Y_2 by $d \oplus b\underline{\nu}_2\underline{\nu}_2 c$.

Such "substitutions" are enabled by the key property of how coupled values $a_1^t \odot b_1$ and $a_2^t \odot b_2$ interact with $\otimes_{\mathcal{T}}$ (Eqn. (36)). The sense in which Eqn. (36) acts as an operation that performs a substitution into the middle of an expression is depicted below:

$$\underbrace{(a_1^t \odot b_1)} \otimes_{\mathcal{T}} (a_2^t \odot b_2) = (a_2 \otimes \underbrace{a_1)^t \odot (b_1} \otimes b_2).$$
$$\xrightarrow{\hspace{10em}}$$

The right-hand-side term may appear to be a strange result to obtain from a "substitution" because the items identified by the right-hand-side underbrace are not nested within the parenthesis structure of the right-hand side. However, the "substitution" is completed after $\lightning^{(t,\cdot)}$ is applied:

$$\lightning^{(t,\cdot)}((a_2 \otimes \underbrace{a_1)^t \odot (b_1} \otimes b_2)) = a_2 \otimes \underbrace{a_1 \otimes b_1} \otimes b_2.$$
$$\xrightarrow{\hspace{8em}}$$

Compared with the similar derivation in §3.3, where "pairing" was used rather than "coupling," the key difference comes in the derivation of Eqn. (43) from the preceding line. That is the point at which we apply Eqn. (34) to distribute $\lightning^{(t,\cdot)}(\cdot)$ across $\oplus_{\mathcal{T}}$.

4.5 Correctness of Alg. 4.4

To prove that Alg. 4.4 finds the least solution of an LCFL equation system over an admissible semiring, we make use of a formalism called ***grammar flow analysis*** (GFA), which connects equation systems to tree and string grammars.

Definition 4.6. *[Möncke and Wilhelm 1991; Ramalingam 1996] Let $(D, \oplus)$ be a combine semilattice.*[9] *An* ***abstract grammar*** *G over $(D, \oplus)$ is a collection of*

[9]When the semilattice is oriented according to the conventions of the abstract-interpretation liter-

context-free grammar productions, where each production θ has the form

$$X_0 \to g_\theta(X_1, \ldots, X_k).$$

Parentheses, commas, and g_θ are terminal symbols. G also associates with each production θ a ***production function*** *that gives an interpretation of g_θ: $[\![g_\theta]\!]\colon D^k \to D$. Consequently, every string α of terminal symbols derived in this grammar (i.e., the yield of a complete derivation tree) denotes a composition of functions, and corresponds to a unique value in D, which we call $val_G(\alpha)$ (or simply $val(\alpha)$ when G is understood).*

Let $L_G(X)$ denote the strings of terminals derivable from a nonterminal X. The ***grammar-flow-analysis problem*** *is to compute, for each nonterminal X, the value*

$$m_G(X) = \bigoplus_{\alpha \in L_G(X)} val_G(\alpha).$$

The value $m_G(X)$ is called the ***combine-over-all-derivations*** *value for nonterminal X.*

We can also associate G with a system of mutually recursive equations, where each equation has the form

$$n_G(X_0) = \bigoplus_{X_0 \to g(X_1,\ldots,X_k) \in G} [\![g]\!](n_G(X_1), \ldots, n_G(X_k)).$$

We will use $n_G(X)$ to denote the value of nonterminal X in the ***least fixed-point solution*** *of G's equations.*

A production function $[\![g]\!]$ is ***infinitely distributive*** *in a given argument position if*

$$[\![g]\!](\ldots, \bigoplus_{j \in J} x_j, \ldots) = \bigoplus_{j \in J} [\![g]\!](\ldots, x_j, \ldots)$$

for J a finite or infinite index set.

THEOREM 4.7. *[Möncke and Wilhelm 1991; Ramalingam 1996] If every production function $[\![g]\!] \in G$ is infinitely distributive in each argument position, then for all nonterminals X, $m_G(X) = n_G(X)$.*

Thm. 4.7 generalizes other similar theorems about the coincidence of the valuations obtained from a path-based semantics and an equational semantics when dataflow functions distribute over the dataflow confluence operator. In particular, see Kam and Ullman [Kam and Ullman 1977] and Sharir and Pnueli [Sharir and Pnueli 1981].

An LCFL equation system over semiring $\mathcal{S}$ can always be formulated as a GFA problem over $\mathcal{S}$ in which all productions are of the form $X \to g()$ with $[\![g]\!] = c \in \mathcal{S}$, or $X_0 \to g(X_1)$ with $[\![g]\!] = \lambda Y.a \otimes Y \otimes b$, where $a, b \in \mathcal{S}$. Because there is at most one nonterminal on each right-hand side, we say that such a grammar is *linear*. The principle can be seen from the following example:

ature, a combine-semilattice is a join-semilattice; when it is oriented according to the conventions of the dataflow-analysis literature, it is a meet-semilattice.

EXAMPLE 4.8. *Eqn. (10) can be formulated as the following GFA problem [Knuth 1977]:*

$$W_2 \to g_1() \mid g_2() \mid g_3(W_2) \mid g_4(W_2),$$

where $[\![g_1]\!] \overset{\text{def}}{=} d$, $[\![g_2]\!] \overset{\text{def}}{=} b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c$, $[\![g_3]\!] \overset{\text{def}}{=} \lambda Y.b \otimes Y \otimes \underline{\nu}_2 \otimes c$, *and* $[\![g_4]\!] \overset{\text{def}}{=} \lambda Y.b \otimes \underline{\nu}_2 \otimes Y \otimes c$.

If for each production $X_0 \to g(X_1)$, $[\![g]\!]$ has the form $[\![g]\!] \overset{\text{def}}{=} \lambda Z.Z\, a$, where $a \in D$, we say that the grammar is *left-linear.*

DEFINITION 4.9. *A linear GFA problem G over admissible semiring $\mathcal{S}$ can be transformed into a left-linear GFA problem $G_\mathcal{T}$ over tensor-product semiring $\mathcal{S}_\mathcal{T}$. G and $G_\mathcal{T}$ have the same set of productions, but the production functions of $G_\mathcal{T}$ are defined as shown in column three below:*

	Production Function	
Production	G	$G_\mathcal{T}$
$X_0 \to g_0()$	$[\![g_0]\!] \overset{\text{def}}{=} a$	$[\![g_0]\!]_\mathcal{T} \overset{\text{def}}{=} (\underline{1}^t \odot a)$
$X_0 \to g_1(X_1)$	$[\![g_1]\!] \overset{\text{def}}{=} \lambda Y.a \otimes Y \otimes b$	$[\![g_1]\!]_\mathcal{T} \overset{\text{def}}{=} \lambda Z.Z \otimes_\mathcal{T} (a^t \odot b)$

OBSERVATION 4.10. *By Defn. 2.1(3),*

—$\otimes$ distributes over infinite combines ($\oplus$), and

—$\otimes_\mathcal{T}$ distributes over infinite tensored combines ($\oplus_\mathcal{T}$).

Consequently, the production functions in Defn. 4.9 meet the infinite-distributivity requirement of Thm. 4.7, and hence Thm. 4.7 applies.

OBSERVATION 4.11. *Suppose that $G_\mathcal{T}$ is such a transformed, left-linear GFA problem over the tensor-product semiring associated with admissible semiring $\mathcal{S}$. For all $\alpha \in L_{G_\mathcal{T}}(X)$, $val_{G_\mathcal{T}}(\alpha)$ can always be written in the form $m^t \odot n$ by repeated application of Eqns. (32) and (27). One merely has to create $val_{G_\mathcal{T}}(\alpha)$ by working bottom-up in the derivation tree for α, and at each production instance apply the transformation*

$$\left(a_1^t \odot b_1\right) \otimes_\mathcal{T} \left(a_2^t \odot b_2\right) = \left(a_1^t \otimes a_2^t\right) \odot (b_1 \otimes b_2) = (a_2 \otimes a_1)^t \odot (b_1 \otimes b_2).$$

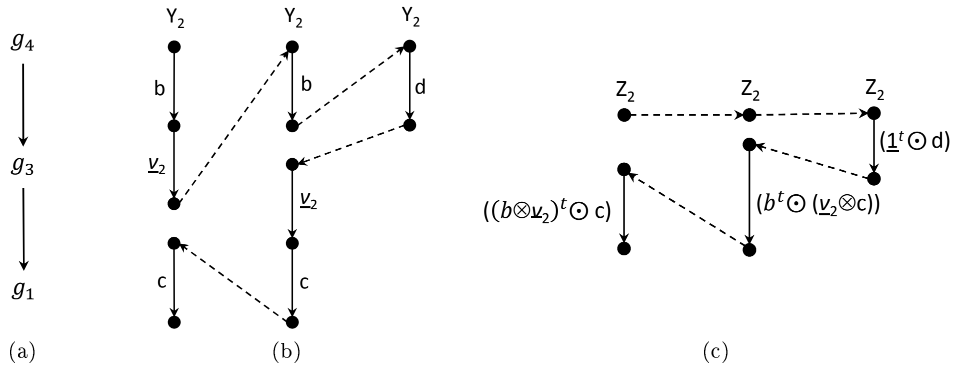
EXAMPLE 4.12. *Defn. 4.9 transforms the GFA problem from Ex. 4.8 into the following tensored, left-linear GFA problem:*

$$W_2 \to g_1() \mid g_2() \mid g_3(W_2) \mid g_4(W_2),$$

where

$$\begin{aligned}
[\![g_1]\!]_\mathcal{T} &\overset{\text{def}}{=} (\underline{1}^t \odot d) \\
[\![g_2]\!]_\mathcal{T} &\overset{\text{def}}{=} (\underline{1}^t \odot (b \otimes \underline{\nu}_2 \otimes \underline{\nu}_2 \otimes c)) \\
[\![g_3]\!]_\mathcal{T} &\overset{\text{def}}{=} \lambda Z.Z \otimes_\mathcal{T} (b^t \odot (\underline{\nu}_2 \otimes c)) \\
[\![g_4]\!]_\mathcal{T} &\overset{\text{def}}{=} \lambda Z.Z \otimes_\mathcal{T} ((b \otimes \underline{\nu}_2)^t \odot c).
\end{aligned}$$

Note that we could also have arrived at this GFA problem by starting with Eqn. (38) and directly creating the corresponding GFA problem over $\mathcal{S}_\mathcal{T}$.



g_4
g_3
g_1
Y_2
Y_2
Y_2
b
$\underline{\nu}_2$
c
b
$\underline{\nu}_2$
c
d
Z_2
Z_2
Z_2
$(\underline{1}^t \odot d)$
$(b^t \odot (\underline{\nu}_2 \otimes c))$
$((b \otimes \underline{\nu}_2)^t \odot c)$
(a)
(b)
(c)

Fig. 4. Graphical representations of (a) the abstract-syntax tree of a string $\alpha = g_4(g_3(g_1()))$ that is in both $L_G(W_2)$ (Ex. 4.8) and $L_{G_\mathcal{T}}(W_2)$ (Ex. 4.12); (b) the unrolled LCFL path that corresponds to $val_G(\alpha)$ in the graphical representation of variable Y_2 from Eqn. (10) (see Fig. 1(b)); and (c) the unrolled regular-language path that corresponds to $val_{G_\mathcal{T}}(\alpha)$ in the graphical representation of variable Z_2 from Eqn. (38) (see Fig. 3).

Because G and $G_\mathcal{T}$ in Defn. 4.9 have exactly the same set of productions, they have the same sets of complete derivation trees, and hence derive the same strings of terminal symbols. In other words, for each nonterminal X,

$$\alpha \in L_G(X) \text{ iff } \alpha \in L_{G_\mathcal{T}}(X). \tag{45}$$

The interpretations that are given to α in the two GFA problems are different; however, the interpretations are related, as shown in the following example:

EXAMPLE 4.13. *Consider the tree $\alpha = g_4(g_3(g_1()))$ depicted in Fig. 4(a), which is in both the languages $L_G(W_2)$ (Ex. 4.8) and $L_{G_\mathcal{T}}(W_2)$ (Ex. 4.12).*

In $L_G(W_2)$, α has the value

$$\begin{aligned} val_G(\alpha) &= [\![g_4]\!]([\![g_3]\!]([\![g_1]\!]())) \\ &= (\lambda Y.b \otimes \underline{\nu}_2 \otimes Y \otimes c)((\lambda Y.b \otimes Y \otimes \underline{\nu}_2 \otimes c)(d)) \\ &= (\lambda Y.b \otimes \underline{\nu}_2 \otimes Y \otimes c)(b \otimes d \otimes \underline{\nu}_2 \otimes c) \\ &= b \otimes \underline{\nu}_2 \otimes b \otimes d \otimes \underline{\nu}_2 \otimes c \otimes c. \end{aligned} \tag{46}$$

Note how the last line of Eqn. (46) is also the value of the path shown in Fig. 4(b), which depicts the unrolled LCFL path that corresponds to $val_G(\alpha)$ in the graphical representation of variable Y_2 from Eqn. (10) (see Fig. 1(b)).

In $L_{G_\mathcal{T}}(W_2)$, α has the value

$$\begin{aligned} val_{G_\mathcal{T}}(\alpha) &= [\![g_4]\!]_\mathcal{T}([\![g_3]\!]_\mathcal{T}([\![g_1]\!]_\mathcal{T}())) \\ &= (\lambda Z.Z \otimes_\mathcal{T} ((b \otimes \underline{\nu}_2)^t \odot c)) \big((\lambda Z.Z \otimes_\mathcal{T} (b^t \odot (\underline{\nu}_2 \otimes c))) \big(\underline{1}^t \odot d\big)\big) \\ &= (\lambda Z.Z \otimes_\mathcal{T} ((b \otimes \underline{\nu}_2)^t \odot c)) \big(\big(\underline{1}^t \odot d\big) \otimes_\mathcal{T} (b^t \odot (\underline{\nu}_2 \otimes c))\big) \\ &= \big(\underline{1}^t \odot d\big) \otimes_\mathcal{T} (b^t \odot (\underline{\nu}_2 \otimes c)) \otimes_\mathcal{T} ((b \otimes \underline{\nu}_2)^t \odot c) \\ &= \big(\underline{1}^t \otimes b^t \otimes (b \otimes \underline{\nu}_2)^t\big) \odot (d \otimes \underline{\nu}_2 \otimes c \otimes c) \\ &= (b \otimes \underline{\nu}_2 \otimes b \otimes \underline{1})^t \odot (d \otimes \underline{\nu}_2 \otimes c \otimes c). \end{aligned} \tag{47}$$

One can clearly see how the fourth line of Eqn. (47) corresponds to the value of the path shown in Fig. 4(c), which depicts the unrolled regular-language path that corresponds to $val_{G_\mathcal{T}}(\alpha)$ in the graphical representation of variable Z_2 from Eqn. (38)

(see Fig. 3).

Eqns. (46) and (47) are related by $\lightning^{(t,\cdot)}$*:*

$$\begin{aligned}\lightning^{(t,\cdot)}(\mathit{val}_{G_\mathcal{T}}(\alpha)) &= \lightning^{(t,\cdot)}\left((b \otimes \underline{\nu}_2 \otimes b \otimes \underline{1})^t \odot (d \otimes \underline{\nu}_2 \otimes c \otimes c)\right)\\ &= b \otimes \underline{\nu}_2 \otimes b \otimes \underline{1} \otimes d \otimes \underline{\nu}_2 \otimes c \otimes c\\ &= \mathit{val}_G(\alpha).\end{aligned}$$

The principle illustrated in Ex. 4.13 is captured by the following property:

LEMMA 4.14. *Let G be a linear GFA problem over semiring $\mathcal{S}$. For every $\alpha \in L_G(X)$ (and hence $\alpha \in L_{G_\mathcal{T}}(X)$),*

$$\lightning^{(t,\cdot)}(\mathit{val}_{G_\mathcal{T}}(\alpha)) = \mathit{val}_G(\alpha). \tag{48}$$

PROOF. By induction on the height h of the derivation tree of α.

Base case, $h = 0$. The derivation tree has the form $g()$. Let $[\![g]\!]$ be a, and thus $\mathit{val}_{G_\mathcal{T}}(g()) = [\![g]\!]_\mathcal{T} = (\underline{1}^t \odot a)$. Then $\lightning^{(t,\cdot)}(\underline{1}^t \odot a) = \underline{1} \otimes a = a = [\![g]\!] = \mathit{val}_G(g())$.

Induction step. Assume that Eqn. (48) holds for all derivation trees of height h. Let $\alpha = g(\beta)$, where β is a string whose derivation tree has height h, so that $\lightning^{(t,\cdot)}(\mathit{val}_{G_\mathcal{T}}(\beta)) = \mathit{val}_G(\beta)$. By Obs. 4.11, $\mathit{val}_{G_\mathcal{T}}(\beta) = m^t \odot n$ and $\mathit{val}_G(\beta) = \lightning^{(t,\cdot)}(m^t \odot n) = m \otimes n$. Further suppose that $[\![g]\!]$ is $\lambda Y.a \otimes Y \otimes b$, and thus $[\![g]\!]_\mathcal{T}$ is $\lambda Z.Z \otimes_\mathcal{T} (a^t \odot b)$. Then

$$\begin{aligned}&\lightning^{(t,\cdot)}(\mathit{val}_{G_\mathcal{T}}(\alpha))\\ &\quad= \lightning^{(t,\cdot)}(\mathit{val}_{G_\mathcal{T}}(g(\beta)))\\ &\quad= \lightning^{(t,\cdot)}([\![g]\!]_\mathcal{T}(\mathit{val}_{G_\mathcal{T}}(\beta)))\\ &\quad= \lightning^{(t,\cdot)}((\lambda Z.Z \otimes_\mathcal{T} (a^t \odot b))(m^t \odot n))\\ &\quad= \lightning^{(t,\cdot)}((m^t \odot n) \otimes_\mathcal{T} (a^t \odot b))\\ &\quad= \lightning^{(t,\cdot)}((m^t \otimes a^t) \odot (n \otimes b))\\ &\quad= \lightning^{(t,\cdot)}((a \otimes m)^t \odot (n \otimes b))\\ &\quad= a \otimes m \otimes n \otimes b\\ &\quad= (\lambda Y.a \otimes Y \otimes b)(m \otimes n)\\ &\quad= [\![g]\!](\mathit{val}_G(\beta))\\ &\quad= \mathit{val}_G(g(\beta))\\ &\quad= \mathit{val}_G(\alpha).\end{aligned}$$

□

COROLLARY 4.15. *For each nonterminal $X \in G$,*

$$\lightning^{(t,\cdot)}(m_{G_\mathcal{T}}(X)) = m_G(X). \tag{49}$$

PROOF.

$$\lightning^{(t,\cdot)}(m_{G_\mathcal{T}}(X)) = \lightning^{(t,\cdot)}\left(\bigoplus_{\alpha \in L_{G_\mathcal{T}}(X)} \mathit{val}_{G_\mathcal{T}}(\alpha)\right) \tag{50}$$

$$= \bigoplus_{\alpha \in L_{G_\mathcal{T}}(X)} \lightning^{(t,\cdot)}(\mathit{val}_{G_\mathcal{T}}(\alpha)) \qquad \text{by Eqn. (35)} \tag{51}$$

$$= \bigoplus_{\alpha \in L_G(X)} \mathit{val}_G(\alpha) \qquad \text{by Lem. 4.14}$$

$$= m_G(X)$$

$\square$

REMARK 4.16. *The step from Eqn. (50) to Eqn. (51) is why the kind of cross-terms that arise with pairing—cf. Eqns. (24) and (25)—do not arise with coupled values created using tensor product. Both approaches can build up values with mirror symmetry* **along** *a single path. However, it is necessary for the readout operation ($\lightning^{(t,\cdot)}$) to distribute across infinite combines (Eqn. (35)) so that we obtain the combine-over-all-LCFL-paths of the readout value of each individual path (see Eqn. (51)).*

We are finally in position to present the proof of the correctness of Alg. 4.4:

THEOREM 4.17. *Given an LCFL equation system $\mathcal{L}$ over admissible semiring $\mathcal{S}$, Alg. 4.4 finds the least solution of $\mathcal{L}$.*

PROOF. We are given an LCFL equation system $\mathcal{L}$ over admissible semiring $\mathcal{S}$. By the method illustrated in Ex. 4.8, $\mathcal{L}$ can be formulated equivalently as a linear GFA problem G over $\mathcal{S}$. We wish to show that Steps 1–4 of Alg. 4.4 compute the least-fixed-point solution $n_G(X)$, for each nonterminal $X \in G$.

From equation system $\mathcal{L}$, Step 1 creates a tensored equation system $\mathcal{L}_\mathcal{T}$. When $\mathcal{L}_\mathcal{T}$ is formulated as a GFA problem, it corresponds to the left-linear GFA problem $G_\mathcal{T}$ over $\mathcal{S}_\mathcal{T}$ obtained by applying Defn. 4.9 to G.

Steps 2 and 3 use Tarjan's path-expression algorithm [Tarjan 1981b] to create an appropriate regular expression for each variable Z_i in $\mathcal{L}_\mathcal{T}$, which are then evaluated. Consequently, by [Tarjan 1981b, Thm. 5], these steps find the combine-over-all-derivations solution to $\mathcal{L}_\mathcal{T}$, and hence compute $m_{G_\mathcal{T}}(X)$ for each nonterminal $X \in G_\mathcal{T}$. Finally, Step 4 computes $\lightning^{(t,\cdot)}(m_{G_\mathcal{T}}(X))$ for $X \in G_\mathcal{T}$. However, by the results presented above, we have

$$\begin{array}{rcll} \lightning^{(t,\cdot)}(m_{G_\mathcal{T}}(X)) & = & m_G(X) & \text{by Cor. 4.15} \\ & = & n_G(X) & \text{by Obs. 4.10 and Thm. 4.7} \end{array}$$

$\square$

4.6 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 $\vec{\nu}$:

$$\boxed{\begin{array}{rcl} \vec{\nu}^{(0)} & = & \vec{f}(\vec{\underline{0}}) \\ \vec{\nu}^{(i+1)} & = & \langle \lightning^{(t,\cdot)}(Z_1^{(i)}), \ldots, \lightning^{(t,\cdot)}(Z_n^{(i)}) \rangle \end{array}} \qquad (52)$$

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

$$\boxed{\tau_{\text{Reg}}(\vec{Y} = \vec{f}(\vec{\nu}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y}))} \qquad (53)$$

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

It may be helpful to see the overall transformation expressed using a single inductive definition, rather than as the composition of the operators τ_{Reg} and $\mathcal{D}$.

DEFINITION 4.18. *The* ***tensored differential*** *of a component function $f_i(\vec{x})$ with respect to X_j at $\underline{\vec{\nu}}$, denoted by $\mathcal{D}^{\mathcal{T}}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Z})$, is defined as follows:*

$$\boxed{\mathcal{D}^{\mathcal{T}}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Z}) = \begin{cases} \underline{0}_{\mathcal{T}} & \text{if } f_i = s \in \mathcal{S} \\ \underline{0}_{\mathcal{T}} & \text{if } f_i = X_k \text{ and } k \neq j \\ Z_j & \text{if } f_i = X_j \\ \bigoplus_{k \in K}{}_{\mathcal{T}}\, \mathcal{D}^{\mathcal{T}}_{X_j} g_k|_{\underline{\vec{\nu}}}(\vec{Z}) & \text{if } f_i = \bigoplus_{k \in K} g_k \\ \begin{pmatrix} \mathcal{D}^{\mathcal{T}}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Z}) \otimes_{\mathcal{T}} (\underline{1}^t \odot h(\underline{\vec{\nu}})) \\ \oplus_{\mathcal{T}} \mathcal{D}^{\mathcal{T}}_{X_j} h|_{\underline{\vec{\nu}}}(\vec{Z}) \otimes_{\mathcal{T}} (g(\underline{\vec{\nu}})^t \odot \underline{1}) \end{pmatrix} & \text{if } f_i = g \otimes h \end{cases}}$$

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

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

An easy inductive argument shows that

$$\boxed{\mathcal{D}^{\mathcal{T}}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Z}) = \tau_{\text{Reg}}(\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y}))}$$

NPA-TP can now be restated as the following iterative algorithm: one solves the sequence of problems defined by Eqn. (52) for $\vec{\nu}$, where $\vec{Z}^{(i)} = \langle Z_1^{(i)}, \ldots, Z_n^{(i)}\rangle$ is the least solution of the following equation system over $\mathcal{S}_{\mathcal{T}}$:

$$\boxed{\vec{Z} = (\overrightarrow{\underline{1}^t} \odot \vec{f}(\vec{\nu}^{(i)})) \oplus_{\mathcal{T}} \mathcal{D}^{\mathcal{T}} \vec{f}|_{\vec{\nu}^{(i)}}(\vec{Z})} \tag{54}$$

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. (54) on each round. (See the discussion of Alg. 7.1 in §7.)

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:

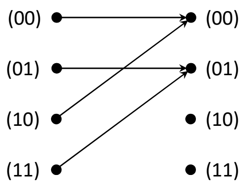
Semiring. A predicate-abstraction domain over predicate set P is a relational weight domain $(\mathcal{P}(A \times A), \cup, ;, \emptyset, id)$ (Defn. 2.2), where A is the set of *Boolean assignments* over P; that is, $A = P \to \text{Bool}$. ($P \to \text{Bool}$ is isomorphic to $\mathcal{P}(P)$.) As discussed just after Defn. 2.2, the semiring value associated with an assignment statement or assume statement `st` is the binary relation $R_{\texttt{st}} \subseteq A \times A$ that represents the effect of `st` on assignments over P.

Let N denote $|A|$. Each semiring element R can be thought of as an $N \times N$ Boolean matrix

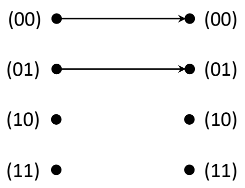
$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,N} \\ \vdots & \ddots & \vdots \\ r_{N,1} & \cdots & r_{N,N} \end{bmatrix}.$$

We will denote such a matrix by $R(A, A')$ when we wish to introduce names for the matrix's index sets (or, equivalently, for the vocabularies of transition relation R).

For instance, suppose that predicate set P is $\{p_1, p_2\}$. Letting $(b_1 b_2)$ denote the assignment in which p_1 has value b_1 and p_2 has value b_2, the set A is $\{(00), (01), (10), (11)\}$. A transition relation $R(A, A') \subseteq A \times A$ can be depicted as a graph. For instance, in a Boolean program, the transition relations for the assignment statement "p_1 := false" and the assume statement "assume p_1 = false" can be depicted as follows:



p_1 := false



assume p_1 = false

A set of assignments can be represented by a characteristic (row-)vector—e.g., [1010] represents $\{(00), (10)\}$—in which case a transition relation $R \subseteq A \times A$ corresponds to a Boolean matrix that transforms a pre-state row-vector to a post-state row-vector. For instance, "p_1 := false" and "assume p_1 = false" have the respective matrices

$$\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 1&0&0&0 \\ 0&1&0&0 \end{bmatrix} \qquad \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix}$$

Each non-zero entry (i, j) in a matrix represents an edge from $i \in A$ to $j' \in A'$ in the graphical depiction.

Transpose. The transpose operation on relational weights is matrix transpose. For instance, the transpose of the matrix for "p_1 := false" is

$$\begin{bmatrix} 1&0&1&0 \\ 0&1&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix}$$

Semantically, the transpose $R^t(A', A)$ is the inverse of relation $R(A, A')$:

$$\begin{aligned} R^t(A', A) &= \{(a', a) \mid R^{-1}(a', a)\} \\ &= \{(a', a) \mid R(a, a')\}. \end{aligned}$$

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, the tensor-product of $R(A, A')$ and $S(B, B')$ is the 4-ary relation

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

The interleaving of the vocabularies in the order (A, B, A', B') mimics the order one obtains from the Kronecker product of the matrix representations.

Coupling. The coupling of $R(A, A')$ and $S(B, B')$ is the tensor-transpose relation

$$\begin{aligned} (R^t \odot S)(A', B, A, B') &= R^t(A', A) \odot S(B, B') \\ &= \{(a', b, a, b') \mid R(a, a') \wedge S(b, b')\}. \end{aligned}$$

In matrix form, we have

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

and thus

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

For instance, $(p_1 := \text{false})^t \odot (\text{assume } p_1 = \text{false})$ is the matrix

$$\left[\begin{array}{cccc|cccc|cccc|cccc}
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{array}\right]$$

Detensor Transpose. If $T(A', B, A, B')$ is a tensor-transpose relation, detensor-transpose is defined as follows:

$$\lightning^{(t,\cdot)}(T) \overset{\text{def}}{=} \{(a,b') \mid T(a',b,a,b') \land a' = b\}. \tag{55}$$

If one thinks of $T(A',B,A,B')$ as defined by a formula with four sets of free variables A', B, A, and B', $\lightning^{(t,\cdot)}(T(A',B,A,B'))$ can also be expressed as follows: $\exists A',B\colon (T(A',B,A,B') \land A' = B)$.

THEOREM 5.1. *The transpose, tensor-product, and detensor-transpose operations defined above satisfy Eqns. (26)–(35).*

PROOF. *Part I, Eqns. (26)–(32)*: Eqns. (26)–(32) are standard properties of matrix transpose and Kronecker product.

Part II, Eqn. (33):

$$\begin{aligned} \lightning^{(t,\cdot)}(R^t \odot S) &= \lightning^{(t,\cdot)}(\{(a',b,a,b') \mid R(a,a') \land S(b,b')\}) \\ &= \{(a,b) \mid R(a,a') \land S(b,b') \land a' = b\} \\ &= R \otimes S \end{aligned}$$

Part II, Eqn. (34): Because $\oplus_{\mathcal{T}}$ for relations corresponds to logical-or ($\lor$), the conjunct "$\ldots \land A' = B$" distributes over $\oplus_{\mathcal{T}}$. Therefore,

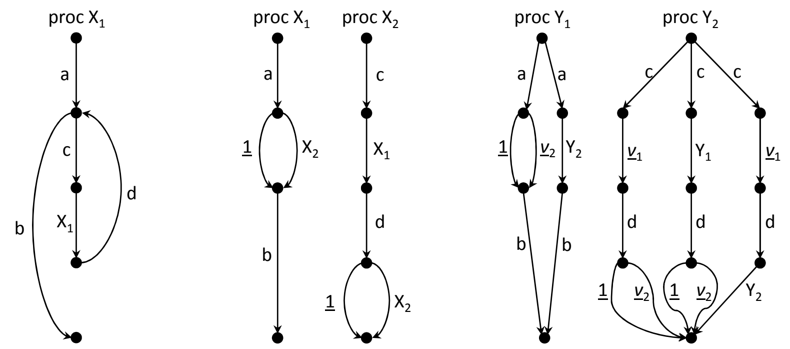
$$\begin{aligned} &\lightning^{(t,\cdot)}(T_1(A',B,A,B') \oplus_{\mathcal{T}} T_2(A',B,A,B')) \\ &\quad= \exists A',B\colon ((T_1(A',B,A,B') \oplus_{\mathcal{T}} T_2(A',B,A,B')) \land A' = B) \\ &\quad= \exists A',B\colon \begin{pmatrix} & (T_1(A',B,A,B') \land A' = B) \\ \oplus_{\mathcal{T}} & (T_2(A',B,A,B') \land A' = B) \end{pmatrix} \\ &\quad= \begin{pmatrix} & (\exists A',B\colon T_1(A',B,A,B') \land A' = B) \\ \oplus_{\mathcal{T}} & (\exists A',B\colon T_2(A',B,A,B') \land A' = B) \end{pmatrix} \\ &\quad= \lightning^{(t,\cdot)}(T_1(A',B,A,B')) \oplus_{\mathcal{T}} \lightning^{(t,\cdot)}(T_2(A',B,A,B')) \end{aligned}$$

Part II, Eqn. (35): The same argument that was used above for Eqn. (34) applies when we replace the binary operator for combining tensored values ($\oplus_{\mathcal{T}}$) with the operator for infinite combines of tensored values $\left(\bigoplus_{i \in I}{}_{\mathcal{T}}\right)$. $\square$

REMARK 5.2. *The semiring values that arise in interprocedural-analysis problems are generally abstractions of concrete functions that transform a set of pre-states to a set of post-states. For that reason, the semantic definitions given above apply more broadly, because they can be interpreted in abstract domains other than predicate-abstraction domains. However, we know of abstract domains, such as the so-called KS domain of modular-arithmetic affine relations [Elder et al. 2014], that support all of the operations used in the semantic definitions, yet do not satisfy all of the properties Eqns. (27)–(35).*

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.



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

(a)

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

(b)

$$Y_1 = a(\underline{1} \oplus \underline{\nu}_2)b \oplus aY_2b$$
$$Y_2 = \begin{pmatrix} c\underline{\nu}_1 d(\underline{1} \oplus \underline{\nu}_2) \\ \oplus\ cY_1d(\underline{1} \oplus \underline{\nu}_2) \\ \oplus\ c\underline{\nu}_1 dY_2 \end{pmatrix}$$

(c)

Fig. 5. Three 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. (c) Linearized equation system over $\vec{Y}$ obtained from (b) via Eqn. (5).

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. 5(a).*

```
$X_1$() {
    $s_a$;
    while ($\star$) {
        $s_c$;
        $X_1$();
        $s_d$
    }
    $s_b$
}
        (i)
```

```
$X_1$() {                 $X_2$() {
    $s_a$;                    $s_c$;
    if ($\star$) $X_2$()      $X_1$();
    $s_b$                     $s_d$;
}                             if ($\star$) $X_2$()
                          }
                (ii)
```

Program (ii) shows one possible transformation of program (i) to put it in loop-free form. Fig. 5(b) shows program (ii) in graphical form, and also as an equation system. Fig. 5(c) shows the linearized equation system for $\vec{Y}$ that is obtained from the equation system in Fig. 5(b) via Eqn. (5).

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 that (i) may be more convenient, and (ii) is somewhat different from an operational standpoint. Our approach involves introducing a Kleene-star operator, and allowing the right-hand side of each equation to be a *regular expression*:

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:***

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

Fig. 5(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.[10] 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*

$$\mathcal{D}_{X_j} f|_{\underline{\vec{\nu}}}(\vec{Y}) = (g(\underline{\vec{\nu}}))^* \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^* \tag{56}$$

PROOF. First, consider the left-hand side of Eqn. (56).

$$\begin{aligned} \mathcal{D}_{X_j} f|_{\underline{\vec{\nu}}}(\vec{Y}) &= \mathcal{D}_{X_j}\left(\bigoplus_{i\in\mathbb{N}} g^i\right)|_{\underline{\vec{\nu}}}(\vec{Y}) \\ &= \bigoplus_{i\in\mathbb{N}} \mathcal{D}_{X_j} g^i|_{\underline{\vec{\nu}}}(\vec{Y}) \\ &= \bigoplus_{i\in\mathbb{N}} \bigoplus_{\substack{j+k=i-1\\ j,k\in\mathbb{N}}} (g(\underline{\vec{\nu}}))^j \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^k \end{aligned} \tag{57}$$

Because semiring multiplication ($\otimes$) is non-commutative, the property used to derive line (57) from the previous line is similar to the familiar rule from calculus

$$\left(\frac{d}{dx} g(x)^i\right)\bigg|_{\nu} = i\, g(\nu)^{i-1} \left(\frac{d}{dx} g(x)\right)\bigg|_{\nu},$$

[10]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. (53) on each round of NPA-TP. See steps 1 and 4 of Alg. 7.1.

re-expressed using i summands, as follows:

$$\sum_{\substack{j+k=i-1\\ j,k\in\mathbb{N}}} g(\nu)^j \left(\frac{d}{dx}g(x)\right)\bigg|_{\nu} g(\nu)^k.$$

Now consider the right-hand side of Eqn. (56).

$$\begin{aligned}(g(\underline{\vec{\nu}}))^* \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^* &= \bigoplus_{j\in\mathbb{N}}(g(\underline{\vec{\nu}}))^j \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes \bigoplus_{k\in\mathbb{N}}(g(\underline{\vec{\nu}}))^k \\ &= \bigoplus_{j,k\in\mathbb{N}} (g(\underline{\vec{\nu}}))^j \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^k \qquad (58)\end{aligned}$$

We wish to show that (57) = (58).

(57) $\supseteq$ (58). For each summand s of (58), we have $0 \le j$ and $0 \le k$. Let $i = j + k + 1$, and thus $i \ge 1$, which means that s also occurs in (57).

(57) $\subseteq$ (58). When $i = 0$, $\{(j,k) \mid j,k \in \mathbb{N} \wedge j+k = -1\} = \emptyset$, and hence, because the index is vacuous, the value in (57) of the sum

$$\bigoplus_{\substack{j+k=i-1\\ j,k\in\mathbb{N}}} (g(\underline{\vec{\nu}}))^j \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^k \qquad (59)$$

is $\underline{0}$. For each $i \ge 1$, Eqn. (59) is some non-vacuous sum s; moreover, $j \in \mathbb{N}$ and $k \in \mathbb{N}$, and hence s also occurs in (58).

□

For NPA, 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. 2.5, extended with one more case for Kleene-star:

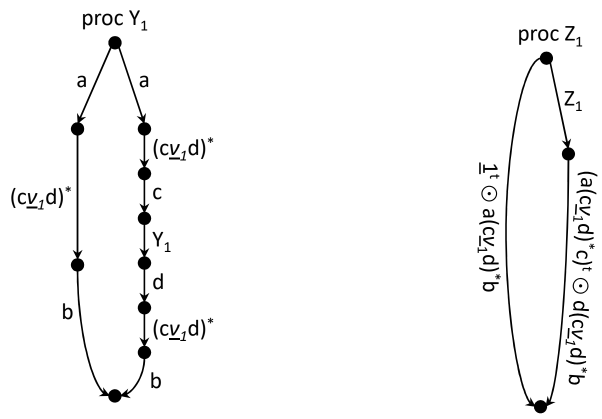
$$\boxed{\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y}) = (g(\underline{\vec{\nu}}))^* \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Y}) \otimes (g(\underline{\vec{\nu}}))^* \quad \text{if } f_i = g^*} \qquad (60)$$

This rule, like the others given in Defn. 2.5, has a single recursive call on $\mathcal{D}_{X_j}$ in each summand. (There is only a single summand in Eqn. (60).) Consequently, when Defn. 2.5 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, Thm. 6.3 implies that we can again create a left-linear equation system over $\mathcal{S}_{\mathcal{T}}$ by applying τ_{Reg} to the LCFL equation system. Alternatively, the tensored differential of $f_i(\vec{X})$ can be obtained by the rule given in Defn. 4.18, extended with one more case for Kleene-star:

$$\boxed{\mathcal{D}^{\mathcal{T}}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Z}) = \mathcal{D}^{\mathcal{T}}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Z}) \otimes_{\mathcal{T}} (((g(\underline{\vec{\nu}}))^*)^t \odot (g(\underline{\vec{\nu}}))^*) \quad \text{if } f_i = g^*}$$

Figs. 6(a) and (b) show, respectively, the LCFL and left-linear equations for Y_1 and Z_1 obtained from Fig. 5(a) by these transformations.

There is an operational difference between the equation system used by Esparza et al. (e.g., Fig. 5(c)) and the one described above (Fig. 6(a)). In particular, each round of NPA creates successively better approximations of a set of procedure


proc Y_1
a
a
$(c\underline{\nu}_1 d)^*$
$(c\underline{\nu}_1 d)^*$
c
Y_1
d
$(c\underline{\nu}_1 d)^*$
b
b
proc Z_1
Z_1
$\underline{1}^t \odot a(c\underline{\nu}_1 d)^* b$
$(a(c\underline{\nu}_1 d)^* c)^t \odot d(c\underline{\nu}_1 d)^* b$

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

(a)

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

(b)

Fig. 6. The NPA (a) and NPA-TP (b) equation systems that result from the equation "$X_1 = a(cX_1d)^*b$" (from Fig. 5(a)), when both NPA and NPA-TP are extended to handle Kleene-star.

summaries, one for each procedure in the program. Operationally, Y_2 is improved on each round of the Esparza et al. algorithm by pre-multiplying Y_2 by $c\underline{\nu}_1 d$ (and adding in some other terms, including one obtained by pre-multiplying Y_1 by c and post-multiplying by $d(\underline{1} \oplus \underline{\nu}_2)$). In contrast, on each round of the method described above, Y_2 is improved by pre-multiplying Y_2 by $a(c\underline{\nu}_1 d)^*$ and post-multiplying by $(c\underline{\nu}_1 d)^* b$.

7. ALGORITHM PRAGMATICS

NPA-TP can be implemented in a straightforward manner using Eqns. (52) and (53). However, as mentioned in §4.6, 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. (52) and (53) as described 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, $\mathcal{E} = \{X_j = Rhs_j(\vec{X}) \mid X_j \in \vec{X}\}$.

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

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

(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 $\langle 0, j\rangle$.
 (The symbols in $\Sigma \stackrel{\text{def}}{=} \{\langle k, j\rangle \mid 0 \le k \le n, 1 \le j \le n\}$ (i.e., $[0..n] \times [1..n]$) serve as proxies for the coefficients in the equation system created in step (2), and will be replaced by appropriate generalized regular expressions in step (5).)

(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 Σ.*

(5) *Create a map m, in which variable Z_i, $1 \le i \le n$, is mapped to the regular expression $\mathcal{R}_i$, but with each symbol in Σ replaced by a generalized regular expression, as follows:*

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

(6) $i \leftarrow 0; \vec{\mu} \leftarrow \vec{f}(\underline{\vec{0}})$

(7) *Repeat*
 (a) $\vec{\nu}^{(i)} = \vec{\mu}$
 (b) $\vec{\mu} = \langle \lightning^{(t,\cdot)}([\![m(Z_j)]\!]_{\mathcal{T}} \vec{\nu}^{(i)}) \mid Z_j \in \vec{Z}\rangle$
 (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. (53): (i) The values $(\underline{1}^t \odot \mathrm{Rhs}_j(\vec{\nu}))$ 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. (53). (ii) The remaining edges of $\mathcal{G}$ encode the regular structure of the recursive portion of Eqn. (53), $\tau_{\text{Reg}}(\vec{Y} = \ldots \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y}))$.

—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 the equation system and corresponding graphical depiction in Fig. 6(b). When Tarjan's algorithm is applied to it in step (4) of Alg. 7.1, the regular expression created for Z_1 is $\langle 0,1 \rangle \oplus \langle 0,1 \rangle (\langle 1,1 \rangle)^*$. After step (5), $m(Z_1)$ is the generalized regular expression*

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

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

—*Evaluate $m(Z_1)$ with respect to the current value of $\langle \underline{\nu}_1 \rangle$ to obtain, say, $w_1 \in \mathcal{S}_{\mathcal{T}}$.*

—*Set $\langle \underline{\nu}_1 \rangle$ to $\langle \lightning^{(t,\cdot)}(w_1) \rangle$.*

Similarly, for Eqn. (38) and the corresponding graphical depiction in Fig. 3, 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

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

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

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

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 $\langle \lightning^{(t,\cdot)}(w_1), \lightning^{(t,\cdot)}(w_2) \rangle$.*

Correctness. Let $\vec{X}^\star$ denote the least fixed-point of equation system $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$. $\vec{X}^\star$ 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 $\vec{X}^\star$. The following theorem shows that each Newton iterate $\vec{\nu}^{(i)}$ is trapped between the corresponding Kleene iterate $\vec{\kappa}^{(i)}$ and the least solution $\vec{X}^{\star}$.

THEOREM 7.3. *Let $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ be an equation system whose least fixed-point is $\vec{X}^{\star}$. Then for all i, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)} \sqsubseteq \vec{X}^{\star}$.*

PROOF. Esparza et al. proved the theorem for NPA applied to an equation system over a general semiring [Esparza et al. 2010, Thm. 3.9], but without occurrences of Kleene-star. For the situation described in §6.1, where loops are handled by transforming each procedure to eliminate all loops, Thm. 4.17 allows the theorem of Esparza et al. to be carried over directly. Note that in Alg. 7.1, steps (6) and (7) create the Newton iterates using the set of regular expressions $m(Z_i)$ created in step (5). These regular expressions can include occurrences of Kleene-star, but those uses of Kleene-star are covered by Thm. 4.17: the regular expressions incorporate both (a) the structure of the LCFL problems that need to be solved on each Newton round (see steps (3) and (4)), and (b) τ_{Reg}-transformed versions of the LCFL right-hand-side terms (see step (2)).

For the situation described in §6.2, the component functions of the equation system can now contain regular operators. The proof of the theorem for this situation is covered by the proof of a slightly more general version of the theorem, Thm. 8.5, given in §8.1. □

The significance of Thm. 7.3 is that because successive Kleene iterates approach $\vec{X}^{\star}$, successive Newton iterates must also approach $\vec{X}^{\star}$.

COROLLARY 7.4. *Given an interprocedural dataflow-analysis problem over an admissible semiring that has no infinite ascending chains, Alg. 7.1 terminates with the least solution.*

8. LOCAL VARIABLES

This section discusses how to extend NPA and NPA-TP to create procedure summaries (see Ex. 2.4) for programs with local variables. We adopt the approach introduced by Knoop and Steffen [1992]. 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 [Müller-Olm and Seidl 2004; Lal et al. 2005].)

DEFINITION 8.1. *[Lal et al. 2005] 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;[11] *i.e., for all $a, a_i, b, b_i \in D$ and $I \subseteq \mathbb{N}$,*

$$\begin{aligned} M\left(\bigoplus_{i\in I} a_i, b\right) &= \bigoplus_{i\in I} M(a_i, b) \\ M\left(a, \bigoplus_{i\in I} b_i\right) &= \bigoplus_{i\in I} M(a, b_i) \end{aligned}$$

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

The introduction of merge functions changes interprocedural dataflow analysis to a problem of "combine-over-all-nested-word-paths" [Knoop and Steffen 1992; Müller-Olm and Seidl 2004; Lal et al. 2005].

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

$$\text{For all } a, c \in D, M(a, c) = a \otimes M(\underline{1}, c). \qquad (61)$$

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. (61) 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 $a_i \otimes m$.

EXAMPLE 8.2. *Consider the following program scheme:*

```
X_1() {                 X_2() }                 X_3() {
    if (⋆) {                if (⋆) {                if (⋆) {
        s_a                     s_c                     s_g;
    } else {                } else {                } else {
        s_b; X_2(); X_1()       s_d; X_3(); X_2()       s_e; X_2(); s_h
    }                       }                       }
}                       }                       }
```

which, with merge functions, corresponds to the following equation system, depicted in Fig. 7:

$$\begin{aligned} X_1 &= a \oplus M(M(b, X_2), X_1) \\ X_2 &= c \oplus M(M(d, X_3), X_2) \\ X_3 &= g \oplus (M(e, X_2) \otimes h). \end{aligned} \qquad (62)$$

By the path-extension property (Defn. 8.1(3)), Eqn. (62) can be re-written as follows, which is depicted in Fig. 8:

$$\begin{aligned} X_1 &= a \oplus (b \otimes M(\underline{1}, X_2) \otimes M(\underline{1}, X_1)) \\ X_2 &= c \oplus (d \otimes M(\underline{1}, X_3) \otimes M(\underline{1}, X_2)) \\ X_3 &= g \oplus (e \otimes M(\underline{1}, X_2) \otimes h). \end{aligned} \qquad (63)$$

[11]The binary case of Defn. 8.1 simplifies to: for all $a, a_1, a_2, b, b_1, b_2 \in D$,

$$\begin{aligned} M(a_1 \oplus a_2, b) &= M(a_1, b) \oplus M(a_2, b) \\ M(a, b_1 \oplus b_2) &= M(a, b_1) \oplus M(a, b_2) \end{aligned}$$

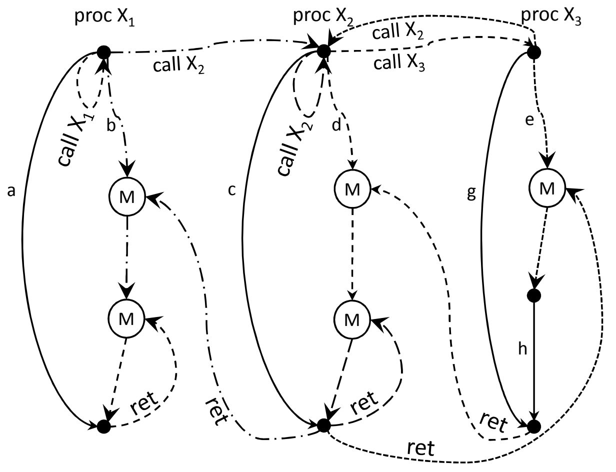


Fig. 7. Graphical depiction of Eqn. (62). Each collection of dotted/dashed lines with the same rendering represents a call, its matching return, and the application of merge function M.

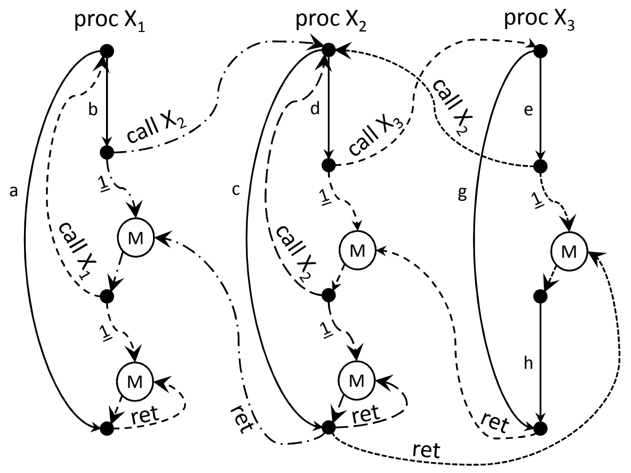


Fig. 8. Graphical depiction of Eqn. (63).

We extend our language of regular expressions with the unary operator Project(·), whose semantics is $[\![\text{Project}(e)]\!]\vec{\nu} = M(\underline{1}, [\![e]\!]\vec{\nu})$. Eqn. (63) can be rewritten using Project as follows:

$$\begin{aligned} X_1 &= a \oplus (b \otimes \text{Project}(X_2) \otimes \text{Project}(X_1)) \\ X_2 &= c \oplus (d \otimes \text{Project}(X_3) \otimes \text{Project}(X_2)) \\ X_3 &= g \oplus (e \otimes \text{Project}(X_2) \otimes h). \end{aligned} \tag{64}$$

However, in depictions of call-sites in figures, we will continue to use M instead of Project because at a call-site it is easier to give a graphical depiction of $a \otimes M(\underline{1}, c)$ than $a \otimes \text{Project}(c)$—see Figs. 8, 9, and 10.

8.1 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 $((G \times L) \times (G \times L) \to \mathbb{B}, \cup, ;, \emptyset, \mathrm{Id})$. A typical element will be denoted by $R(G, L, G', L')$.

In this case, the following is an acceptable merge function:

$$\begin{aligned} M(R_1, R_2) &= R_1 \otimes M(\underline{1}, R_2) \\ &= R_1 \otimes \mathrm{Project}(R_2) \\ \mathrm{Project}(R) &= \{(g, m, g', m) \mid R(g, l, g', l')\}. \end{aligned} \tag{65}$$

If one thinks of $R(G, L, G', L')$ as defined by a formula with four sets of free variables G, L, G', and L', $\mathrm{Project}(R(G, L, G', L'))$ can also be expressed as follows: $(\exists L, L' \colon R(G, L, G', L')) \wedge (L = L')$.

LEMMA 8.3. *The M operation defined in Eqn. (65) is an acceptable merge function for $\mathcal{S}$, in the sense of Defn. 8.1.*

PROOF. See App. B. □

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:

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

The intuition behind this definition is as follows:

—NPA is a kind of sampling method for the state space of the program: if procedure P has multiple call-sites, then the linearized program used during a given round of Newton's method allows the analyzer to sample the state space of P by taking the $\oplus$ of various paths through P. Along each such path through P, the abstract values for the call-sites encountered are held fixed, except for possibly one call-site on the path, which is explored by visiting (the linearized version of) the called procedure. The abstract value for P and all the other procedures are updated according to the results of this state-space exploration, and the algorithm proceeds to the next Newton round.

—Each call-site on each of the paths in P is still just a call-site, regardless of whether its value is held fixed, or whether the (linearized) called procedure is explored. For each such call-site, the merge function still needs to be invoked.

EXAMPLE 8.4. *Fig. 9 depicts the equation system that results from applying the NPA linearizing transformation to Eqn. (64), which creates the following equation system:*

$$\begin{aligned} Y_1 &= \begin{pmatrix} a \oplus (b \otimes \mathit{Project}(\underline{\nu_2}) \otimes \mathit{Project}(\underline{\nu_1})) \\ \oplus (b \otimes \mathit{Project}(\underline{\nu_2}) \otimes \mathit{Project}(Y_1)) \\ \oplus (b \otimes \mathit{Project}(Y_2) \otimes \mathit{Project}(\underline{\nu_1})) \end{pmatrix} \\ Y_2 &= \begin{pmatrix} c \oplus (d \otimes \mathit{Project}(\underline{\nu_3}) \otimes \mathit{Project}(\underline{\nu_2})) \\ \oplus (d \otimes \mathit{Project}(\underline{\nu_3}) \otimes \mathit{Project}(Y_2)) \\ \oplus (d \otimes \mathit{Project}(Y_3) \otimes \mathit{Project}(\underline{\nu_2})) \end{pmatrix} \\ Y_3 &= \begin{pmatrix} g \oplus (e \otimes \mathit{Project}(\underline{\nu_2}) \otimes h) \\ \oplus (e \otimes \mathit{Project}(Y_2) \otimes h). \end{pmatrix} \end{aligned} \tag{66}$$

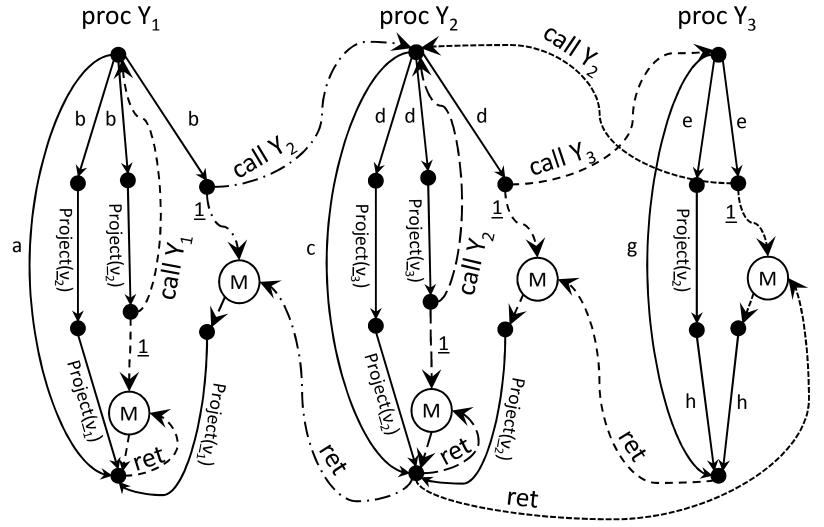


Fig. 9. Graphical depiction of the equation system that results from applying the NPA linearizing transformation to Eqn. (64).

Correctness. With the extension given here for local variables and in §6.2 for loops, the component functions of an equation system $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ can now contain both regular operators and occurrences of the operator Project. Let $\vec{X}^{\star}$ denote the least fixed-point of $\mathcal{E}$. The following theorem extends Thm. 7.3 to this situation:

THEOREM 8.5. *Let $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ be an equation system whose right-hand-side terms can contain both regular operators and occurrences of the operator Project, and whose least fixed-point is $\vec{X}^{\star}$. Then for all i, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)} \sqsubseteq \vec{X}^{\star}$.*

PROOF. See App. A.1. □

As mentioned earlier, Esparza et al. proved a similar theorem for NPA applied to an equation system over a general semiring [Esparza et al. 2010, Thm. 3.9], but without occurrences of Kleene-star and Project.

Thm. 8.5 concerns the variant of NPA defined by Eqns. (4) and (5), where $\mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}$ is extended so that Eqn. (7) of Defn. 2.5 incorporates both (i) the rule for Kleene-star from §6.2, and (ii) the rule for Project from §8.1. As mentioned in footnote 6, Esparza et al. also show that if Eqn. (5) is changed to Eqn. (6) (repeated here for the reader's convenience),

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

the combinations Eqns. (4) and (5) and Eqns. (4) and (6) produce the same set of iterates $\vec{\nu}^{(0)}, \vec{\nu}^{(1)}, \ldots, \vec{\nu}^{(i)}, \ldots$ [Esparza et al. 2010, Prop. 7.1]. The following theorem shows that this property carries over to our extensions of NPA, as well:

THEOREM 8.6. *Let $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ be an equation system whose right-hand-side terms can contain both regular operators and occurrences of the operator Project.*

Let the sequence $N \overset{\text{def}}{=} [\underline{\vec{\nu}}^{(i)} \mid i \in \mathbb{N}]$ be defined by Eqns. (4) and (5), and the sequence $M \overset{\text{def}}{=} [\underline{\vec{\mu}}^{(i)} \mid i \in \mathbb{N}]$ be defined by Eqns. (4) and (6). Then $M = N$.

Proof. See App. A.2. $\square$

8.2 Merge Functions and NPA-TP[PA]

To use NPA-TP[PA] to solve an equation system over a predicate-abstraction domain that involves local variables, we will focus on a special kind of equation system, whose solution provides a projection value for each variable in the original equation system. For instance, Eqn. (63) is rewritten as follows:

$$\begin{array}{lclcl} W_1 &=& \text{Project}(X_1) &=& \text{Project}(a \oplus (b \otimes W_2 \otimes W_1)) \\ W_2 &=& \text{Project}(X_2) &=& \text{Project}(c \oplus (d \otimes W_3 \otimes W_2)) \\ W_3 &=& \text{Project}(X_3) &=& \text{Project}(g \oplus (e \otimes W_2 \otimes h)). \end{array} \tag{67}$$

In a program-analysis problem, the value of W_i serves as a summary of procedure X_i suitable for use in a caller of X_i. Once the $\overrightarrow{W}$ values are in hand, one can obtain the $\vec{X}$ values by evaluating the right-hand sides of the original equation system. We introduce a name for an equation system of this form:

Definition 8.7. *An equation system over predicate-abstraction domain PA is a* **projection-equation system** *if it has the form*[12]

$$\mathcal{E} : \overrightarrow{W} = \mathit{Project}(\vec{e}(\overrightarrow{W})),$$

where each component function is defined by an expression of the form Project(e_k), and e_k is an expression over variables in $\overrightarrow{W}$, constants from PA, and the operators $\oplus$, $\otimes$, and *.

We use $\overrightarrow{W}^\star$ to denote the least solution of $\mathcal{E}$.

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. (67) the locations of Project$(\cdot)$ are always associated with the bodies of procedures. When the NPA linearizing transformation is applied to projection-equation system

$$\mathcal{E} : \overrightarrow{W} = \text{Project}(\vec{e}(\overrightarrow{W})),$$

for the ith Newton round, we obtain the equation system

$$\vec{Y} = \text{Project}\left(\vec{e}(\vec{\nu}^{(i)})\right) \oplus \text{Project}\left(\mathcal{D}\vec{e}|_{\vec{\nu}^{(i)}}(\vec{Y})\right),$$

and thus each equation has the form

$$Y_k \;=\; \text{Project}\left(a_k^{(i)}\right) \oplus \text{Project}\left(\bigoplus_j \bigoplus_l \left(b_{j,k,l}^{(i)} \otimes Y_j \otimes c_{j,k,l}^{(i)}\right)\right)$$

[12]When Project (or Project$_{\mathcal{T}}$) is applied to a vector, our convention is that the operator is applied componentwise.

$$= \text{Project}\left(a_k^{(i)} \oplus \bigoplus_j \bigoplus_l \left(b_{j,k,l}^{(i)} \otimes Y_j \otimes c_{j,k,l}^{(i)}\right)\right), \tag{68}$$

where the superscript $^{(i)}$ serves as a reminder that the value depends on the value of $\vec{\nu}^{(i)}$. In other words, the result of applying the NPA linearizing transformation to a projection-equation system is a ***linear*** projection-equation system.

We extend the definition of regularizing transformation τ_{Reg} from Defn. 4.2 as follows:

$$\frac{Y_k = \text{Project}\left(a_k^{(i)} \oplus \bigoplus_j \bigoplus_l \left(b_{j,k,l}^{(i)} \otimes Y_j \otimes c_{j,k,l}^{(i)}\right)\right)}{Z_k = \text{Project}_{\mathcal{T}}\left(\begin{array}{c}\left(\underline{1}^t \odot a_k^{(i)}\right) \\ \oplus_{\mathcal{T}} \bigoplus_{\mathcal{T}\, j} \bigoplus_{\mathcal{T}\, l} \left(Z_j \otimes_{\mathcal{T}} \left((b_{j,k,l}^{(i)})^t \odot c_{j,k,l}^{(i)}\right)\right)\end{array}\right)} \; \tau_{\text{REG}}$$

where Z_k and Z_j are variables that take on values from tensor-product semiring $\mathcal{S}_{\mathcal{T}}$.[13]

Turning to the case of tensor-transpose relational weights for use with a predicate-abstraction domain, a suitable merge function can be defined as follows:

$$\begin{aligned} M_{\mathcal{T}}(T_1, T_2) &= T_1 \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(T_2) \\ \text{Project}_{\mathcal{T}}(T) &= \{(g_1', m_1, g_2, m_2, g_1, m_1, g_2', m_2) \mid T(g_1', l, g_2, l, g_1, l_1, g_2', l_2')\}. \end{aligned} \tag{69}$$

If one thinks of $T(G_1', L_1', G_2, L_2, G_1, L_1, G_2', L_2')$ as defined by a formula with eight sets of free variables G_1', L_1', G_2, L_2, G_1, L_1, G_2', and L_2', $\text{Project}_{\mathcal{T}}(T(G_1', L_1', G_2, L_2, G_1, L_1, G_2', L_2'))$ can also be expressed as follows:

$$\begin{aligned} &(\exists L_1', L_2, L_1, L_2' \colon (T(G_1', L_1', G_2, L_2, G_1, L_1, G_2', L_2') \wedge (L_1' = L_2))) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \wedge (L_1 = L_1') \wedge (L_2 = L_2'). \end{aligned}$$

The intuition behind Eqn. (69) is as follows:

—The conjunction $\ldots \wedge (L_1' = L_2)$ inside the existential quantifier performs a kind of "eager" detensor-transpose on the portion of the relation that summarizes the callee's transformation on its local variables.

—The existential quantification of L_1', L_2, L_1, and L_2', havocs the locals (in a manner similar to the existential quantification of L and L' in Eqn. (65)).

—The conjuncts $\ldots \wedge\ (L_1 = L_1') \wedge (L_2 = L_2')$ set the summary relation for the callee's effect on the caller's locals to the identity relation (in a manner similar to the conjunct $\ldots \wedge (L = L')$ in Eqn. (65)).

[13]An alternative way to express the tensored regularizing transformation is that the tensored differential of $f_i(\vec{X})$ can be obtained by the rule given in Defn. 4.18, extended with one more case for Project:

$$\boxed{\mathcal{D}^{\mathcal{T}}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Z}) = \text{Project}_{\mathcal{T}}(\mathcal{D}^{\mathcal{T}}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{Z})) \quad \text{if } f_i = \text{Project}(g)}$$

LEMMA 8.8. *The $M_{\mathcal{T}}$ operation defined in Eqn. (69) is an acceptable merge function for $\mathcal{S}_{\mathcal{T}}$, in the sense of Defn. 8.1. In addition, Project$_{\mathcal{T}}$ from Eqn. (69) has the following properties: for all $a, a_i, b \in \mathcal{S}_{\mathcal{T}}$ and $I \subseteq \mathbb{N}$,*

$$\begin{aligned}
\mathit{Project}_{\mathcal{T}}\left(\bigoplus_{i\in I}{}_{\mathcal{T}}\, a_i\right) &= \bigoplus_{i\in I}{}_{\mathcal{T}}\, \mathit{Project}_{\mathcal{T}}(a_i) && (70)\\
\mathit{Project}_{\mathcal{T}}(\mathit{Project}_{\mathcal{T}}(a)) &= \mathit{Project}_{\mathcal{T}}(a) && (71)\\
\mathit{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b) &= \mathit{Project}_{\mathcal{T}}(a \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b)) && (72)\\
&= \mathit{Project}_{\mathcal{T}}(\mathit{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} b) && (73)
\end{aligned}$$

PROOF. See App. C. □

These properties allow us to push occurrences of Project$_{\mathcal{T}}$ down to tensor-product-semiring constants, as shown in the following lemma:

LEMMA 8.9. *For all Z defined by an equation of the form*

$$Z = \mathit{Project}_{\mathcal{T}}\left(a \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} (Z_i \otimes_{\mathcal{T}} b_i)\right),$$

(1) $\mathit{Project}_{\mathcal{T}}(Z) = Z$

(2) Project$_{\mathcal{T}}$ can be pushed down to semiring constants to obtain

$$Z = \mathit{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} (Z_i \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b_i)).$$

PROOF. Part (1):

$$\begin{aligned}
Z &= \text{Project}_{\mathcal{T}}\left(a \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} (Z_i \otimes_{\mathcal{T}} b_i)\right)\\
&= \text{Project}_{\mathcal{T}}\left(\text{Project}_{\mathcal{T}}\left(a \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} (Z_i \otimes_{\mathcal{T}} b_i)\right)\right) && \text{by Eqn. (71)}\\
&= \text{Project}_{\mathcal{T}}(Z) && \text{by assumption}
\end{aligned}$$

Part (2):

$$\begin{aligned}
Z &= \text{Project}_{\mathcal{T}}\left(a \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} (Z_i \otimes_{\mathcal{T}} b_i)\right)\\
&= \text{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} \text{Project}_{\mathcal{T}}(Z_i \otimes_{\mathcal{T}} b_i) && \text{by Eqn. (70)}\\
&= \text{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} \text{Project}_{\mathcal{T}}(\text{Project}_{\mathcal{T}}(Z_i) \otimes_{\mathcal{T}} b_i) && \text{by Part (1)}\\
&= \text{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} (\text{Project}_{\mathcal{T}}(Z_i) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b_i)) && \text{by Eqn. (73)}\\
&= \text{Project}_{\mathcal{T}}(a) \oplus_{\mathcal{T}} \bigoplus_{i}{}_{\mathcal{T}} (Z_i \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b_i)) && \text{by Eqn. (71)}
\end{aligned}$$

□

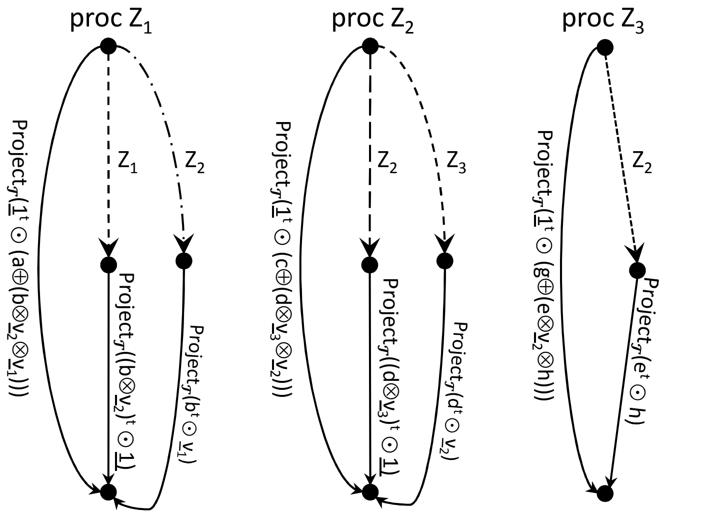


Fig. 10. Graphical depiction of the equation system that is produced for Eqn. (66) after occurrences of $\text{Project}_{\mathcal{T}}$ are introduced after step 2 of Alg. 7.1.

Thus, Lem. 8.9(2) allows each equation of the form

$$Z_k = \text{Project}_{\mathcal{T}}\left(\begin{array}{c}\left(\underline{1}^t \odot a_k^{(i)}\right)\\ \oplus_{\mathcal{T}} \bigoplus_{\mathcal{T}\, j} \bigoplus_{\mathcal{T}\, l} \left(Z_j \otimes_{\mathcal{T}} \left((b_{j,k,l}^{(i)})^t \odot c_{j,k,l}^{(i)}\right)\right)\end{array}\right)$$

to be rewritten so that all occurrences of $\text{Project}_{\mathcal{T}}$ have been pushed down to the tensor-product-semiring constants to obtain an equation system in which all equations have the form[14]

$$Z_k = \begin{array}{c}\text{Project}_{\mathcal{T}}\left(\underline{1}^t \odot a_k^{(i)}\right)\\ \oplus_{\mathcal{T}} \bigoplus_{\mathcal{T}\, j} \bigoplus_{\mathcal{T}\, l} \left(Z_j \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}\left((b_{j,k,l}^{(i)})^t \odot c_{j,k,l}^{(i)}\right)\right)\end{array} \tag{74}$$

The significance of having an equation system in which all equations have the form shown in Eqn. (74) is two-fold: (i) $\text{Project}_{\mathcal{T}}$ is applied exclusively to constants in $\mathcal{S}_{\mathcal{T}}$, and (ii) the equation system is left-recursive. Consequently, one can apply Tarjan's algorithm to this system of equations over $\vec{Z}$, as is done in step 4 of Alg. 7.1.

[14]Alternatively, for tensor-transpose relational weights, the act of pushing down of occurrences of $\text{Project}_{\mathcal{T}}$ can be built into τ_{Reg}:

$$\frac{Y_k = \text{Project}\left(a_k^{(i)} \oplus \bigoplus_j \bigoplus_l \left(b_{j,k,l}^{(i)} \otimes Y_j \otimes c_{j,k,l}^{(i)}\right)\right)}{Z_k = \begin{array}{c}\text{Project}_{\mathcal{T}}\left(\underline{1}^t \odot a_k^{(i)}\right)\\ \oplus_{\mathcal{T}} \bigoplus_{\mathcal{T}\, j} \bigoplus_{\mathcal{T}\, l} \left(Z_j \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}\left((b_{j,k,l}^{(i)})^t \odot c_{j,k,l}^{(i)}\right)\right)\end{array}}\ \tau_{\text{Reg}}$$

EXAMPLE 8.10. *For Eqn. (66), we would obtain the following equations (see also Fig. 10):*

$$\begin{aligned}
Z_1 &= \begin{pmatrix} \quad \mathit{Project}_{\mathcal{T}}(\underline{1}^t \odot (a \oplus (b \otimes \underline{\nu}_2 \otimes \underline{\nu}_1))) \\ \oplus_{\mathcal{T}} \ Z_1 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}((b \otimes \underline{\nu}_2)^t \odot \underline{1}) \\ \oplus_{\mathcal{T}} \ Z_2 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(b^t \odot \underline{\nu}_1) \end{pmatrix} \\
Z_2 &= \begin{pmatrix} \quad \mathit{Project}_{\mathcal{T}}(\underline{1}^t \odot (c \oplus (d \otimes \underline{\nu}_3 \otimes \underline{\nu}_2))) \\ \oplus_{\mathcal{T}} \ Z_2 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}((d \otimes \underline{\nu}_3)^t \odot \underline{1}) \\ \oplus_{\mathcal{T}} \ Z_3 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(d^t \odot \underline{\nu}_2) \end{pmatrix} \\
Z_3 &= \begin{pmatrix} \quad \mathit{Project}_{\mathcal{T}}(\underline{1}^t \odot (g \oplus (e \otimes \underline{\nu}_2 \otimes h))) \\ \oplus_{\mathcal{T}} \ Z_2 \otimes_{\mathcal{T}} \mathit{Project}_{\mathcal{T}}(e^t \odot h). \end{pmatrix}
\end{aligned}$$

Equivalently, one can elide all occurrences of Project in the original equation system $\mathcal{E}$, and in steps 1–4 of Alg. 7.1 work with an equation system with no occurrences of Project$(\cdot)$. Then, in step 5, occurrences of Project$_{\mathcal{T}}$ can be introduced by using the following method to create map m: each variable Z_i, $1 \leq i \leq n$, is mapped to the regular expression

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

EXAMPLE 8.11. *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*

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

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

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

In practice, to use NPA-TP[PA] to solve a projection-equation system $\mathcal{E} : \overrightarrow{W} = \mathrm{Project}(\vec{e}(\overrightarrow{W}))$, one would employ a variant of Alg. 7.1. However, for the purpose of arguing correctness, we will use the following more straightforward breakdown of NPA-TP[PA]:

(1) Alg. 8.12—given below—is used for solving linear projection-equation systems. (Alg. 8.12 is essentially Alg. 4.4 with a few minor wording changes.)

(2) We create the Newton sequence $\vec{\nu}^{(i)}$, defined by

$$\boxed{\begin{aligned} \vec{\nu}^{(0)} &= \mathrm{Project}(\vec{e}(\underline{\vec{0}})) \\ \vec{\nu}^{(i+1)} &= \lightning^{(t,\cdot)}(\vec{Z}^{(i)}) \end{aligned}} \qquad (75)$$

where $\vec{Z}^{(i)}$ is the least solution of Eqn. (74), and $\lightning^{(t,\cdot)}(\vec{Z}^{(i)})$ is obtained via Alg. 8.12.

ALGORITHM 8.12. *To solve a linear projection-equation system $\mathcal{L}$ over predicate-abstraction domain PA,*

(1) *Apply the version of τ_{Reg} for projection-equation systems to $\mathcal{L}$ to create a left-linear projection-equation system $\mathcal{L}_{\mathcal{T}}$ over the tensor-product semiring $PA_{\mathcal{T}}$.*

(2) *Use Tarjan's path-expression algorithm to find a regular expression Reg_k for each variable Z_k in Eqn. (74).*

(3) *Obtain $\vec{Z}$, the least solution to Eqn. (74): for each variable Z_k, evaluate Reg_k; i.e., $Z_k \leftarrow [\![Reg_k]\!]_{\mathcal{T}}$, where $[\![\cdot]\!]_{\mathcal{T}}$ denotes the interpretation of the regular-expression operators in $PA_{\mathcal{T}}$.*

(4) *Apply $\lightning^{(t,\cdot)}$ to each component of $\vec{Z}$; i.e., $\vec{\nu}_k^{(i+1)} \leftarrow \lightning^{(t,\cdot)}(Z_k)$.*

THEOREM 8.13. *Given a linear projection-equation system $\mathcal{L}$ over predicate-abstraction domain PA, Alg. 8.12 finds the least solution of $\mathcal{L}$.*

PROOF. See App. A.3. $\square$

THEOREM 8.14. *Suppose that $\mathcal{E} : \overrightarrow{W} = Project(\vec{e}(\overrightarrow{W}))$ is a projection-equation system over predicate-abstraction domain PA, and we create the sequence $\vec{\nu}^{(i)}$ defined by Eqn. (75). Then, for all i, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)} \sqsubseteq \overrightarrow{W}^{\star}$.*

PROOF. A projection-equation system over a predicate-abstraction domain is just a special case of an equation system over an admissible semiring, and therefore the assumptions of Thm. 8.14 meet the conditions of Thm. 8.5. Consequently, for all i, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)} \sqsubseteq \overrightarrow{W}^{\star}$. $\square$

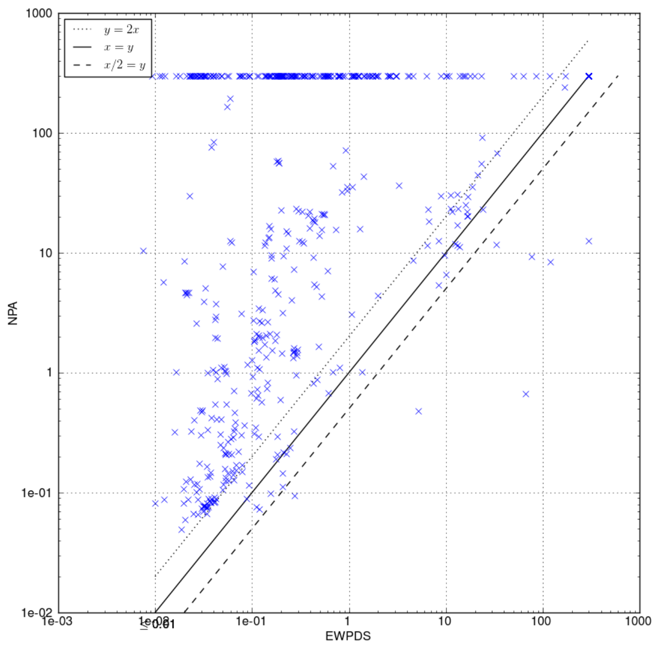
9. IMPLEMENTATION AND EXPERIMENTS

The Implemented Solvers. We experimented with implementations of NPA and NPA-TP, along with two non-Newton solvers. All four solvers were implemented using the Weighted Automaton Library (WALi) [Kidd et al. 2007].

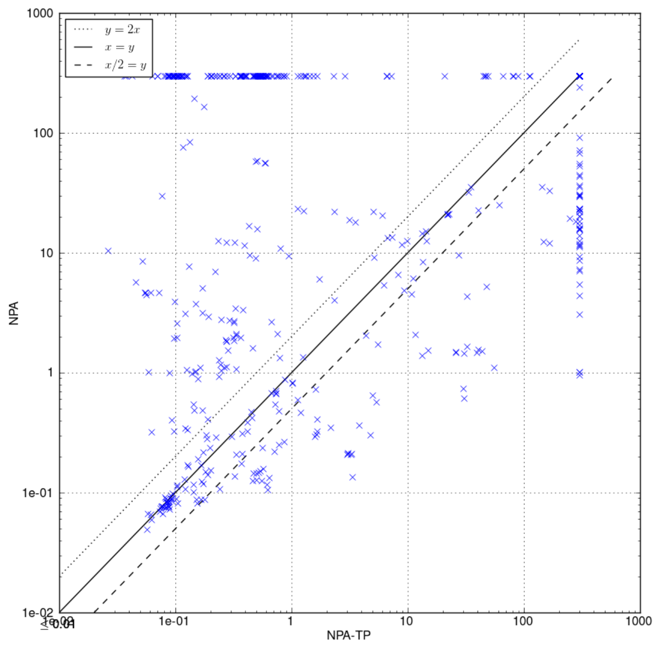
—One conventional solver used chaotic iteration (implemented using the post* algorithm for EWPDSs [Lal et al. 2005], followed by "path_summary" [Reps et al. 2005, Alg. 4]). The other used an adaptation of Tarjan's path-expression algorithm [Tarjan 1981a] for interprocedural analysis (the post* algorithm for FWPDSs [Lal and Reps 2006], followed by path_summary). We refer to these as "EWPDS" and "FWPDS," respectively.[15]

—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. 2.5)—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.

[15]In essence, the EWPDS algorithm performs chaotic iteration on the original equation system, whereas the FWPDS algorithm (i) applies Tarjan's path-expression algorithm to each CFG of the program to create a system of recursive equations with regular right-hand sides, and then (ii) solves the resulting equations via chaotic iteration. Both algorithms support merge functions.

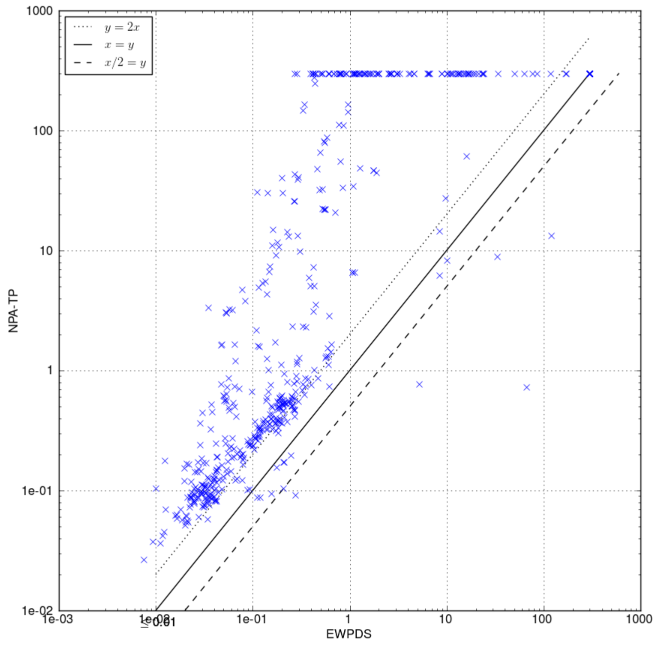


(a) EWPDS vs. NPA.
Geometric means: 8.75 → 31.6

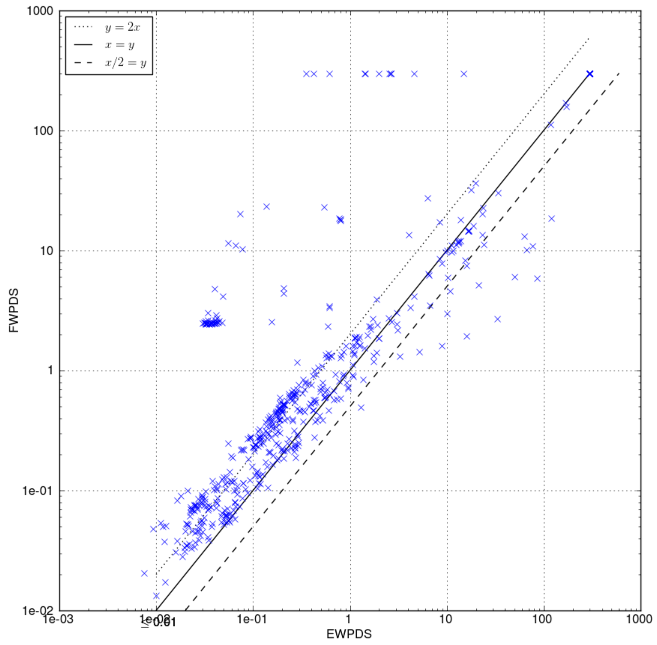


(b) NPA-TP vs. NPA.
Geometric means: 1.62 → 4.61

Fig. 11. Experimental results (first of three figures): 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 sets of Y/X values: (i) when Y and X both complete, and (ii) when non-completion counts as 300 seconds.

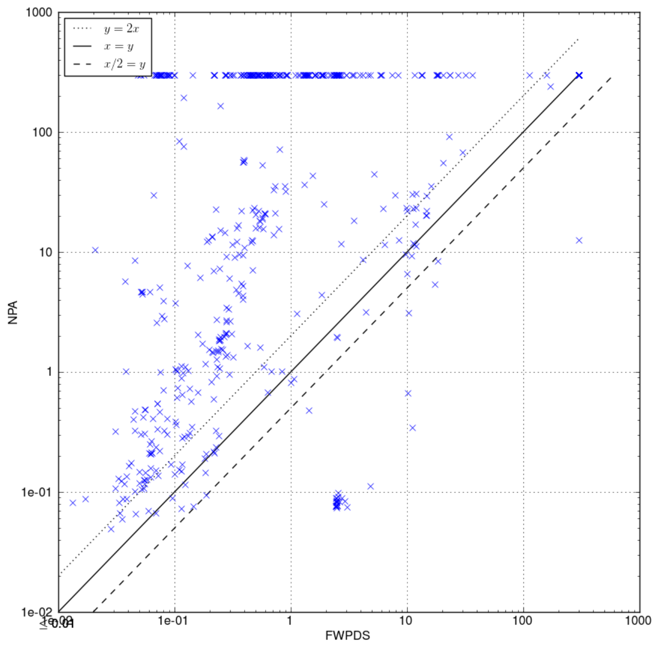


(a) EWPDS vs. NPA-TP.
Geometric means: 5.20 → 6.84

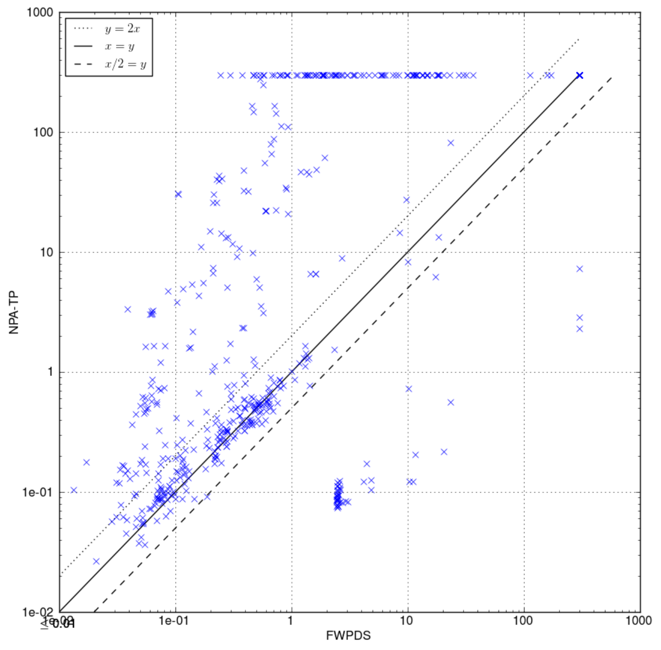


(b) EWPDS vs. FWPDS.
Geometric means: 2.22 → 2.15

Fig. 12. Experimental results continued (second of three figures).



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



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

Fig. 13. Experimental results continued (third of three figures).

EWPDS and FWPDS are standard solvers available in WALi; NPA and NPA-TP were implemented using primitives available in WALi. Except for EWPDS, all the solvers make use of WALi's implementation of Tarjan's path-expression method. In essence, Tarjan's method provides a way to solve a left-linear equation system. Moreover, when a weighted pushdown system contains no procedure calls, the regular expressions created for a WALi FWPDS are exactly those of Tarjan's method. NPA-TP needs to solve a sequence of related left-linear equation systems; the required regular expressions are obtained by creating an appropriate FWPDS that contains no procedure calls. Similarly, a call on path_summary can be seen as solving a left-linear equation system; to control for beneficial/detrimental effects because of the use of Tarjan's algorithm in both the FWPDS and NPA-TP solvers, an appropriate FWPDS that contains no procedure calls was used to implement the calls to path_summary in all runs of all four solvers.

The solvers use a predicate-abstraction domain implemented as a subclass of WALi's semiring class/interface. A semiring element corresponds to a predicate-transformer relation of the predicate-abstraction domain. For NPA-TP, we used an augmented abstract domain that supports $\cdot^t$, $\odot$, and $\lightning^{(t,\cdot)}$ on Kronecker products of predicate-transformer relations.

OBDD Variable-Ordering Issues. The predicate-transformer relations of the predicate-abstraction domain are represented with Ordered Binary Decision Diagrams (OBDDs) [Bryant 1986]. As is well-known, the size of the OBDD for a Boolean function is sensitive to the order chosen for the Boolean variables. When NPA-TP is applied to a predicate-abstraction problem, the nature of the operations that need to be performed places multiple constraints on what constitutes a good variable ordering. It is necessary to reconcile three competing issues; that is, the variable ordering must be suitable for

(1) Representing untensored relations, such as $\vec{\nu}^{(i)}$ and $\vec{\mu}$.
(2) Performing $\otimes_{\mathcal{T}}$ and $\oplus_{\mathcal{T}}$ of tensored relations (e.g., for evaluating $[\![m(Z_j)]\!]_{\mathcal{T}}\vec{\nu}^{(i)}$ in step 7b).
(3) Performing $\lightning^{(t,\cdot)}$ on a tensored relation—e.g., for applying $\lightning^{(t,\cdot)}$ to the result of $[\![m(Z_j)]\!]_{\mathcal{T}}\vec{\nu}^{(i)}$ in step 7b)—in particular, for representing the constraint $A' = B$ imposed during $\lightning^{(t,\cdot)}$.

We use the following notation to talk about a limited family of variable orderings:

Definition 9.1. *Let $A = [a_1, a_2, \ldots, a_N]$ and $B = [b_1, b_2, \ldots, b_N]$ be two disjoint, equal-size vocabularies of Boolean variables. The interleaved vocabulary $[a_1, b_1, a_2, b_2, \ldots, a_N, b_N]$ is denoted by $(\!|A \bowtie B|\!)$.*

Untensored relations are often nearly the identity relation (i.e., semiring identity $\underline{1}$); hence, $(\!|A' \bowtie A|\!)$ is a good ordering for addressing issue 1.

In general, a tensored value $T(A', B, A, B')$ represents a $\oplus_{\mathcal{T}}$ of multiple tensor-product summands, where each tensor-product operation couples two untensored relations, each of whose values equals an $\otimes$ over a sequence of relations:

$$T(A', B, A, B') = \bigoplus_{j}{}_{\mathcal{T}} \begin{pmatrix} (\otimes[R_{j,k}(A, A') \mid k \in [n..1]])^t \\ \odot\ (\otimes[S_{j,k}(B, B') \mid k \in [1..n]]) \end{pmatrix} \tag{76}$$

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

For a tensored value $T(A', B, A, B')$, we use the variable ordering $(\!|(A' \bowtie A) \bowtie (B \bowtie B')|\!)$. Because the respective bits of A' and B are close together, it is a good ordering for representing $A' = B$ and hence addresses issue 3.

The untensored values created by $(\otimes[R_{j,k}(A, A') \mid k \in [n..1]])$ and $(\otimes[S_{j,k}(B, B') \mid k \in [1..n]])$ in Eqn. (76) capture abstract-state transformations from widely separated parts of the program: typically, they represent the transformation from the entry of a procedure P up to a procedure call C, and the transformation after C to the exit of P, respectively). Because these transformations are not *a priori* related, $(\!|(A' \bowtie A) \bowtie (B \bowtie B')|\!)$ may not be a particularly good ordering. However, as observed above, untensored relations are often nearly the identity relation, in which case interleaving $(A' \bowtie A)$ with $(B \bowtie B')$ will keep OBDDs small, which addresses issue 2.

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?

Our test suite consisted of 584 Boolean programs from the 3,366 Boolean programs distributed with Microsoft's Static Driver Verifier [Static Driver Verifier]. 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 the submission to POPL 2016). 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. I. 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. I reports the average number of rounds (for completed runs only), including the final round needed to determine quiescence.

Figs. 11, 12, and 13 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. 11(a) shows that chaotic iteration (EWPDS) performs far better than NPA (geometric means: 8.75 → 31.6). Thus, at least for this test suite of Boolean programs, these results answer Question 2 in the negative.

—Fig. 11(b) shows that the implementation of NPA-TP performs better than NPA (geometric means: 1.62 → 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. 12(a) shows that NPA-TP is still slower than chaotic iteration (geometric means: 5.20 → 6.84), which answers Question 4 in the negative. However, we see that NPA-TP did better against chaotic iteration than NPA did.

—Fig. 12(b) shows that EWPDS is about 2x faster than FWPDS (geometric means: 2.22 → 2.15), although FWPDS is faster for some of the more compute-intensive problems.

—Fig. 13(a) shows that FWPDS is much faster than NPA (geometric means: 4.62 → 14.7).

—Fig. 13(b) shows that FWPDS is faster than NPA-TP (geometric means: 1.94 → 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

Tarjan's path-expression algorithm was used earlier by Lal and Reps [2006] in a much more straightforward algorithm for interprocedural dataflow analysis. (The algorithm is the one referred to as FWPDS in §9.) As in our method, they apply the path-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.

As mentioned in §3.2, if the extend ($\otimes$) operation is commutative, it is trivial to turn a linear problem into a left-linear problem, after which the path-expression algorithm can be applied directly (without having to introduce tensored values). Moreover, Esparza et al. proved that for an idempotent and commutative semiring, the least fixed-point of an equation system over N variables is obtained via NPA in at most N rounds [Esparza et al. 2010, Thm. 7.7].

Tensor Product and Detensor-Transpose. Admissible semirings were used by Lal et al. [2008] 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 via intersection of weighted transducers. An operation similar to $\lightning^{(t,\cdot)}$ was used to read out answers from the combined transducer.

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.

Lal et al. [2007] 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 [Lal et al. 2007, §4]. The problem can be recast using an LCFL equation system to which the algorithm of the present paper can be applied.

Grathwohl et al. [2014] 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.

LCFL Equation Systems Over Non-Commutative Semirings. In a survey paper on graph-reachability techniques in databases, Yannakakis [1990] considered several classes of **L-reachability problems** in directed graphs. Given a language $L \subseteq \Sigma^*$ and a directed graph G whose edges are labeled with elements of Σ, node n_2 is L-reachable from n_1 iff there exists a path π in G, from n_1 to n_2, whose edge labels, concatenated in the order the edges occur in π, form a word $w_\pi \in L$. Every CFL-reachability problem can be encoded as a special kind of Datalog program, called a *chain program* [Yannakakis 1990].

Yannakakis described an algorithm for solving all-pairs LCFL-reachability problems in time $O(ne)$, where n and e are the number of nodes and edges in G, respectively. The algorithm applies to linear chain programs, which means that it can also be used to solve an LCFL equation system over a given abstract domain of predicate-abstraction transformers. For instance, consider the same-generation query on a graph with edge labels $\{\text{up}, \text{down}, \text{flat}\}$:

$$\begin{aligned} \text{sg}(W,Z) &:- \ \text{up}(W,X), \text{sg}(X,Y), \text{down}(Y,Z) \\ \text{sg}(W,Z) &:- \ \text{flat}(W,Z) \end{aligned} \tag{77}$$

Eqn. (77) is a linear chain program. It is the kind of Datalog program that would arise when analyzing the following program with respect to some predicate-abstraction domain:

```
sg() {
    if (⋆)  $s_{\text{flat}}$
    else {
        $s_{\text{up}}$; sg(); $s_{\text{down}}$
    }
}
```

which has the linear equation system

$$X_{\text{sg}} \;=\; s_{\text{flat}} \oplus (s_{\text{up}} \otimes X_{\text{sg}} \otimes s_{\text{down}}).$$

In other words, the value of relation $\text{sg}(\cdot,\cdot)$ in Eqn. (77) is the summary for procedure sg, while the relations $\text{up}(\cdot,\cdot)$, $\text{down}(\cdot,\cdot)$, and $\text{flat}(\cdot,\cdot)$ are semiring constants—fixed predicate-abstraction transformers—for the statements s_{flat}, s_{up}, and s_{down}.

Yannakakis transforms Eqn. (77) by introducing new relations (variables) so that each right-hand side has at most two symbols, at most one of which is a variable:

$$\begin{aligned} \text{sg}(W,Z) &:- \ \text{up}(W,X), \text{new}(X,Z) \\ \text{new}(X,Z) &:- \ \text{sg}(X,Y), \text{down}(Y,Z) \\ \text{sg}(W,Z) &:- \ \text{flat}(W,Z) \end{aligned}$$

The transformed query is then evaluated bottom-up via semi-naive evaluation, which—in dataflow-analysis terminology—corresponds to performing chaotic iteration at the granularity of individual tuples.

In our experiments, semiring values—transformers of the predicate-abstraction domain—were implemented with BDDs, which does not mesh well with evaluation at the granularity of individual tuples. Therefore, LCFL problems in our implementation of NPA were solved using a method that involves reinterpretation of regular expressions (and thus the NPA and NPA-TP implementations used analogous techniques):

—The implementation of NPA creates a WALi weighted pushdown system that encodes the LCFL problem for a given Newton round, and applies WALi's FWPDS solver (which, as needed, evaluates the untensored regular expressions that occur on the right-hand sides of the LCFL equation system).

—The NPA-TP implementation uses the steps given in Alg. 7.1 to evaluate an appropriate set of tensored regular expressions, followed by $\lightning^{(t,\cdot)}$.

The technique of McNaughton and Yamada [1960] 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 [Tarjan 1981a].

Independently and contemporaneously with our work, Schlund [2016, §3.4.4] proposed using Kronecker products to solve the linear equations that arise when NPA is applied to non-commutative problems in which semiring elements are matrices with entries drawn from a commutative semiring. The predicate-abstraction domains that we used in our experiments fall into this category (see Defn. 2.2 and

§5). In our work, we have also given an algebraic characterization of the properties needed for the regularizing transformation to be applicable (see Defns. 4.1 and 4.2).

In follow-on work to the present paper, we have developed an analyzer [Kincaid et al. 2016] in which a value is a formula that defines a transition relation; transpose is the reverse of the relation that the formula defines; and tensor product is conjunction. The interpretation of Kleene-star (e.g., b^*) computes an overapproximation of the transitive closure of b by (i) extracting a recurrence relation from the formula for b, (ii) solving the recurrence relation, and (iii) returning a formula that expresses the solution [Farzan and Kincaid 2015]. (The interpretation of a tensored Kleene-star is similar, but involves four-vocabulary formulas.) For technical reasons, the value space does not meet the conditions of NPA-TP, but the basic ideas used are the same. In future work, we hope to apply similar ideas to programs that use recursion and/or loops to manipulate strings or arrays, as well as to the analysis of probabilistic programs (for computing expectations of variables modified in loops and recursive procedures).

Other Newtonian Solvers. A different implementation of NPA is discussed in [Schlund et al. 2013]. The two experiments reported were both with commutative semirings, for which NPA-TP is not needed.

Kafle et al. [2016] have created an implementation of a Newtonian solver for Constrained Horn Clauses (CHC). Compared with our implementation of NPA-TP, each has features that the other does not: (i) the CHC solver works with an abstract domain that is not an ω-continuous semiring (although the solver could be applied to Datalog), but (ii) the CHC solver has no analog of NPA-TP's regularizing transformation (Defn. 4.2). With respect to (ii), although the title of [Kafle et al. 2016] includes the phrase "using a linear Horn-clause solver," the paper says that they use "the solver described in [Kafle and Gallagher 2015] without refinement as a linear Horn clause solver." The latter solver does not include any special machinery for the special case of linear Horn clauses [Gallagher 2016].

In 1986, Ullman and Van Gelder [1986] studied the parallel complexity of Datalog, and gave an algorithm for evaluating Datalog programs under the PRAM model. The evaluation method is not described in terms of a syntactic transformation that differentiates Horn clauses, and the paper does not draw a connection with Newton's method (numerical or otherwise). However, when viewed from the proper vantage point, the Ullman/Van-Gelder evaluation method can be seen as a variant of NPA-TP:

—The algorithm follows the NPA paradigm of holding all but one of the right-hand-side relations of each rule fixed during a given evaluation round.

—The algorithm manipulates an "implication graph" whose nodes are all possible ground facts; each evaluation round adds some edges to the graph using a Newton-like criterion, and then performs transitive closure.

Note that in the instantiation of NPA-TP for predicate abstraction, the tensor-product-based coupling-operator doubles the arity of the relations involved in an equation system, and transitive closure is performed on 4-ary tensored relations. This aspect has a direct analog in the Ullman/Van-Gelder evaluation method: the implication graph has edges between binary facts (i.e., 2-tuple to 2-tuple), and can be thought of as a 4-ary tensored relation. Transitive closure is performed on the implication graph, and thus, just as in the instantiation of NPA-TP for predicate abstraction, transitive closure is performed over 4-tuples. In effect, Ullman and Van Gelder perform a Kleene-star of a tensored relation, and consequently their technique can be considered to be an instance of the NPA-TP semiring-based framework developed in the present paper.

Our work was motivated by the desire to apply an intraprocedural solver to the linearized problems that arise on successive Newton rounds. The motivation behind work by Ganty et al. [2016] is similar: they created a Newtonian solver for recursive programs over the integers (but also implicitly describe a version of their method that could be applied to bounded-height domains, such as predicate-abstraction domains). Their approach differs from ours; in their work, they perform a transformation that restructures the linearized, possibly recursive, multi-procedure program P_k obtained for Newton-round k into a non-recursive program P'_k. The transformation introduces occurrences of `havoc` and `assume` so that, during the analysis of P'_k, the various parts of P_k are analyzed in an order different from the order in which they would be executed in P_k. P'_k can be analyzed by any analyzer that can handle `havoc` and `assume`. In principle, in-line expansion could be performed to turn P'_k into a single-procedure program, which could then be analyzed by any intraprocedural analyzer that can handle `havoc` and `assume`. In contrast, NPA-TP relies on the properties of a tensor-product domain to lift a linear equation system to a left-linear—and hence regular—equation system.

The relationship of our work to Ganty et al. [2016] is similar to the relationship of Lal et al. [2008] to Lal and Reps [2009]. The subjects in the two pairs of papers are different—interprocedural analysis of sequential programs vs. analysis of concurrent programs under a context-switch bound, respectively. However, in each pair, the first paper uses an algebraic formalism based on tensor product, and the second uses a code transformation that introduces occurrences of `havoc` and `assume` to restructure the code so that

(1) The analysis of the transformed program provides information about the properties of the original untransformed program.
(2) The transformed program can be analyzed by a simpler sequential analyzer (an intraprocedural analyzer in the case of Ganty et al. [2016] and an interprocedural analyzer in the case of Lal and Reps [2009]).

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. 11(a)). Our goal of speeding up Newtonian program analysis was achieved (Fig. 11(b)); however, NPA-TP is still slower than EWPDS (Fig. 12(a)). NPA-TP is also slower than FWPDS (Fig. 13(b)), a more straightforward way of using Tarjan's algorithm for interprocedural dataflow analysis [Lal and Reps 2006]. 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. 12(b)). 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); J.P. Gallagher and P. Ganty for help in understanding the relationship of NPA-TP with the methods of Kafle et al. [2016] and Ganty et al. [2016]; and P. Koutris for pointing out the relationship of NPA-TP with [Ullman and Van Gelder 1986]. We thank the POPL 2016 reviewers for their feedback on Reps et al. [2016] and the TOPLAS referees for their comments on the submission.

REFERENCES

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

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

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

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

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

Droste, M., Kuich, W., and Vogler, H., Eds. 2009. *Handbook of Weighted Automata.* Springer-Verlag.

Elder, M., Lim, J., Sharma, T., Andersen, T., and Reps, T. 2014. Abstract domains of affine relations. *Trans. on Prog. Lang. and Syst. 36,* 4 (Jan.).

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

Esparza, J., Kiefer, S., and Luttenberger, M. 2010. Newtonian program analysis. *J. ACM 57,* 6.

Farzan, A. and Kincaid, Z. 2015. Compositional recurrence analysis. In *FMCAD*.

Gallagher, J. 2016. Personal communication.

Ganty, P., Iosif, R., and Konečný, F. 2016. Underapproximation of procedure summaries for integer programs. *Softw. Tools for Tech. Transfer*. Corrected version available as arXiv:1210.4289v3 (10.1007/s10009-016-0420-7).

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

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

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

Kafle, B. and Gallagher, J. 2015. Horn clause verification with convex polyhedral abstraction and tree automata-based refinement. *Computer Languages, Systems & Structures*. http://dx.doi.org/10.1016/j.cl.2015.11.00.

Kafle, B., Gallagher, J., and Ganty, P. 2016. Solving non-linear Horn clauses using a linear Horn clause solver. In *Proc. 3rd Workshop on Horn Clauses for Verification and Synthesis*.

Kam, J. and Ullman, J. 1976. Global data flow analysis and iterative algorithms. *J. ACM 23,* 1, 158–171.

Kam, J. and Ullman, J. 1977. Monotone data flow analysis frameworks. *Acta Inf. 7,* 3, 305–318.

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

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

Kincaid, Z., Breck, J., Forouhi Boroujeni, A., and Reps, T. 2016. Compositional recurrence analysis revisited. Tech. Rep. TR-1840, Comp. Sci. Dept., Univ. of Wisconsin, Madison, WI. Oct.

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

Knuth, D. 1977. A generalization of Dijkstra's algorithm. *Inf. Proc. Let. 6,* 1, 1–5.

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

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

Lal, A. and Reps, T. 2009. Reducing concurrent analysis under a context bound to sequential analysis. *Formal Methods in System Design 35,* 1, 73–97.

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

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

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

Litvinov, G., Rodionov, A., Sergeev, S., and Sobolevski, A. 2013. Universal algorithms for solving the matrix Bellman equations over semirings. *Soft Computing 17,* 10, 1767–1785.

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

Möncke, U. and Wilhelm, R. 1991. Grammar flow analysis. In *Attribute Grammars, Applications and Systems, (Int. Summer School SAGA)*. 151–186.

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

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

Ramalingam, G. 1996. *Bounded Incremental Computation.* Springer-Verlag.

Reps, T., Horwitz, S., and Sagiv, M. 1995. Precise interprocedural dataflow analysis via graph reachability. In *POPL*. 49–61.

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

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

Reps, T., Turetsky, E., and Prabhu, P. 2016. Newtonian program analysis via tensor product. In *POPL*.

Schlund, M. 2016. Algebraic systems of fixpoint equations over semirings: Theory and applications. Ph.D. thesis, Lehrstuhl für Informatik VII, Technischen Universität München, Munich, Ger.

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

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

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

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

Tarjan, R. 1981a. Fast algorithms for solving path problems. *J. ACM 28,* 3, 594–614.

Tarjan, R. 1981b. A unified approach to path problems. *J. ACM 28,* 3, 577–593.

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

Ullman, J. and Van Gelder, A. 1986. Parallel complexity of logical query programs. In *Foundations of Computer Science*.

Vyssotsky, V. and Wegner, P. 1963. 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).

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

A. CORRECTNESS PROOFS

The paper presents two extensions to NPA and three variants of NPA-TP, which are discussed in the following sections of the paper:

Operators	NPA	NPA-TP
$\oplus, \otimes$	Esparza et al. [Esparza et al. 2008; 2010]	§4
$\oplus, \otimes, ^*$	§6.2	§6.2
$\oplus, \otimes, ^*$, Project	§8.1	§8.2

In general, we consider an equation system $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ over admissible semiring $\mathcal{S}$, possibly with occurrences of Kleene-star and Project in the component functions of $\vec{f}$.[16] We use $\vec{X}^\star$ to denote the least solution of $\mathcal{E}$ (i.e., $\vec{X}^\star$ is the least fixed-point of $\mathcal{E}$ and equals $\bigoplus_{i=0}^{\infty} \vec{f}^{\,i}(\underline{\vec{0}})$). $\vec{X}^\star$ exists because we are working with an ω-continuous semiring. For the NPA cases, correctness means

—The Newton sequence obtained using the respective extended definition of the multivariate differential converges to $\vec{X}^\star$.

For the three NPA-TP cases, correctness needs to be argued in two parts:

[16]An interprocedural dataflow-analysis problem over admissible semiring $\mathcal{S}$ can be converted into an equation system of the above form by applying Tarjan's path-expression algorithm to the CFG of each procedure—as is done in step 1 of Alg. 7.1. The result is an equation system $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ in which

—the variables in $\vec{X}$ correspond to the procedures of the program, and

—the right-hand side of each equation is an expression over variables in $\vec{X}$, constants from $\mathcal{S}$, and the operators $\oplus$, $\otimes$, *, and Project.

That is, $\mathcal{E} = \{X_j = \text{Rhs}_j(\vec{X}) \mid X_j \in \vec{X}\}$. For NPA with Project, each expression Rhs_j has an occurrence of $\text{Project}(X_k)$ for each call on procedure k in procedure j.

(1) We need to show that our method for solving an LCFL equation system $\mathcal{L}$ over an admissible semiring finds the least solution of $\mathcal{L}$.
(2) We need to show that the Newton sequence obtained using
    (a) the respective extended definition of the multivariate differential, and
    (b) the LCFL equation solver
    converges to $\vec{X}^\star$.

The remainder of this appendix is structured around the entries in the following table that do not follow immediately from Esparza et al. [Esparza et al. 2010, Thm. 3.9] and Thm. 4.17:

Operators		NPA-TP	
	NPA	Part 1	Part 2
$\oplus, \otimes$	[Esparza et al. 2010], Thm. 3.9	Thm. 4.17 (§4.5)	[Esparza et al. 2010], Thm. 3.9
$\oplus, \otimes, ^*$	Thm. 8.5 (App. A.1)		Thm. 8.5 (App. A.1)
$\oplus, \otimes, ^*, \text{Project}$		Thm. 8.14 (§8.2 and App. A.3)	

Fortunately, matters boil down to two theorems:

—Thm. 8.5 (App. A.1), which extends NPA to equation systems that contain occurrences of Kleene-star and Project.
—Thm. 8.14 (§8.2), which extends NPA to projection-equation systems—equation systems of the special form introduced in §8.2—over a predicate-abstraction domain.

Thm. 8.14 relies on Thm. 8.13, whose proof is given in App. A.3. Thm. 8.13 is similar to Thm. 4.17—the proof of correctness of Alg. 4.4 for solving an LCFL equation system over an admissible semiring—except that it applies to a linear projection-equation system over a predicate-abstraction domain.

For the two NPA cases and Part 2 of each of the three NPA-TP cases, our goal is to relate Kleene iterate $\vec{\kappa}^{(i)}$, Newton iterate $\vec{\nu}^{(i)}$, and $\vec{X}^\star$—in particular, to show that $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)} \sqsubseteq \vec{X}^\star$ holds. Esparza et al. proved a similar theorem [Esparza et al. 2010, Thm. 3.9], but because we now allow an equation system to contain occurrences of Kleene-star and Project, we need to prove the result for such extended systems.

Note that the operators $\oplus$, $\otimes$, and Project all distribute over $\oplus$, in each argument position:

—For $\oplus$, distributivity follows from associativity, commutativity, and idempotence.
—The distributivity of $\otimes$ over $\oplus$ is assumed in Defn. 2.1.
—The distributivity of Project over $\oplus$ follows from Defn. 8.1(2) and the definition of Project(a) as $M(\underline{1}, a)$.

Distributivity implies that $\oplus$, $\otimes$, and Project are all monotonic in each argument position. For instance, suppose that $a \sqsubseteq b$ and operation $p(\ldots, v_i, \ldots)$ is distributive in the i^{th} position. Then $p(\ldots, b, \ldots) = p(\ldots, a \oplus b, \ldots) = p(\ldots, a, \ldots) \oplus p(\ldots, b, \ldots)$, and hence $p(\ldots, a, \ldots) \sqsubseteq p(\ldots, b, \ldots)$.

Kleene-star does not distribute over $\oplus$, but is monotonic: let $a \sqsubseteq b$; by the monotonicity of $\otimes$, for all $i \in \mathbb{N}$, $a^i \sqsubseteq b^i$; then $a^* = \bigoplus_{i\in\mathbb{N}} a^i \sqsubseteq \bigoplus_{i\in\mathbb{N}} (a^i \oplus b^i) = \bigoplus_{i\in\mathbb{N}} b^i$ $= b^*$.

By the argument above, each subexpression of an expression over variables in $\vec{X}$, constants from $\mathcal{S}$, and the operators $\oplus$, $\otimes$, *, and Project is monotonic in each argument. In particular, each subexpression of each component function f_i is monotonic, as are each subexpression of $\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y})$, $\mathcal{D}f_i|_{\underline{\vec{\nu}}}(\vec{Y})$, and $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$. In addition, $\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{Y})$, $\mathcal{D}f_i|_{\underline{\vec{\nu}}}(\vec{Y})$, and $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$ are each monotonic in $\underline{\vec{\nu}}$.

A.1 NPA with Operators $\oplus$, $\otimes$, *, and Project

For brevity, we combine the proofs for (i) NPA with $\oplus$, $\otimes$, and *, and (ii) NPA with $\oplus$, $\otimes$, *, and Project.

To be clear about the algorithm we are considering, we state here the algorithm for NPA with $\oplus$, $\otimes$, *, and Project.

Algorithm A.1. *The input is an equation system $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ over admissible semiring $\mathcal{S}$, possibly with occurrences of Kleene-star and Project in the component functions of $\vec{f}$. Let $\vec{X}$ denote the set of variables in $\mathcal{E}$.*

```
(1) $i \leftarrow 0; \vec{\mu} \leftarrow \vec{f}(\underline{\vec{0}})$
(2) Repeat
    (a) $\vec{\nu}^{(i)} = \vec{\mu}$
    (b) $\vec{\mu}$ = the least solution of
            $\vec{Y} = \vec{f}(\vec{\nu}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y}),$   (5)
        where $\mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}$ is extended so that Eqn. (7) of Defn. 2.5 incorporates both
        (i) the rule for Kleene-star from §6.2, and (ii) the rule for Project from
        §8.1.
    (c) $i \leftarrow i + 1$
    until $(\vec{\nu}^{(i-1)} = \vec{\mu})$
(3) Return $\vec{\mu}$
```

Lemma A.2. *For all $\vec{w}$ such that $\underline{\vec{\nu}} \sqsubseteq \vec{w}$, $\mathcal{D}_{X_j} f_i|_{\underline{\vec{\nu}}}(\vec{w}) \sqsubseteq f_i(\vec{w})$.*

PROOF. By structural induction on the expression for f_i.

$$\begin{array}{lrll}
\text{if } f_i = s \in \mathcal{S}: & \text{LHS} & = \underline{0} \sqsubseteq s = \text{RHS} & \\
\text{if } f_i = X_k \text{ and } k \neq j: & \text{LHS} & = \underline{0} \sqsubseteq w_k = \text{RHS} & \\
\text{if } f_i = X_j: & \text{LHS} & = w_j = \text{RHS} & \\
\text{if } f_i = g \otimes h: & \text{LHS} & = \begin{pmatrix} \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\, g(\underline{\vec{\nu}}) \otimes \mathcal{D}_{X_j} h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} & \\
& & \sqsubseteq g(\vec{w}) \otimes h(\underline{\vec{\nu}}) \oplus g(\underline{\vec{\nu}}) \otimes h(\vec{w}) & \text{by the ind. hyp.} \\
& & \sqsubseteq g(\vec{w}) \otimes h(\vec{w}) \oplus g(\vec{w}) \otimes h(\vec{w}) & \text{by monotonicity} \\
& & = g(\vec{w}) \otimes h(\vec{w}) & \\
& & = \text{RHS} & \\
\text{if } f_i = \bigoplus_{k \in K} g_k: & \text{LHS} & = \bigoplus_{k \in K} \mathcal{D}_{X_j} g_k|_{\underline{\vec{\nu}}}(\vec{w}) & \\
& & \sqsubseteq \bigoplus_{k \in K} g_k(\vec{w}) & \text{by the ind. hyp.} \\
& & = \text{RHS} & \\
\text{if } f_i = g^*: & \text{LHS} & = g(\underline{\vec{\nu}})^* \otimes \mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes g(\underline{\vec{\nu}})^* & \\
& & \sqsubseteq g(\underline{\vec{\nu}})^* \otimes g(\vec{w}) \otimes g(\underline{\vec{\nu}})^* & \text{by the ind. hyp.} \\
& & \sqsubseteq g(\vec{w})^* \otimes g(\vec{w}) \otimes g(\vec{w})^* & \text{by monotonicity} \\
& & = \bigoplus_{i=1}^{\infty} g(\vec{w})^i & \\
& & = \underline{0} \oplus \left(\bigoplus_{i=1}^{\infty} g(\vec{w})^i\right) & \text{for all } a,\ \underline{0} \oplus a = a \\
& & = \bigoplus_{i=0}^{\infty} g(\vec{w})^i & \\
& & = \text{RHS} & \\
\text{if } f_i = \text{Project}(g): & \text{LHS} & = \text{Project}(\mathcal{D}_{X_j} g|_{\underline{\vec{\nu}}}(\vec{w})) & \\
& & \sqsubseteq \text{Project}(g(\vec{w})) & \text{by the ind. hyp.} \\
& & = \text{RHS} &
\end{array}$$

□

COROLLARY A.3. *For all $\vec{w}$ such that $\underline{\vec{\nu}} \sqsubseteq \vec{w}$, $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{w}) \sqsubseteq \vec{f}(\vec{w})$.*

PROOF. Immediate from the definition of $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{w})$ (Defn. 2.5) and Lem. A.2. □

LEMMA A.4. *Let*

(1) $\underline{\vec{\nu}}$ be a vector such that $\underline{\vec{\nu}} \sqsubseteq \vec{X}^\star$.

(2) $\vec{Y}^\star$ denote the least solution of $\vec{Y} = \vec{f}(\underline{\vec{\nu}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$.

Then $\vec{Y}^\star \sqsubseteq \vec{X}^\star$.

PROOF. Let $\vec{F}$ denote the function $\lambda\vec{y}.\vec{f}(\underline{\vec{\nu}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{y})$. By Kleene's fixed-point theorem, $\vec{Y}^\star = \bigoplus_{i=0}^{\infty} \vec{F}^i(\underline{\vec{0}})$. We proceed by induction.

Base case. $\vec{F}^0(\underline{\vec{0}}) = \underline{\vec{0}} \sqsubseteq \vec{X}^\star$.

Induction step. Assume that

$$\vec{F}^k(\underline{\vec{0}}) \sqsubseteq \vec{X}^\star. \tag{78}$$

$\vec{F}$ is monotonic, so by applying $\vec{F}$ to both sides of Eqn. (78), we have

$$\begin{aligned}
\vec{F}^{k+1}(\vec{\underline{0}}) &\sqsubseteq \vec{F}(\vec{X}^\star) \\
&= \vec{f}(\vec{\underline{\nu}}) \oplus \mathcal{D}\vec{f}|_{\vec{\underline{\nu}}}(\vec{X}^\star) \\
&\sqsubseteq \vec{f}(\vec{\underline{\nu}}) \oplus \vec{f}(\vec{X}^\star) && \text{by premise 1 and Cor. A.3} \\
&= \vec{f}(\vec{X}^\star) && \text{by premise 1 and the monotonicity of } \vec{f} \\
&= \vec{X}^\star && \vec{X}^\star \text{ is a fixed-point of } \vec{f}
\end{aligned}$$

Thus, for all $i \in \mathbb{N}$, $\vec{F}^i(\vec{\underline{0}}) \sqsubseteq \vec{X}^\star$, and hence $\vec{Y}^\star \sqsubseteq \vec{X}^\star$. □

LEMMA A.5. *Suppose that $\vec{a} \sqsubseteq \vec{b}$, $\vec{Y}^\star_{\vec{a}}$ is the least solution to $\vec{Y} = \vec{f}(\vec{a}) \oplus \mathcal{D}\vec{f}|_{\vec{a}}(\vec{Y})$, and $\vec{Y}^\star_{\vec{b}}$ is the least solution to $\vec{Y} = \vec{f}(\vec{b}) \oplus \mathcal{D}\vec{f}|_{\vec{b}}(\vec{Y})$. Then $\vec{Y}^\star_{\vec{a}} \sqsubseteq \vec{Y}^\star_{\vec{b}}$.*

PROOF. Let $\vec{F}_{\vec{a}}$ and $\vec{F}_{\vec{b}}$ denote the functions $\lambda\vec{y}.\vec{f}(\vec{a}) \oplus \mathcal{D}\vec{f}|_{\vec{a}}(\vec{y})$ and $\lambda\vec{y}.\vec{f}(\vec{b}) \oplus \mathcal{D}\vec{f}|_{\vec{b}}(\vec{y})$, respectively. By the monotonicity of $\vec{f}$ and $\mathcal{D}\vec{f}|_{\vec{\underline{\nu}}}(\vec{Y})$, $\vec{F}_{\vec{a}} \sqsubseteq \vec{F}_{\vec{b}}$. By an easy induction, for all $j \in \mathbb{N}$, $\vec{F}^j_{\vec{a}}(\vec{\underline{0}}) \sqsubseteq \vec{F}^j_{\vec{b}}(\vec{\underline{0}})$, and hence

$$\vec{Y}^\star_{\vec{a}} = \bigoplus_{j=0}^{\infty} \vec{F}^j_{\vec{a}}(\vec{\underline{0}}) \sqsubseteq \bigoplus_{j=0}^{\infty} \vec{F}^j_{\vec{b}}(\vec{\underline{0}}) = \vec{Y}^\star_{\vec{b}}.$$

□

COROLLARY A.6. *The elements $\{\vec{\underline{\nu}}^{(k)}\}$ of the sequence obtained using Eqns. (4) and (5) have the property that, for all $i \in \mathbb{N}$, $\vec{\underline{\nu}}^{(i)} \sqsubseteq \vec{\underline{\nu}}^{(i+1)}$.*

PROOF. *Base case.* From the form of the equation $\vec{Y} = \vec{f}(\vec{\underline{\nu}}) \oplus \mathcal{D}\vec{f}|_{\vec{\underline{\nu}}}(\vec{Y})$ and the monotonicity of $\vec{f}$, we know that $\vec{Y}^\star \sqsupseteq \vec{f}(\vec{\underline{\nu}}) \sqsupseteq \vec{f}(\vec{\underline{0}})$, and hence for all $i \in \mathbb{N}$, $\vec{\underline{\nu}}^{(i)} \sqsupseteq \vec{f}(\vec{\underline{0}})$. In particular,

$$\vec{\underline{\nu}}^{(1)} \sqsupseteq \vec{f}(\vec{\underline{0}}) = \vec{\underline{\nu}}^{(0)}.$$

Induction step. Assume that $\vec{\underline{\nu}}^{(k-1)} \sqsubseteq \vec{\underline{\nu}}^{(k)}$; show that $\vec{\underline{\nu}}^{(k)} \sqsubseteq \vec{\underline{\nu}}^{(k+1)}$.
By Lem. A.5,

$$\vec{\underline{\nu}}^{(k)} = \vec{Y}^\star_{\vec{\underline{\nu}}^{(k-1)}} \sqsubseteq \vec{Y}^\star_{\vec{\underline{\nu}}^{(k)}} = \vec{\underline{\nu}}^{(k+1)}.$$

Therefore, for all $i \in \mathbb{N}$, $\vec{\underline{\nu}}^{(i)} \sqsubseteq \vec{\underline{\nu}}^{(i+1)}$. □

Theorem 8.5. *Let $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ be an equation system whose right-hand-side terms can contain both regular operators and occurrences of the operator Project, and whose least fixed-point is $\vec{X}^\star$. Then for all i, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)} \sqsubseteq \vec{X}^\star$.*

PROOF. Let $\vec{Y}^\star_k$ denote the least solution of

$$\vec{Y} = \vec{f}(\vec{\underline{\nu}}^{(k)}) \oplus \mathcal{D}\vec{f}|_{\vec{\underline{\nu}}^{(k)}}(\vec{Y}).$$

Part 1: Show that, for all $i \in \mathbb{N}$, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\underline{\nu}}^{(i)}$.

Base case. $\vec{\kappa}^{(0)} = \vec{\underline{0}} \sqsubseteq \vec{f}(\vec{\underline{0}}) = \vec{\underline{\nu}}^{(0)}$.

Induction step. Assume that $\vec{\kappa}^{(k)} \sqsubseteq \underline{\vec{\nu}}^{(k)}$. Then $\vec{\kappa}^{(k+1)} = \vec{f}(\vec{\kappa}^{(k)}) \sqsubseteq \vec{f}(\underline{\vec{\nu}}^{(k)}) \sqsubseteq \vec{Y}_k^\star = \underline{\vec{\nu}}^{(k+1)}$.

Therefore, for all $i \in \mathbb{N}$, $\vec{\kappa}^{(i)} \sqsubseteq \vec{\nu}^{(i)}$.

Part 2: Show that, for all $i \in \mathbb{N}$, $\underline{\vec{\nu}}^{(i)} \sqsubseteq \vec{X}^\star$.

Base case. $\underline{\vec{\nu}}^{(0)} = \vec{f}(\underline{\vec{0}}) \sqsubseteq \bigoplus_{i=0}^{\infty} \vec{f}^{\,i}(\underline{\vec{0}}) = \vec{X}^\star$.

Induction step. Assume that $\underline{\vec{\nu}}^{(k)} \sqsubseteq \vec{X}^\star$. By the induction hypothesis and Lem. A.4, $\vec{Y}_k^\star \sqsubseteq \vec{X}^\star$. Consequently, $\underline{\vec{\nu}}^{(k+1)} = \vec{Y}_k^\star \sqsubseteq \vec{X}^\star$.

Therefore, for all $i \in \mathbb{N}$, $\underline{\vec{\nu}}^{(i)} \sqsubseteq \vec{X}^\star$. $\square$

A.2 NPA with Operators $\oplus$, $\otimes$, *, and Project, and Iterates Computed Via $\vec{Y} = \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\vec{\nu}^{(i)}}(\vec{Y})$

This section addresses NPA with the operators $\oplus$, $\otimes$, *, and Project, but where Eqn. (5) in Alg. A.1 is changed to Eqn. (6):

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

The advantage of using Eqn. (6) is that it eliminates the need to compute $\vec{f}(\vec{\nu}^{(i)})$ explicitly on each Newton round. In this section, we show that the combinations Eqns. (4) and (5) and Eqns. (4) and (6) produce the same set of iterates $\vec{\nu}^{(0)}, \vec{\nu}^{(1)}, \ldots, \vec{\nu}^{(i)}, \ldots$. (See the proof of Thm. 8.6 at the end of this subsection.)

LEMMA A.7. *For all $\vec{w}$ such that $\underline{\vec{\nu}} \sqsupseteq \vec{w}$, $f_i(\underline{\vec{0}}) \oplus \mathcal{D}f_i|_{\underline{\vec{\nu}}}(\vec{w}) \sqsupseteq f_i(\vec{w})$.*

Proof. By structural induction on the expression for f_i.

$$\begin{aligned}
&\text{if } f_i = s \in \mathcal{S}: && \text{LHS} = s \oplus \underline{0} = s = \text{RHS} \\
&\text{if } f_i = X_j: && \text{LHS} = \underline{0} \oplus \bigoplus_{k=1}^{n} \mathcal{D}_{X_k} X_j|_{\underline{\vec{\nu}}}(\vec{w}) \\
& && \phantom{\text{LHS}} = w_j \\
& && \phantom{\text{LHS}} = \text{RHS} \\
&\text{if } f_i = \bigoplus_{k\in K} g_k: && \text{LHS} = \bigoplus_{k\in K} g_k(\underline{\vec{0}}) \oplus \bigoplus_{k\in K} \mathcal{D}g_k|_{\underline{\vec{\nu}}}(\vec{w}) \\
& && \phantom{\text{LHS}} = \bigoplus_{k\in K} \Big(g_k(\underline{\vec{0}}) \oplus \mathcal{D}g_k|_{\underline{\vec{\nu}}}(\vec{w}) \Big) \\
& && \phantom{\text{LHS}} \sqsupseteq \bigoplus_{k\in K} g_k(\vec{w}) \\
& && \phantom{\text{LHS}} = \text{RHS} \\
&\text{if } f_i = g \otimes h: && \text{LHS} = (g\otimes h)(\underline{\vec{0}}) \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\, g(\underline{\vec{\nu}}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \\
& && \phantom{\text{LHS}} = \Big(g(\underline{\vec{0}}) \otimes h(\underline{\vec{0}}) \Big) \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\, g(\underline{\vec{\nu}}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \\
& && \phantom{\text{LHS}} = \begin{pmatrix} \Big(g(\underline{\vec{0}}) \otimes h(\underline{\vec{0}}) \Big) \\ \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{0}}) \\ \oplus\, g(\underline{\vec{0}}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \\ \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\, g(\underline{\vec{\nu}}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \end{pmatrix} \\
& && \phantom{\text{LHS}} = \begin{pmatrix} \Big(g(\underline{\vec{0}}) \otimes h(\underline{\vec{0}}) \Big) \oplus \Big(\mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{0}}) \Big) \\ \oplus \Big(g(\underline{\vec{0}}) \otimes h(\underline{\vec{0}}) \Big) \oplus \Big(g(\underline{\vec{0}}) \oplus \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \Big) \\ \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\, g(\underline{\vec{\nu}}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \end{pmatrix} \\
& && \phantom{\text{LHS}} = \begin{pmatrix} \Big(g(\underline{\vec{0}}) \oplus \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \Big) \otimes h(\underline{\vec{0}}) \\ \oplus\, g(\underline{\vec{0}}) \otimes \Big(h(\underline{\vec{0}}) \oplus \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \Big) \\ \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\, g(\underline{\vec{\nu}}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \end{pmatrix} \\
& && \phantom{\text{LHS}} \sqsupseteq \begin{pmatrix} g(\vec{w}) \otimes h(\underline{\vec{0}}) \\ \oplus\, g(\underline{\vec{0}}) \otimes h(\vec{w}) \\ \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\underline{\vec{\nu}}) \\ \oplus\, g(\underline{\vec{\nu}}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \end{pmatrix} \\
& && \phantom{\text{LHS}} \sqsupseteq \begin{pmatrix} g(\vec{w}) \otimes h(\underline{\vec{0}}) \\ \oplus\, g(\underline{\vec{0}}) \otimes h(\vec{w}) \\ \oplus \begin{pmatrix} \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\vec{w}) \\ \oplus\, g(\vec{w}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \end{pmatrix} \end{pmatrix} \\
& && \phantom{\text{LHS}} = \begin{pmatrix} \Big(g(\vec{w}) \otimes h(\underline{\vec{0}}) \Big) \oplus \big(g(\vec{w}) \otimes \mathcal{D}h|_{\underline{\vec{\nu}}}(\vec{w}) \big) \\ \oplus \Big(g(\underline{\vec{0}}) \otimes h(\vec{w}) \Big) \oplus \big(\mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w}) \otimes h(\vec{w}) \big) \end{pmatrix}
\end{aligned}$$

$$
\begin{aligned}
&= \begin{pmatrix} g(\vec{w}) \otimes \Big(h(\underline{\vec{0}}) \oplus \mathcal{D}h|_{\underline{\vec{v}}}(\vec{w})\Big) \\ \oplus \Big(g(\underline{\vec{0}}) \oplus \mathcal{D}g|_{\underline{\vec{v}}}(\vec{w})\Big) \otimes h(\vec{w}) \end{pmatrix} \\
&\sqsupseteq g(\vec{w}) \otimes h(\vec{w}) \\
&= \text{RHS}
\end{aligned}
$$

$$
\begin{aligned}
\text{if } f_i = g^* : \qquad \text{LHS} &= g^*(\underline{\vec{0}}) \oplus \mathcal{D}g^*|_{\underline{\vec{v}}}(\vec{w}) \\
&= \left(\bigoplus_{j=0}^{\infty} g^j\right)(\underline{\vec{0}}) \oplus \mathcal{D}\left(\bigoplus_{j=0}^{\infty} g^j\right)|_{\underline{\vec{v}}}(\vec{w}) \\
&= \left(\bigoplus_{j=0}^{\infty} g^j(\underline{\vec{0}})\right) \oplus \bigoplus_{j=0}^{\infty} \left(\mathcal{D}g^j|_{\underline{\vec{v}}}(\vec{w})\right) \\
&= \bigoplus_{j=0}^{\infty} \Big(g^j(\underline{\vec{0}}) \oplus \mathcal{D}g^j|_{\underline{\vec{v}}}(\vec{w})\Big)
\end{aligned}
$$

We now need to show by induction that for all j,

$$
g^j(\underline{\vec{0}}) \oplus \mathcal{D}g^j|_{\underline{\vec{v}}}(\vec{w}) \sqsupseteq g^j(\vec{w}). \tag{79}
$$

Note that Eqn. (79) does not follow immediately from the structural induction because g^j is not a syntactic constituent of f_i $(= g^*)$.

Base case, $j = 0$.

$$
\text{LHS} = g^0(\underline{\vec{0}}) \oplus \mathcal{D}g^0|_{\underline{\vec{v}}}(\vec{w}) = \underline{1} \oplus \underline{0} = \underline{1} = g^0(\vec{w}) = \text{RHS}.
$$

Induction step. Assume that $g^k(\underline{\vec{0}}) \oplus \mathcal{D}g^k|_{\underline{\vec{v}}}(\vec{w}) \sqsupseteq g^k(\vec{w})$. We need to show that $g^{k+1}(\underline{\vec{0}}) \oplus \mathcal{D}g^{k+1}|_{\underline{\vec{v}}}(\vec{w}) \sqsupseteq g^{k+1}(\vec{w})$. However, $g^{k+1} = g^k \otimes g$, so the same argument used in the case for "$f_i = g \otimes h$" in the structural induction can be reused:

$$
\begin{aligned}
\text{LHS} &= (g^k \otimes g)(\underline{\vec{0}}) \oplus \begin{pmatrix} \mathcal{D}g^k|_{\underline{\vec{v}}}(\vec{w}) \otimes g(\underline{\vec{v}}) \\ \oplus\, g^k(\underline{\vec{v}}) \otimes \mathcal{D}g|_{\underline{\vec{v}}}(\vec{w}) \end{pmatrix} \\
&= \Big(g^k(\underline{\vec{0}}) \otimes g(\underline{\vec{0}})\Big) \oplus \begin{pmatrix} \mathcal{D}g^k|_{\underline{\vec{v}}}(\vec{w}) \otimes g(\underline{\vec{v}}) \\ \oplus\, g^k(\underline{\vec{v}}) \otimes \mathcal{D}g|_{\underline{\vec{v}}}(\vec{w}) \end{pmatrix} \\
&= \ldots \textit{intermediate steps omitted} \ldots \\
&= \begin{pmatrix} g^k(\vec{w}) \otimes \Big(g(\underline{\vec{0}}) \oplus \mathcal{D}g|_{\underline{\vec{v}}}(\vec{w})\Big) \\ \oplus \Big(g^k(\underline{\vec{0}}) \oplus \mathcal{D}g^k|_{\underline{\vec{v}}}(\vec{w})\Big) \otimes g(\vec{w}) \end{pmatrix} \\
&\sqsupseteq g^k(\vec{w}) \otimes g(\vec{w}) \\
&= \text{RHS}
\end{aligned}
$$

Therefore, for all $j \in \mathbb{N}$, $g^j(\underline{\vec{0}}) \oplus \mathcal{D}g^j|_{\underline{\vec{v}}}(\vec{w}) \sqsupseteq g^j(\vec{w})$.

The case for "if $f_i = g^*$" in the structural induction continues as follow:

$$
\begin{aligned}
\bigoplus_{j=0}^{\infty} \Big(g^j(\underline{\vec{0}}) \oplus \mathcal{D}g^j|_{\underline{\vec{v}}}(\vec{w})\Big) &\sqsupseteq \bigoplus_{j=0}^{\infty} g^j(\vec{w}) \\
&= \left(\bigoplus_{j=0}^{\infty} g^j\right)(\vec{w}) \\
&= g^*(\vec{w}) \\
&= \text{RHS}
\end{aligned}
$$

$$\text{if } f_i = \text{Project}(g): \quad \begin{aligned} \text{LHS} &= \text{Project}(g(\underline{\vec{0}})) \oplus \text{Project}(\mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w})) \\ &= \text{Project}(g(\underline{\vec{0}}) \oplus \mathcal{D}g|_{\underline{\vec{\nu}}}(\vec{w})) \\ &\sqsupseteq \text{Project}(g(\vec{w})) \\ &= \text{RHS} \end{aligned}$$

□

Corollary A.8. *For all $\vec{w}$ such that $\underline{\vec{\nu}} \sqsupseteq \vec{w}$, $\vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{w}) \sqsupseteq \vec{f}(\vec{w})$.*

Proof. Immediate from the definition of $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{w})$ (Defn. 2.5) and Lem. A.7. □

Remark A.9. *Cor. A.8 is almost, but not quite the dual of Cor. A.3:*

$$\begin{array}{lllcl} \textit{Cor. A.3:} & \underline{\vec{\nu}} \sqsubseteq \vec{w} \textit{ implies that} & \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{w}) & \sqsubseteq & \vec{f}(\vec{w}) \\ \textit{Cor. A.8:} & \underline{\vec{\nu}} \sqsupseteq \vec{w} \textit{ implies that} & \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{w}) & \sqsupseteq & \vec{f}(\vec{w}) \end{array}$$

The reason why Lem. A.7 is stated in terms of $\mathcal{D}f_i|_{\underline{\vec{\nu}}}(\vec{w})$ rather than $\mathcal{D}_{X_j}f_i|_{\underline{\vec{\nu}}}(\vec{w})$, as in Lem. A.2, is because in the "if $f_i = X_j$" case of Lem. A.7, we need access to **all** *of the $\mathcal{D}_{X_k}X_j|_{\underline{\vec{\nu}}}(\vec{w})$ terms, so that their combine yields w_j.*

Also, the reason we have "$\vec{f}(\underline{\vec{0}}) \oplus \ldots$" in Lem. A.7 is so that the "if $f_i = s \in \mathcal{S}$" case works out correctly: the $\vec{f}(\underline{\vec{0}})$ term supplies the value s on the left-hand side, which is needed to match the $f_i(w) = s$ that appears on the right-hand side.

Lemma A.10. *Suppose that $\vec{a} \sqsubseteq \vec{b}$, $\vec{Y}^\star_{\vec{a}}$ is the least solution to $\vec{Y} = \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\vec{a}}(\vec{Y})$, and $\vec{Y}^\star_{\vec{b}}$ is the least solution to $\vec{Y} = \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\vec{b}}(\vec{Y})$. Then $\vec{Y}^\star_{\vec{a}} \sqsubseteq \vec{Y}^\star_{\vec{b}}$.*

Proof. Let $\vec{G}_{\vec{a}}$ and $\vec{G}_{\vec{b}}$ denote the functions $\lambda\vec{y}.\vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\vec{a}}(\vec{y})$ and $\lambda\vec{y}.\vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\vec{b}}(\vec{y})$, respectively. By the monotonicity of $\mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$, $\vec{G}_{\vec{a}} \sqsubseteq \vec{G}_{\vec{b}}$. By an easy induction, for all $j \in \mathbb{N}$, $\vec{G}^j_{\vec{a}}(\underline{\vec{0}}) \sqsubseteq \vec{G}^j_{\vec{b}}(\underline{\vec{0}})$, and hence

$$\vec{Y}^\star_{\vec{a}} = \bigoplus_{j=0}^{\infty} \vec{G}^j_{\vec{a}}(\underline{\vec{0}}) \sqsubseteq \bigoplus_{j=0}^{\infty} \vec{G}^j_{\vec{b}}(\underline{\vec{0}}) = \vec{Y}^\star_{\vec{b}}.$$

□

Corollary A.11. *The elements $\{\underline{\vec{\nu}}^{(k)}\}$ of the sequence obtained using Eqns. (4) and (6) have the property that, for all $i \in \mathbb{N}$, $\underline{\vec{\nu}}^{(i)} \sqsubseteq \underline{\vec{\nu}}^{(i+1)}$.*

Proof. *Base case.* From the form of the equation $\vec{Y} = \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}}(\vec{Y})$, we know that $\vec{Y}^\star \sqsupseteq \vec{f}(\underline{\vec{0}})$, and hence for all $i \in \mathbb{N}$, $\underline{\vec{\nu}}^{(i)} \sqsupseteq \vec{f}(\underline{\vec{0}})$. In particular,

$$\underline{\vec{\nu}}^{(1)} \sqsupseteq \vec{f}(\underline{\vec{0}}) = \underline{\vec{\nu}}^{(0)}.$$

Induction step. Assume that $\underline{\vec{\nu}}^{(k-1)} \sqsubseteq \underline{\vec{\nu}}^{(k)}$; show that $\underline{\vec{\nu}}^{(k)} \sqsubseteq \underline{\vec{\nu}}^{(k+1)}$.
By Lem. A.10,

$$\underline{\vec{\nu}}^{(k)} = \vec{Y}^\star_{\underline{\vec{\nu}}^{(k-1)}} \sqsubseteq \vec{Y}^\star_{\underline{\vec{\nu}}^{(k)}} = \underline{\vec{\nu}}^{(k+1)}.$$

Therefore, for all $i \in \mathbb{N}$, $\underline{\vec{\nu}}^{(i)} \sqsubseteq \underline{\vec{\nu}}^{(i+1)}$. □

The next theorem shows that it does not matter whether we compute the Newton iterates using Eqn. (5) or Eqn. (6).

Theorem 8.6. *Let $\mathcal{E} : \vec{X} = \vec{f}(\vec{X})$ be an equation system whose right-hand-side terms can contain both regular operators and occurrences of the operator Project. Let the sequence $N \stackrel{\text{def}}{=} [\underline{\vec{\nu}}^{(i)} \mid i \in \mathbb{N}]$ be defined by Eqns. (4) and (5), and the sequence $M \stackrel{\text{def}}{=} [\underline{\vec{\mu}}^{(i)} \mid i \in \mathbb{N}]$ be defined by Eqns. (4) and (6). Then $M = N$.*

Proof. For all $i \in \mathbb{N}$, $\underline{\vec{\mu}}^{(i)} \sqsubseteq \underline{\vec{\mu}}^{(i+1)}$ and $\underline{\vec{\nu}}^{(i)} \sqsubseteq \underline{\vec{\nu}}^{(i+1)}$, by Cors. A.6 and A.11, respectively.

We now show by induction that, for all $i \in \mathbb{N}$, $\underline{\vec{\mu}}^{(i)} = \underline{\vec{\nu}}^{(i)}$.

Base case, $i = 0$. $\underline{\vec{\mu}}^{(0)} = \vec{f}(\underline{0}) = \underline{\vec{\nu}}^{(0)}$.

Induction step. Assume that $\underline{\vec{\mu}}^{(i)} = \underline{\vec{\nu}}^{(i)}$; show that $\underline{\vec{\mu}}^{(i+1)} = \underline{\vec{\nu}}^{(i+1)}$.

$\vec{U}^\star \stackrel{\text{def}}{=} \underline{\vec{\mu}}^{(i+1)}$ is the least solution of $\vec{U} = \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\vec{U})$, whereas $\underline{\vec{\nu}}^{(i+1)}$ is the least solution of $\vec{Y} = \vec{f}(\underline{\vec{\nu}}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}^{(i)}}(\vec{Y})$. Let $\vec{G}_i$ denote the function $\lambda\vec{u}.\vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\vec{u})$ and $\vec{F}_i$ denote the function $\lambda\vec{y}.\vec{f}(\underline{\vec{\nu}}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}^{(i)}}(\vec{y})$. Because $\vec{f}(\underline{\vec{0}}) \sqsubseteq \vec{f}(\underline{\vec{\nu}})$ and $\underline{\vec{\mu}}^{(i)} = \underline{\vec{\nu}}^{(i)}$, $\vec{G}_i \sqsubseteq \vec{F}_i$. By an easy induction, for all $j \in \mathbb{N}$ $\vec{G}_i^j(\underline{\vec{0}}) \sqsubseteq \vec{F}_i^j(\underline{\vec{0}})$, and hence

$$\underline{\vec{\mu}}^{(i+1)} = \bigoplus_{j=0}^{\infty} \vec{G}_i^j(\underline{\vec{0}}) \sqsubseteq \bigoplus_{j=0}^{\infty} \vec{F}_i^j(\underline{\vec{0}}) = \underline{\vec{\nu}}^{(i+1)}. \tag{80}$$

We now wish to show that $\underline{\vec{\mu}}^{(i+1)}$ is a fixed-point of $\vec{F}_i$.

$$\begin{aligned}
\vec{F}_i(\underline{\vec{\mu}}^{(i+1)}) &= \vec{F}_i(\vec{U}^\star) & \\
&= \vec{f}(\underline{\vec{\nu}}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\nu}}^{(i)}}(\vec{U}^\star) & \\
&= \vec{f}(\underline{\vec{\mu}}^{(i)}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\vec{U}^\star) & \text{because } \underline{\vec{\nu}}^{(i)} = \underline{\vec{\mu}}^{(i)} \\
&= \vec{f}(\underline{\vec{\mu}}^{(i)}) \oplus \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\vec{U}^\star) & \vec{f}(\underline{\vec{0}}) \sqsubseteq \vec{f}(\underline{\vec{\mu}}^{(i)}) \\
&= \vec{f}(\underline{\vec{\mu}}^{(i)}) \oplus \vec{U}^\star & \vec{U}^\star = \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\vec{U}^\star)
\end{aligned}$$

However,

$$\begin{aligned}
\vec{f}(\underline{\vec{\mu}}^{(i)}) &\sqsubseteq \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\underline{\vec{\mu}}^{(i)}) & \text{by Cor. A.8, using } \underline{\vec{\mu}}^{(i)} \sqsubseteq \underline{\vec{\mu}}^{(i)} \\
&\sqsubseteq \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\underline{\vec{\mu}}^{(i+1)}) & \text{by Cor. A.11 and monotonicity} \\
&= \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\vec{U}^\star) & \underline{\vec{\mu}}^{(i+1)} = \vec{U}^\star \\
&= \vec{U}^\star & \vec{U}^\star = \vec{f}(\underline{\vec{0}}) \oplus \mathcal{D}\vec{f}|_{\underline{\vec{\mu}}^{(i)}}(\vec{U}^\star)
\end{aligned}$$

Putting the above two derivations together, we have,

$$\vec{F}_i(\underline{\vec{\mu}}^{(i+1)}) = \vec{f}(\underline{\vec{\mu}}^{(i)}) \oplus \vec{U}^\star = \vec{U}^\star = \underline{\vec{\mu}}^{(i+1)} \tag{81}$$

Because $\underline{\vec{\nu}}^{(i+1)}$ is the ***least*** fixed-point of $\vec{F}_i$, Eqns. (80) and (81) imply that $\underline{\vec{\mu}}^{(i+1)} = \underline{\vec{\nu}}^{(i+1)}$.

Thus, for all $i \in \mathbb{N}$, $\underline{\vec{\mu}}^{(i)} = \underline{\vec{\nu}}^{(i)}$, and hence $M = N$. □

A.3 Correctness of Alg. 8.12 for Solving a Linear Projection-Equation System

This section proves the correctness of Alg. 8.12, our method for solving a linear projection-equation system over a predicate-abstraction domain PA. As with the proof of Alg. 4.4 (§4.5), the proof in this section is based on grammar flow analysis; however, the details are different in certain places.

We first prove three lemmas that we will need shortly.

LEMMA A.12. *For all $b, c \in PA$ and $T \in PA_{\mathcal{T}}$,*

$$\lightning^{(t,\cdot)}(T \otimes_{\mathcal{T}} (b^t \odot c)) = b \otimes \lightning^{(t,\cdot)}(T) \otimes c.$$

PROOF. Let the vocabularies of b, c, and T be $b(p,p')$, $c(q,q')$, and $T(p',q,p,q')$, respectively.

$$\begin{aligned}
\text{RHS} &= b \otimes \{(p,q) \mid T(r,r,p,q)\} \otimes c \\
&= \{(p,s) \mid b(p,s)\} \otimes \{(s,t) \mid T(r,r,s,t)\} \otimes \{(t,q') \mid c(t,q')\} \\
&= \{(p,q') \mid b(p,s) \wedge T(r,r,s,t) \wedge c(t,q')\}
\end{aligned}$$

$$\begin{aligned}
\text{LHS} &= \lightning^{(t,\cdot)} \begin{pmatrix} \{(\bar{p}',\bar{q},\bar{p},\bar{q}') \mid T(\bar{p}',\bar{q},\bar{p},\bar{q}')\} \\ \otimes_{\mathcal{T}} \ \{(p',q,p,q') \mid b(p,p') \wedge c(q,q')\} \end{pmatrix} \\
&= \lightning^{(t,\cdot)} \left(\left\{ \begin{array}{l} (\bar{p}',\bar{q},p,q') \mid T(\bar{p}',\bar{q},\bar{p},\bar{q}') \wedge b(p,p') \wedge c(q,q') \\ \qquad\qquad\qquad\qquad \wedge\ \bar{p} = p' \wedge \bar{q}' = q \end{array} \right\} \right) \\
&= \{(p,q') \mid b(p,p') \wedge T(\bar{p}',\bar{q},\bar{p},\bar{q}') \wedge c(q,q') \wedge \bar{p} = p' \wedge \bar{q}' = q \wedge \bar{p}' = \bar{q}\} \\
&= \{(p,q') \mid b(p,s) \wedge T(r,r,s,t) \wedge c(t,q')\} \\
&= \text{RHS}
\end{aligned}$$

□

The next two lemmas, Lemmas A.14 and A.15, concern GFA problems related by production functions that reflect the structure of the right-hand sides of Eqns. (68) and (74). More precisely, we consider Eqn. (68) rewritten to the following form

$$Y_k = \text{Project}\left(a_k^{(i)}\right) \oplus \bigoplus_j \bigoplus_l \text{Project}\left(b_{j,k,l}^{(i)} \otimes Y_j \otimes c_{j,k,l}^{(i)}\right) \tag{82}$$

so that the production functions of a GFA problems can be read out immediately: each production function has one the two forms shown in column two of the table in Fig. 14. The related GFA problems, for grammars G and $G_{\mathcal{T}}$, are defined by columns two and three of the table in Fig. 14.

OBSERVATION A.13.

—*By Defn. 2.1(3) and Lem. 8.3(2) (see App. B), a production function of the kind that appears in column two of Fig. 14, namely, $\lambda Y.Project(b \otimes Y \otimes c)$, distributes over infinite combines ($\oplus$).*

—*By Defn. 2.1(3), a production function of the kind that appears in column three of Fig. 14, namely, $\lambda Z.Z \otimes_{\mathcal{T}} Project_{\mathcal{T}}(b^t \odot c)$, distributes over infinite tensored combines ($\oplus_{\mathcal{T}}$).*[17]

[17]In $\lambda Z.Z \otimes_{\mathcal{T}}$ Project$_{\mathcal{T}}(b^t \odot c)$, Project$_{\mathcal{T}}$ is applied to $(b^t \odot c)$, which is an $\mathcal{S}_{\mathcal{T}}$ constant, and thus although Project$_{\mathcal{T}}$ does distribute over infinite tensored combines (see Lem. 8.8(4) in App. C), this property does not play any direct role in the correctness argument.

Production	Production Function	
	G	$G_{\mathcal{T}}$
$X_0 \to g_0()$	$[\![g_0]\!] \overset{\text{def}}{=} \text{Project}(a)$	$[\![g_0]\!]_{\mathcal{T}} \overset{\text{def}}{=} \text{Project}_{\mathcal{T}}(\underline{1}^t \odot a)$
$X_0 \to g_1(X_1)$	$[\![g_1]\!] \overset{\text{def}}{=} \lambda Y.\text{Project}(b \otimes Y \otimes c)$	$[\![g_1]\!]_{\mathcal{T}} \overset{\text{def}}{=} \lambda Z.Z \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c)$

Fig. 14. Linear GFA problems used in the proof of Lem. A.15.

Consequently, the production functions in Fig. 14 meet the infinite-distributivity requirement of Thm. 4.7, and hence Thm. 4.7 applies.

LEMMA A.14. *For every $\alpha \in L_{G_{\mathcal{T}}}(X)$,*

$$val_{G_{\mathcal{T}}}(\alpha) = Project_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\alpha)). \tag{83}$$

PROOF. By induction on the height h of the derivation tree of α.

Base case, $h = 0$. The derivation tree has the form $g()$. Let $[\![g]\!]$ be Project(a), and thus $[\![g]\!]_{\mathcal{T}}$ is Project$_{\mathcal{T}}(\underline{1}^t \odot a)$. Then

$$\begin{array}{rcll} val_{G_{\mathcal{T}}}(\alpha) & = & val_{G_{\mathcal{T}}}(g()) & \\ & = & [\![g]\!]_{\mathcal{T}} & \\ & = & \text{Project}_{\mathcal{T}}(\underline{1}^t \odot a) & (\dagger) \\ & = & \text{Project}_{\mathcal{T}}(\text{Project}_{\mathcal{T}}(\underline{1}^t \odot a)) & \text{by Eqn. (71)} \\ & = & \text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\alpha)) & \text{by line } (\dagger) \end{array}$$

Induction step. Assume that Eqn. (83) holds for all derivation trees of height h. Let $\alpha = g(\beta)$, where β is a string whose derivation tree has height h, so that $val_{G_{\mathcal{T}}}(\beta) = \text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta))$. Let $[\![g]\!]$ be $\lambda Y.\text{Project}(b \otimes Y \otimes c)$, and thus $[\![g]\!]_{\mathcal{T}}$ is $\lambda Z.Z \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c)$.

$$\begin{array}{rll} val_{G_{\mathcal{T}}}(\alpha) & = val_{G_{\mathcal{T}}}(g(\beta)) & \\ & = [\![g]\!]_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta)) & \\ & = (\lambda Z.Z \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c))(val_{G_{\mathcal{T}}}(\beta)) & \\ & = val_{G_{\mathcal{T}}}(\beta) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c) & (\ddagger) \\ & = \text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta)) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c) & \text{by the ind. hyp.} \\ & = \text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c)) & \text{by Eqn. (72)} \\ & = \text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\alpha)) & \text{by line } (\ddagger) \end{array}$$

□

The next lemma, Lem. A.15, is similar to Lem. 4.14; however, because of the occurrences of Project and Project$_{\mathcal{T}}$ in Eqns. (82) and (74), respectively, we can no longer assume that Obs. 4.11 holds—i.e., it is no longer true that $val_{G_{\mathcal{T}}}(\alpha)$ can always be written in the form $m^t \odot n$ by repeated application of Eqn. (32).

LEMMA A.15. *Let G and $G_{\mathcal{T}}$ be linear GFA problems over PA and $PA_{\mathcal{T}}$, respectively, with (i) the same set of productions, and (ii) production functions related according to the table given in Fig. 14. Then for every $\alpha \in L_{G_{\mathcal{T}}}(X)$,*

$$\lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\alpha)) = val_G(\alpha). \tag{84}$$

PROOF. By induction on the height h of the derivation tree of α.

Base case, $h = 0$. The derivation tree has the form $g()$. Let $[\![g]\!]$ be $\text{Project}(a)$, and thus $[\![g]\!]_{\mathcal{T}}$ is $\text{Project}_{\mathcal{T}}(\underline{1}^t \odot a)$. Then

$$\begin{aligned}
\lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\alpha)) &= \lightning^{(t,\cdot)}([\![g]\!]_{\mathcal{T}}) \\
&= \lightning^{(t,\cdot)}(\text{Project}_{\mathcal{T}}(\underline{1}^t \odot a)) \\
&= \text{Project}(\lightning^{(t,\cdot)}(\underline{1}^t \odot a)) \qquad \text{by Lem. C.1} \\
&= \text{Project}(\underline{1} \otimes a) \\
&= \text{Project}(a) \\
&= [\![g]\!] \\
&= val_G(g()) \\
&= val_G(\alpha).
\end{aligned}$$

Induction step. Assume that Eqn. (84) holds for all derivation trees of height h. Let $\alpha = g(\beta)$, where β is a string whose derivation tree has height h, so that $\lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\beta)) = val_G(\beta)$. Let $[\![g]\!]$ be $\lambda Y.\text{Project}(b \otimes Y \otimes c)$, and thus $[\![g]\!]_{\mathcal{T}}$ is $\lambda Z.Z \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c)$.

$$\begin{aligned}
&\lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\alpha)) \\
&\quad= \lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(g(\beta))) \\
&\quad= \lightning^{(t,\cdot)}([\![g]\!]_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta))) \\
&\quad= \lightning^{(t,\cdot)}((\lambda Z.Z \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c))(val_{G_{\mathcal{T}}}(\beta))) \\
&\quad= \lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\beta) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c)) \\
&\quad= \lightning^{(t,\cdot)}(\text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta)) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b^t \odot c)) && \text{by Lem. A.14} \\
&\quad= \lightning^{(t,\cdot)}(\text{Project}_{\mathcal{T}}(\text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta)) \otimes_{\mathcal{T}}(b^t \odot c))) && \text{by Eqn. (73)} \\
&\quad= \lightning^{(t,\cdot)}(\text{Project}_{\mathcal{T}}(val_{G_{\mathcal{T}}}(\beta) \otimes_{\mathcal{T}}(b^t \odot c))) && \text{by Lem. A.14} \\
&\quad= \text{Project}(\lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\beta) \otimes_{\mathcal{T}}(b^t \odot c))) && \text{by Lem. C.1} \\
&\quad= \text{Project}(b \otimes \lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\beta)) \otimes c))) && \text{by Lem. A.12} \\
&\quad= \text{Project}(b \otimes val_G(\beta) \otimes c))) && \text{by the ind. hyp.} \\
&\quad= (\lambda Y.\text{Project}(b \otimes Y \otimes c))(val_G(\beta)) \\
&\quad= [\![g]\!](val_G(\beta)) \\
&\quad= val_G(g(\beta)) \\
&\quad= val_G(\alpha).
\end{aligned}$$

□

For the linear GFA problems G and $G_{\mathcal{T}}$ generated by a linear projection-equation system $\mathcal{L}$ over PA, the proof of Cor. 4.15 carries over essentially verbatim. The only change is to use Lem. A.15 instead of Lem. 4.14 to justify the step from the second line to the third line in the proof.

COROLLARY A.16. *For each nonterminal $X \in G$,*

$$\lightning^{(t,\cdot)}(m_{G_{\mathcal{T}}}(X)) = m_G(X). \tag{85}$$

PROOF.

$$\begin{aligned}
\lightning^{(t,\cdot)}(m_{G_{\mathcal{T}}}(X)) &= \lightning^{(t,\cdot)}\Big(\bigoplus_{\alpha \in L_{G_{\mathcal{T}}}(X)} val_{G_{\mathcal{T}}}(\alpha)\Big) \\
&= \bigoplus_{\alpha \in L_{G_{\mathcal{T}}}(X)} \lightning^{(t,\cdot)}(val_{G_{\mathcal{T}}}(\alpha)) \\
&= \bigoplus_{\alpha \in L_G(X)} val_G(\alpha) && \text{by Lem. A.15} \\
&= m_G(X).
\end{aligned}$$

□

Theorem 8.13. *Given a linear projection-equation system $\mathcal{L}$ over predicate-abstraction domain PA, Alg. 8.12 finds the least solution of $\mathcal{L}$.*

PROOF. We are given a linear projection-equation system $\mathcal{L}$ over predicate-abstraction domain PA, which can be formulated equivalently as a linear GFA problem G over PA, in which each of the production functions has one the two forms shown in column two of the table in Fig. 14. We wish to show that Steps 1–4 of Alg. 8.12 compute the least-fixed-point solution $n_G(X)$, for each nonterminal $X \in G$.

From equation system $\mathcal{L}$, Step 1 creates a tensored equation system $\mathcal{L}_{\mathcal{T}}$. When $\mathcal{L}_{\mathcal{T}}$ is formulated as a GFA problem, it corresponds to the left-linear GFA problem $G_{\mathcal{T}}$ over PA$_{\mathcal{T}}$ obtained by applying the transformation given by the table in Fig. 14.

Steps 2 and 3 use Tarjan's path-expression algorithm [Tarjan 1981b] to create an appropriate regular expression for each variable Z_i in $\mathcal{L}_{\mathcal{T}}$, which are then evaluated. Consequently, by [Tarjan 1981b, Thm. 5], these steps find the combine-over-all-derivations solution to $\mathcal{L}_{\mathcal{T}}$, and hence compute $m_{G_{\mathcal{T}}}(X)$ for each nonterminal $X \in G_{\mathcal{T}}$. Finally, Step 4 computes $\lightning^{(t,\cdot)}(m_{G_{\mathcal{T}}}(X))$ for $X \in G_{\mathcal{T}}$. However, by the results presented above, we have

$$\begin{aligned}
\lightning^{(t,\cdot)}(m_{G_{\mathcal{T}}}(X)) &= m_G(X) && \text{by Cor. A.16} \\
&= n_G(X) && \text{by Obs. A.13 and Thm. 4.7}
\end{aligned}$$

□

B. PROPERTIES OF M AND PROJECT

Lemma 8.3. *The M operation defined in Eqn. (65) is an acceptable merge function for $\mathcal{S}$, in the sense of Defn. 8.1.*

PROOF. (1) (M is $\underline{0}$-strict.)

(a) For all $a \in \mathcal{S}$,

$$\begin{aligned}
M(a, \underline{0}) &= a \otimes \text{Project}(\underline{0}) \\
&= a \otimes ((\exists L, L' : \text{false}) \wedge (L = L')) \\
&= a \otimes (\text{false} \wedge (L = L')) \\
&= a \otimes \text{false} \\
&= a \otimes \underline{0} \\
&= \underline{0}
\end{aligned}$$

(b) For all $b \in \mathcal{S}$, $M(\underline{0}, b) = \underline{0} \otimes \text{Project}(b) = \underline{0}$.

(2) (M is infinitely distributive in both argument positions.)

(a) For all $a_i, b \in \mathcal{S}$ and $I \subseteq \mathbb{N}$,

$$\begin{aligned} M\left(\bigoplus_{i\in I} a_i, b\right) &= \left(\bigoplus_{i\in I} a_i\right) \otimes \text{Project}(b) \\ &= \bigoplus_{i\in I} (a_i \otimes \text{Project}(b)) \\ &= \bigoplus_{i\in I} M(a_i, b) \end{aligned}$$

(b) For all $a, b_i \in \mathcal{S}$ and $I \subseteq \mathbb{N}$,

$$M\left(a, \bigoplus_{i\in I} b_i\right) = a \otimes \text{Project}\left(\bigoplus_{i\in I} b_i\right).$$

The operation $\text{Project}(R(G, L, G', L'))$ defined in Eqn. (65) constructs the answer relation by considering each tuple in R independently; i.e.,

$$\text{Project}(R(G, L, G', L')) = \{(g, k, g', k) \mid R(g, l, g', l')\}.$$

Consequently, Project distributes over set-union—the $\oplus$ operation of $\mathcal{S}$—including infinite set-unions. Therefore,

$$\begin{aligned} a \otimes \text{Project}\left(\bigoplus_{i\in I} b_i\right) &= a \otimes \bigoplus_{i\in I} \text{Project}(b_i) \\ &= \bigoplus_{i\in I} (a \otimes \text{Project}(b_i)) \\ &= \bigoplus_{i\in I} M(a, b_i) \end{aligned}$$

(3) (M has the path-extension property.) For all $a, b, c \in \mathcal{S}$,

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

□

C. PROPERTIES OF $M_{\mathcal{T}}$ AND PROJECT$_{\mathcal{T}}$

Lemma 8.8. *The $M_{\mathcal{T}}$ operation defined in Eqn. (69) is an acceptable merge function for $\mathcal{S}_{\mathcal{T}}$, in the sense of Defn. 8.1. In addition, Project$_{\mathcal{T}}$ from Eqn. (69) has the following properties: for all $a, a_i, b \in \mathcal{S}_{\mathcal{T}}$ and $I \subseteq \mathbb{N}$,*

$$\textit{Project}_{\mathcal{T}}\left(\bigoplus_{i\in I}{}_{\mathcal{T}}\, a_i\right) = \bigoplus_{i\in I}{}_{\mathcal{T}}\, \textit{Project}_{\mathcal{T}}(a_i) \qquad (70)$$

$$\textit{Project}_{\mathcal{T}}(\textit{Project}_{\mathcal{T}}(a)) = \textit{Project}_{\mathcal{T}}(a) \qquad (71)$$

$$\begin{aligned} \textit{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} \textit{Project}_{\mathcal{T}}(b) &= \textit{Project}_{\mathcal{T}}(a \otimes_{\mathcal{T}} \textit{Project}_{\mathcal{T}}(b)) && (72) \\ &= \textit{Project}_{\mathcal{T}}(\textit{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} b) && (73) \end{aligned}$$

PROOF. (1) ($M_{\mathcal{T}}$ is $\underline{0}_{\mathcal{T}}$-strict.)

(a) For all $a \in \mathcal{S}_{\mathcal{T}}$,

$$\begin{aligned} M_{\mathcal{T}}(a, \underline{0}_{\mathcal{T}}) &= a \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(\underline{0}_{\mathcal{T}}) \\ &= a \otimes_{\mathcal{T}} \begin{pmatrix} (\exists L_1', L_2, L_1, L_2' : (\text{false} \wedge (L_1' = L_2))) \\ \wedge\ (L_1 = L_1') \wedge (L_2 = L_2') \end{pmatrix} \\ &= a \otimes_{\mathcal{T}} (\text{false} \wedge (L_1 = L_1') \wedge (L_2 = L_2')) \\ &= a \otimes_{\mathcal{T}} \text{false} \\ &= a \otimes_{\mathcal{T}} \underline{0} \\ &= \underline{0} \end{aligned}$$

(b) For all $b \in \mathcal{S}_{\mathcal{T}}$, $M_{\mathcal{T}}(\underline{0}_{\mathcal{T}}, b) = \underline{0}_{\mathcal{T}} \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b) = \underline{0}_{\mathcal{T}}$.

(2) ($M_{\mathcal{T}}$ is infinitely distributive in both argument positions.)

(a) For all $a_i, b \in \mathcal{S}_{\mathcal{T}}$ and $I \subseteq \mathbb{N}$,

$$\begin{aligned} M_{\mathcal{T}}\left(\left(\bigoplus_{i \in I}{}_{\mathcal{T}}\, a_i\right), b\right) &= \left(\bigoplus_{i \in I}{}_{\mathcal{T}}\, a_i\right) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b) \\ &= \bigoplus_{i \in I}{}_{\mathcal{T}}\, (a_i \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b)) \\ &= \bigoplus_{i \in I}{}_{\mathcal{T}}\, M_{\mathcal{T}}(a_i, b) \end{aligned}$$

(b) For all $a, b_i \in \mathcal{S}_{\mathcal{T}}$ and $I \subseteq \mathbb{N}$,

$$M_{\mathcal{T}}\left(a, \left(\bigoplus_{i \in I}{}_{\mathcal{T}}\, b_i\right)\right) = a \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}\left(\bigoplus_{i \in I}{}_{\mathcal{T}}\, b_i\right).$$

The operation $\text{Project}_{\mathcal{T}}(T(G_1', L_1', G_2, L_2, G_1, L_1, G_2', L_2'))$ defined in Eqn. (69) constructs the answer relation by considering each tuple in T independently. Consequently, $\text{Project}_{\mathcal{T}}$ distributes over set-union—the $\oplus_{\mathcal{T}}$ operation of $\mathcal{S}_{\mathcal{T}}$—including infinite set-unions. Therefore,

$$\begin{aligned} a \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}\left(\bigoplus_{i \in I}{}_{\mathcal{T}}\, b_i\right) &= a \otimes_{\mathcal{T}} \bigoplus_{i \in I}{}_{\mathcal{T}}\, \text{Project}_{\mathcal{T}}(b_i) \\ &= \bigoplus_{i \in I}{}_{\mathcal{T}} (a \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(b_i)) \\ &= \bigoplus_{i \in I}{}_{\mathcal{T}}\, M_{\mathcal{T}}(a, b_i) \end{aligned}$$

(3) ($M_{\mathcal{T}}$ has the path-extension property.) For all $a, b, c \in \mathcal{S}_{\mathcal{T}}$,

$$\begin{aligned} M_{\mathcal{T}}(a \otimes_{\mathcal{T}} b, c) &= (a \otimes_{\mathcal{T}} b) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(c) \\ &= a \otimes_{\mathcal{T}} (b \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(c)) \\ &= a \otimes_{\mathcal{T}} M_{\mathcal{T}}(b, c) \end{aligned}$$

(4) (Eqn. (70) holds.) The infinite distributivity of $\text{Project}_{\mathcal{T}}$ was argued in the proof of case (2) above.

(5) (Eqn. (71) holds.) First, because $\underline{1}_{\mathcal{T}}$ is the identity relation $\{(g_1, l_1, g_2, l_2, g_1, l_1, g_2, l_2)\}$, it is straightforward to show from Eqn. (69)

$$
\begin{aligned}
&\mathrm{Project}_{\mathcal{T}}(T_1) \otimes_{\mathcal{T}} \mathrm{Project}_{\mathcal{T}}(T_2) \\
&\qquad = \begin{array}{rl} & \{(g_1', a, g_2, b, g_1, a, g_2', b) \mid T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \wedge l_1' = l_2\} \\ \otimes_{\mathcal{T}} & \{(\bar{g}_1', \bar{a}, \bar{g}_2, \bar{b}, \bar{g}_1, \bar{a}, \bar{g}_2', \bar{b}) \mid T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \wedge \bar{l}_1' = \bar{l}_2\} \end{array} \\
&\qquad = \left\{ (g_1', a, g_2, b, \bar{g}_1, \bar{a}, \bar{g}_2', \bar{b}) \;\middle|\; \begin{array}{l} \quad T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \wedge l_1' = l_2 \\ \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \wedge \bar{l}_1' = \bar{l}_2 \\ \wedge\ g_1 = \bar{g}_1' \wedge a = \bar{a} \wedge g_2' = \bar{g}_2 \wedge b = \bar{b} \end{array} \right\} \\
&\qquad = \left\{ (g_1', a, g_2, b, \bar{g}_1, a, \bar{g}_2', b) \;\middle|\; \begin{array}{l} \quad T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \\ \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \wedge\ l_1' = l_2 \wedge \bar{l}_1' = \bar{l}_2 \wedge g_1 = \bar{g}_1' \wedge g_2' = \bar{g}_2 \end{array} \right\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Project}_{\mathcal{T}}(T_1 \otimes_{\mathcal{T}} \mathrm{Project}_{\mathcal{T}}(T_2)) \\
&\qquad = \mathrm{Project}_{\mathcal{T}} \left(\begin{array}{rl} & \left\{ \begin{array}{l} (g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \\ \qquad \mid T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \end{array} \right\} \\ \otimes_{\mathcal{T}} & \left\{ \begin{array}{l} (\bar{g}_1', e, \bar{g}_2, f, \bar{g}_1, e, \bar{g}_2', f) \\ \qquad \mid T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \wedge \bar{l}_1' = \bar{l}_2 \end{array} \right\} \end{array} \right) \\
&\qquad = \mathrm{Project}_{\mathcal{T}} \left(\left\{ \begin{array}{l} (g_1', l_1', g_2, l_2, \bar{g}_1, e, \bar{g}_2', f) \\ \qquad \mid\ T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \\ \qquad \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \qquad \wedge\ \bar{l}_1' = \bar{l}_2 \\ \qquad \wedge\ g_1 = \bar{g}_1' \wedge l_1 = e \\ \qquad \wedge\ g_2' = \bar{g}_2 \wedge l_2' = f \end{array} \right\} \right) \\
&\qquad = \left\{ (g_1', a, g_2, b, \bar{g}_1, a, \bar{g}_2', b) \;\middle|\; \begin{array}{l} \quad T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \\ \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \wedge\ l_1' = l_2 \wedge \bar{l}_1' = \bar{l}_2 \\ \wedge\ g_1 = \bar{g}_1' \wedge l_1 = e \\ \wedge\ g_2' = \bar{g}_2 \wedge l_2' = f \end{array} \right\} \\
&\qquad = \left\{ (g_1', a, g_2, b, \bar{g}_1, a, \bar{g}_2', b) \;\middle|\; \begin{array}{l} \quad T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \\ \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \wedge\ l_1' = l_2 \wedge \bar{l}_1' = \bar{l}_2 \\ \wedge\ g_1 = \bar{g}_1' \wedge g_2' = \bar{g}_2 \end{array} \right\}
\end{aligned}
$$

Fig. 15. Proof of Eqn. (72).

that $\mathrm{Project}_{\mathcal{T}}(\underline{1}_{\mathcal{T}}) = \underline{1}_{\mathcal{T}}$. We then have, for all $a \in \mathcal{S}_{\mathcal{T}}$,

$$
\begin{array}{lll}
\mathrm{Project}_{\mathcal{T}}(\mathrm{Project}_{\mathcal{T}}(a)) & & \\
\quad = \mathrm{Project}_{\mathcal{T}}(\mathrm{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} \underline{1}_{\mathcal{T}}) & & \underline{1}_{\mathcal{T}} \text{ is the identity for } \otimes_{\mathcal{T}} \\
\quad = \mathrm{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} \mathrm{Project}_{\mathcal{T}}(\underline{1}) & & \text{by Eqn. (73)} \\
\quad = \mathrm{Project}_{\mathcal{T}}(a) \otimes_{\mathcal{T}} \underline{1} & & \mathrm{Project}_{\mathcal{T}}(\underline{1}_{\mathcal{T}}) = \underline{1}_{\mathcal{T}} \\
\quad = \mathrm{Project}_{\mathcal{T}}(a) & & \underline{1}_{\mathcal{T}} \text{ is the identity for } \otimes_{\mathcal{T}}
\end{array}
$$

(6) (Eqn. (72) holds.) See Fig. 15.

(7) (Eqn. (73) holds.) See Fig. 16.

□

$$\text{Project}_{\mathcal{T}}(T_1) \otimes_{\mathcal{T}} \text{Project}_{\mathcal{T}}(T_2) = \langle \text{See Fig. 15} \rangle$$

$$
\begin{aligned}
&\text{Project}_{\mathcal{T}}(\text{Project}_{\mathcal{T}}(T_1) \otimes_{\mathcal{T}} T_2) \\
&\quad = \text{Project}_{\mathcal{T}} \left(\begin{array}{l} \quad \left\{ \begin{array}{l} (g_1', c, g_2, d, g_1, c, g_2', d) \\ \qquad \mid T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \wedge l_1' = l_2 \end{array} \right\} \\ \otimes_{\mathcal{T}} \left\{ \begin{array}{l} (\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \qquad \mid T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \end{array} \right\} \end{array} \right) \\
&\quad = \text{Project}_{\mathcal{T}} \left(\left\{ \begin{array}{l} (g_1', c, g_2, d, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \qquad \mid\ T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \wedge l_1' = l_2 \\ \qquad \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \qquad \wedge\ g_1 = \bar{g}_1' \wedge \bar{l}_1' = c \wedge g_2' = \bar{g}_2 \wedge \bar{l}_2 = d \end{array} \right\} \right) \\
&\quad = \left\{ (g_1', a, g_2, b, \bar{g}_1, a, \bar{g}_2', b) \;\middle|\; \begin{array}{l} \quad T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \wedge l_1' = l_2 \\ \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \wedge\ g_1 = \bar{g}_1' \wedge \bar{l}_1' = c \wedge g_2' = \bar{g}_2 \wedge \bar{l}_2 = d \\ \wedge\ c = d \end{array} \right\} \\
&\quad = \left\{ (g_1', a, g_2, b, \bar{g}_1, a, \bar{g}_2', b) \;\middle|\; \begin{array}{l} \quad T_1(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \\ \wedge\ T_2(\bar{g}_1', \bar{l}_1', \bar{g}_2, \bar{l}_2, \bar{g}_1, \bar{l}_1, \bar{g}_2', \bar{l}_2') \\ \wedge\ l_1' = l_2 \wedge \bar{l}_1' = \bar{l}_2 \\ \wedge\ g_1 = \bar{g}_1' \wedge g_2' = \bar{g}_2 \end{array} \right\}
\end{aligned}
$$

Fig. 16. Proof of Eqn. (73).

LEMMA C.1. *For all $b \in \mathcal{S}_{\mathcal{T}}$,*

$$\lightning^{(t,\cdot)}(\mathit{Project}_{\mathcal{T}}(b)) = \mathit{Project}(\lightning^{(t,\cdot)}(b)).$$

PROOF.

$$
\begin{aligned}
\text{LHS} &= \lightning^{(t,\cdot)}(\text{Project}_{\mathcal{T}}(\{(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \mid \\
&\qquad\qquad b(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2')\})) \\
&= \lightning^{(t,\cdot)}(\{(g_1', n, g_2, r, g_1, n, g_2', r) \mid \\
&\qquad\qquad b(g_1', q, g_2, q, g_1, l_1, g_2', l_2')\}) \\
&= \{(g_1, n, g_2', n) \mid b(p, q, p, q, g_1, l_1, g_2', l_2')\}
\end{aligned}
$$

$$
\begin{aligned}
\text{RHS} &= \text{Project}(\lightning^{(t,\cdot)}(\{(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2') \mid \\
&\qquad\qquad b(g_1', l_1', g_2, l_2, g_1, l_1, g_2', l_2')\})) \\
&= \text{Project}(\{(g_1, l_1, g_2', l_2') \mid b(p, q, p, q, g_1, l_1, g_2', l_2')\}) \\
&= \{(g_1, n, g_2', n) \mid b(p, q, p, q, g_1, l_1, g_2', l_2')\} \\
&= \text{LHS}
\end{aligned}
$$

□