Calculating Graph Algorithms for Dominance and Shortest Path[⋆]

Ilya Sergey[1], Jan Midtgaard[2], and Dave Clarke[1]

[1] KU Leuven, Belgium
{first.last}@cs.kuleuven.be
[2] Aarhus University, Denmark
jmi@cs.au.dk

Abstract. We calculate two iterative, polynomial-time graph algorithms from the literature: a dominance algorithm and an algorithm for the single-source shortest path problem. Both algorithms are calculated directly from the definition of the properties by *fixed-point fusion* of (1) a least fixed point expressing all finite paths through a directed graph and (2) Galois connections that capture dominance and path length.
The approach illustrates that reasoning in the style of fixed-point calculus extends gracefully to the domain of graph algorithms. We thereby bridge common practice from the school of program calculation with common practice from the school of static program analysis, and build a novel view on iterative graph algorithms as instances of abstract interpretation.

Keywords: graph algorithms, dominance, shortest path algorithm, fixed-point fusion, fixed-point calculus, Galois connections

1 Introduction

Calculating an *implementation* from a *specification* is central to two active subfields of theoretical computer science, namely the calculational approach to program development [1,12,13] and the calculational approach to abstract interpretation [18, 22, 33, 34]. The advantage of both approaches is clear: the resulting implementations are provably correct by construction. Whereas the former is a general approach to program development, the latter approach is mainly used for developing provably sound static analyses (with notable exceptions [19,25]). Both approaches are anchored in some of the same discrete mathematical structures, namely partial orders, complete lattices, fixed points and Galois connections.

Graphs and graph algorithms are foundational to computer science as they capture the essence of networks, compilers, social connections, and much more. One well-known class of graph algorithms is the shortest path algorithms, exemplified by Dijkstra's single-source shortest path algorithm [16,28]. Dominance

[⋆] This work was carried out while the first author was visiting Aarhus University in the fall of 2011.

algorithms are another class of widespread use: for transforming programs into static single assignment form [5], for optimizing functional programs [30], for ownership typing [35], for information flow analysis [10], etc. In this paper we reconsider the calculational foundations for such algorithms in the context of fixed-point calculus and Galois connections.

In order to bridge two worlds, namely, calculational program development and semantics-based program analysis, we employ the toolset of both fixed-point calculus [1] and abstract interpretation [22], and show that solutions for finite path properties in graphs can be obtained by these means, yielding polynomial-time algorithms that are correct by construction. In doing so, we utilize Galois connections to extract properties (namely dominance and shortest path) from sets of paths in graphs similar to how Galois connections are used to extract properties from program executions in abstract interpretation.

2 Background

We first highlight the relevant mathematical preliminaries. Readers familiar with lattices and orders [27] as found in the fixed-point calculus [1], basic abstract interpretation [26], and relational algebra [7] may wish to proceed directly to Section 3.

2.1 Notation

We use the standard notation $\wp(X)$ for the powerset of X. When working with sets and relations, we will make use of Eindhoven notation for quantified expressions [40]. The general pattern is $\langle Q\ x : p(x) : t(x)\rangle$, where Q is some quantifier (e.g., "$\forall$" or "$\exists$"), x is a sequence of free variables (also called *dummies*), $p(x)$ is a predicate, which must be satisfied by the dummies, and $t(x)$ is an expression, defined in terms of the dummies. For instance, for cases "$\forall$" or "$\exists$", we have the following relations with the standard notation:

$$\begin{aligned}\langle \forall x : p(x) : q(x)\rangle &\iff \forall x.(p(x) \Rightarrow q(x))\\ \langle \exists x : p(x) : q(x)\rangle &\iff \exists x.(p(x) \wedge q(x))\end{aligned}$$

Following the same notation, we use set comprehensions $\{x : p(x) : q(x)\}$ as a shorthand for $\langle \cup x : p(x) : \{q(x)\}\rangle$, where x contains components to union over, p is a filtering condition and $\{q(x)\}$ is a yielded result for a particular combination from x.[3] The square brackets around a formula indicate a universal quantification over any free variables, not mentioned in the preamble. For instance, $[\ x \vee \neg x\ ]$.

In the proofs, we will overload the equality sign "$=$" to mean equivalence between two subsequent steps of a derivation, supplying a textual explanation in fancy brackets: $\wr\ \ldots\ \wr$.

[3] This notation is equivalent to the traditional notation $\{q(x) \mid p(x)\}$, which does not make explicit which variables are bound and which variables are free.

2.2 Basics of domain theory and abstract interpretation

A *complete lattice* $\langle C; \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ is a partial order $\langle C; \sqsubseteq \rangle$ such that there exists a least upper bound (or join) $\sqcup S$ and a greatest lower bound (or meet) $\sqcap S$ of all subsets $S \subseteq C$. In particular $\sqcup C = \top$ and $\sqcap C = \bot$.

A point x is a fixed point of a function f if $f(x) = x$. Given two partial orders, $\langle C, \sqsubseteq \rangle$ and $\langle A, \leq \rangle$, a function f of type $C \to A$ is monotone if $[\ x \sqsubseteq y \implies f(x) \leq f(y)\ ]$. By the Knaster-Tarski fixed-point theorem a monotone endo-function f over a complete lattice has a *least fixed point* $\mathsf{lfp}_{\sqsubseteq} f = \sqcap\{x \mid f(x) \sqsubseteq x\}$. Algorithmically the least fixed point of a monotone function f over a complete lattice of finite *height*[4] can be computed by Kleene iteration: $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot) \sqsubseteq f^3(\bot) \sqsubseteq \ldots$ since $\mathsf{lfp}_{\sqsubseteq} f = \sqcup_{i\geq 0} f^i(\bot)$.

A *Galois connection* is a pair of functions α, γ (in the present case, between two partial orders $\langle C, \sqsubseteq \rangle$ and $\langle A, \leq \rangle$), such that $[\ \alpha(c) \leq a \iff c \sqsubseteq \gamma(a)\ ]$. The function α is referred to as the *lower adjoint* and the function γ is referred to as the *upper adjoint*.[5] We typeset Galois connections as:

$$\langle C, \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq \rangle$$

sometimes with double arrow heads to stress that an adjoint is surjective. Galois connections enjoy a number of properties of which we only highlight a few. Both adjoints of a Galois connection are monotone. Furthermore, for a Galois connection between two complete lattices the lower adjoint distributes over the least upper bound: $[\ \alpha(\sqcup \mathcal{X}) = \bigvee \alpha(\mathcal{X})\ ]$. Finally if a function between two complete lattices distributes over the least upper bound, then it is the lower adjoint of a Galois connection with its corresponding upper adjoint uniquely determined by $\gamma(a) = \sqcup\{c \mid \alpha(c) \leq a\}$ [23].

Galois connections can be constructed compositionally: given $\langle C, \sqsubseteq \rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle A_1, \leq_1 \rangle$ and $\langle A_1, \leq_1 \rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle A_2, \leq_2 \rangle$, one has $\langle C, \sqsubseteq \rangle \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} \langle A_2, \leq_2 \rangle$.

Galois connections interact with least fixed points by *fixed-point fusion* [1]:

$$\alpha \circ F_c \mathrel{\dot{\leq}} F_a \circ \alpha \implies \alpha(\mathsf{lfp}\ F_c) \leq \mathsf{lfp}\ F_a \tag{1}$$

$$\alpha \circ F_c = F_a \circ \alpha \implies \alpha(\mathsf{lfp}\ F_c) = \mathsf{lfp}\ F_a \tag{2}$$

for monotone functions F_c and F_a (where we have written $f \mathrel{\dot{\leq}} g$ for the pointwise ordering $[\ f(x) \leq g(x)\ ]$).[6] Note that we overload the notation for a least fixed point lfp, using it for different domains and orders, without specifying them explicitly, when it is obvious from the context. For instance, in the definitions (1) and (2) we use lfp in both cases, assuming in fact $\mathsf{lfp}_{\sqsubseteq}$ for F_c and $\mathsf{lfp}_{\leq}$ for F_a, respectively.

[4] Traditionally, the *height* of a lattice $\langle C; \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ denotes the maximal length of a (possibly infinite) strictly increasing chain of elements $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \sqsubseteq x_i \in C$.

[5] In the abstract interpretation literature where they are typically associated with some information loss they are known as the *abstraction* and *concretization* functions, respectively [23].

[6] The first implication is also referred to as the *fixed-point transfer theorem* [22] and the latter implication is known as a *complete abstraction* [23].

2.3 Elements of relational algebra

Composition is a well-known operation on relations. For given relations $R \subseteq A \times B$ and $S \subseteq B \times C$, their composition $R \circ S \subseteq A \times C$ is defined as follows:

$$R \circ S \equiv \{x, y, z : \langle x, y\rangle \in R \wedge \langle y, z\rangle \in S : \langle x, z\rangle\}. \tag{3}$$

A particular case of relation composition is function composition. In order for function composition to be consistent with the right-to-left $\circ$-notation, we also think of a function of type $A \to B$ as a relation over $B \times A$ [12].

Another important notion we are going to use is *factors* [7]. Given three relations $R \subseteq A \times B$, $S \subseteq B \times C$ and $T \subseteq A \times C$, the *left factor* $T/S \subseteq A \times B$ and the *right factor* $R\backslash T \subseteq B \times C$ are defined pointwise as follows:

$$[\ x\ T/S\ y \equiv \langle \forall z : y\ S\ z : x\ T\ z\rangle\ ] \tag{4}$$

$$[\ x\ R\backslash T\ y \equiv \langle \forall z : z\ R\ x : z\ T\ y\rangle\ ] \tag{5}$$

Both the notions of composition and factors are helpful for reasoning in *point-free style*: while composition eliminates existential quantifications, the factor operations eliminate universal quantification. It is also notable that

$$[\ R \circ S \subseteq T \iff S \subseteq R\backslash T\ ] \tag{6}$$

$$[\ R \circ S \subseteq T \iff R \subseteq T/S\ ] \tag{7}$$

so we have

$$[\ T/S \supseteq R \iff S \subseteq R\backslash T\ ] \tag{8}$$

which makes it possible to consider the eta-expanded factors $(T/) = \lambda\mathcal{X}.T/\mathcal{X}$ and $(\backslash T) = \lambda\mathcal{X}.\mathcal{X}\backslash T$ as the adjoints of a Galois connection:

$$\langle \wp(B \times C), \subseteq\rangle \underset{(T/)}{\overset{(\backslash T)}{\leftrightarrows}} \langle \wp(A \times B), \supseteq\rangle \tag{9}$$

3 Calculating a Dominance Algorithm

In this section we derive an algorithm to compute a dominance relation of a directed graph. We first express dominance as a lower adjoint over a set of finite paths. We then calculate the dominance computation algorithm using fixed-point fusion with a least fixed point expressing all finite paths through a graph.

3.1 Graphs and finite paths

Definition 1 (Directed Graph). *A* rooted *directed graph is a triple $G = \langle V, E, v_0\rangle$, where V is a set of nodes, $E \subseteq V \times V$ is a set of edges and $v_0 \in V$ is a designated* initial *node.*

We use the notation $(u \to v)$ to indicate the edge $\langle u, v\rangle \in E$. A non-empty ***path*** $\sigma \in V^+$ in a graph G is a sequence of nodes $\sigma = u_0 \dots u_n$, such that for all $i \in 1 \dots n$, $(u_{i-1} \to u_i)$. Given a graph $G = \langle V, E, v_0\rangle$, all finite paths starting from v_0 can be obtained by "walking through the set of edges", which leads to the following definition:

Definition 2 (Finite path functional). [7] *Given a fixed graph $G = \langle V, E, v_0\rangle$, a finite path functional $p_G : \wp(V^+) \to \wp(V^+)$ is defined as follows:*

$$p_G(\mathcal{X}) = \{\sigma, v : \sigma \in \mathcal{X} \wedge (\mathtt{last}(\sigma) \to v) : \sigma v\}, \tag{10}$$

where the function `last` *of type $V^+ \to V$ on non-empty paths is defined by*

$$\mathtt{last}(\sigma u) = u. \tag{11}$$

In some cases, we will also consider `last` as a relation (i.e., $\mathtt{last} \subseteq V \times V^+$) in order to compose it with other relations.

Using the well-known observation [7,26] that $\langle \wp(V^+), \subseteq\rangle$ is a complete lattice with $\sqcup = \cup$, $\sqcap = \cap$, $\bot = \emptyset$ and $\top = V^+$, and the fact that p_G is monotone, one can express the set of finite paths through a graph G, starting in v_0 as the following *least fixed point*:

$$\mathtt{P}_G = \mathsf{lfp}(\lambda \mathcal{X}.\{v_0\} \cup p_G(\mathcal{X})) \tag{12}$$

By a simple inductive argument any finite path through G starting in v_0 belongs to $\mathtt{P}_G$. In the remainder of this section we consider a fixed graph $G = \langle V, E, v_0\rangle$.

3.2 Dominance in finite paths

A classical definition of dominance in a graph is stated as follows [36]:

> *A node u* ***dominates*** *node v if u belongs to every path from the initial node v_0 of the graph to v.*

Our goal is to derive an algorithm for computing dominators directly from the definition above. Clearly, the set of all finite paths cannot be examined in general, since it is infinite in the presence of cycles in the graph. Nevertheless, we start from the definition of dominance in a set of paths.

Definition 3. *The function* `dom` *of type $\wp(V^+) \to \wp(V \times V)$ is defined for all $\mathcal{X} \subseteq V^+$ as follows:*

$$[\ u\ \mathtt{dom}(\mathcal{X})\ v = \langle \forall \sigma : \sigma \in \mathcal{X} \wedge \mathtt{last}(\sigma) = v : u\ \mathtt{in}\ \sigma\rangle\ ], \tag{13}$$

[7] The same functional is traditionally used within the *partial trace collecting semantics* [26].

where the relation $\mathtt{in} \subseteq V \times V^+$ *is defined by:*

$$\mathtt{in} = \mathsf{lfp}(\lambda \mathcal{X}.\mathtt{last} \cup \mathcal{X} \circ \mathtt{pre}) \tag{14}$$

and $\mathtt{pre} \subseteq V^+ \times V^+$ *is* $\{\sigma, v : \sigma \in V^+ \wedge \sigma v \in V^+ : \langle \sigma, \sigma v \rangle\}$

In words, Definition 3 says that u antecedes v for all paths in $\mathcal{X}$ trailed by v. One can notice that if there are no paths in $\mathcal{X}$ that end with v, then all nodes u dominate v. Also, a node u dominates itself for any $\mathcal{X}$.

3.3 A Galois connection between sets of finite paths and dominance relations

We proceed by building a Galois connection between the lattice of finite paths through a graph and a lattice of relations on the nodes, using the dominance function `dom` from Definition 3 as a lower adjoint:

$$\langle \wp(V^+), \subseteq \rangle \underset{\mathtt{dom}}{\overset{\overline{\mathtt{dom}}}{\leftrightarrows}} \langle \wp(V \times V), \supseteq \rangle \tag{15}$$

In order to do so, we first reformulate dominance in point-free style using factors (see Section 2). The new equivalent definition is established by the following lemma:

Lemma 1.

$$\mathtt{dom} = (\mathtt{in}/) \circ f \tag{16}$$

where

$$f(\mathcal{X}) = \{\sigma : \sigma \in \mathcal{X} : \langle \mathtt{last}(\sigma), \sigma \rangle\} \tag{17}$$

If we consider `last` as a relation, we can construct its powerset, $\wp(\mathtt{last})$. If we furthermore view the latter as a type, f's signature is $\wp(V^+) \to \wp(\mathtt{last})$.

Proof. For all $u, v \in V$

$$\begin{aligned}
& u \ \mathtt{dom}(\mathcal{X}) \ v \\
= & \ \wr \text{ by definition (13) } \wr \\
& \langle \forall \sigma : \sigma \in \mathcal{X} \wedge \mathtt{last}(\sigma) = v : u \ \mathtt{in} \ \sigma \rangle \\
= & \ \wr \text{ by definition of } f \text{ (17), definition of / (4) } \wr \\
& u \ \mathtt{in}/f(\mathcal{X}) \ v
\end{aligned}$$

□

Second, we show that `dom` is indeed a lower adjoint of the desired Galois connection. Since `dom` is equivalent to a composition of `in/` and f and we already know that `in/` is a lower adjoint from $\langle \wp(\mathtt{last}), \subseteq \rangle$ to $\langle \wp(V \times V), \supseteq \rangle$ (see Section 2.3), we only need to show that f is a lower adjoint from $\langle \wp(V^+), \subseteq \rangle$ to $\langle \wp(\mathtt{last}), \subseteq \rangle$. The following lemma delivers this result.

Lemma 2. *f is a lower adjoint in a Galois connection*

$$\langle \wp(V^+), \subseteq \rangle \underset{f}{\overset{\overline{f}}{\leftrightarrows}} \langle \wp(\texttt{last}), \subseteq \rangle.$$

Proof. Suppose, $\mathcal{X} \subseteq V^+$ and $T \subseteq \texttt{last}$ (i.e., $[\ u\ T\ \sigma \Rightarrow u = \texttt{last}(\sigma)\ ]$). Then

$$\begin{aligned}
& f(\mathcal{X}) \subseteq T \\
= & \ \wr\ \text{by definition (17)}\ \int \\
& \langle \forall \sigma : \sigma \in \mathcal{X} : \texttt{last}(\sigma)\ T\ \sigma \rangle \\
= & \ \wr\ \text{by definition of } \subseteq\ \int \\
& \mathcal{X} \subseteq \{\sigma : \texttt{last}(\sigma)\ T\ \sigma : \sigma\} \\
= & \ \wr\ T \subseteq \texttt{last}, \text{ taking } T' = \{v, \sigma : v\ T\ \sigma : \sigma\}\ \int \\
& \mathcal{X} \subseteq T'
\end{aligned}$$

Therefore, f is a lower adjoint with upper adjoint the function that maps a relation T which is a subset of `last` to its right component:

$$\overline{f}(T) = \{v, \sigma : v\ T\ \sigma : \sigma\}$$

□

The following corollary states the fixed-point fusion property (2) with respect to `dom`:

Corollary 1. *The function* `dom` *is a lower adjoint in a Galois connections between sets of paths ordered by the $\subseteq$ relation and subsets of $V \times V$ ordered by the $\supseteq$ relation. Furthermore, if h is a monotone function of type $\langle \wp(V^+), \subseteq \rangle \to \langle \wp(V^+), \subseteq \rangle$ and g is a monotone function of type $\langle \wp(V \times V), \supseteq \rangle \to \langle \wp(V \times V), \supseteq \rangle$, then*

$$\texttt{dom} \circ h = g \circ \texttt{dom} \Rightarrow \texttt{dom}(\mathsf{lfp}_{\subseteq}\ h) = \mathsf{lfp}_{\supseteq}\ g,$$

where the least fixed points are computed with respect to the appropriate order relations: $\subseteq$ for h and $\supseteq$ for g.

Proof. Follows from the fact that $\texttt{dom} = (\texttt{in}/) \circ f$ (Lemma 1), the composition property of Galois connections applied to (2) and Lemma 2. □

3.4 A dominance computation functional

Having a Galois connection between the lattice of sets of finite paths and the lattice of dominance relations enable us to derive a functional for computing the dominance relation, induced by the set of *all* paths, which we defined as $\mathtt{P}_G$. In this section, we extract an algorithm to compute the actual dominance relation corresponding to all finite paths in the graph.

We construct the ***dominance computation functional*** $\mathcal{F}_{\mathscr{D}}$ from the finite path functional p_G and the lower adjoint `dom`, such that:

$$\texttt{dom} \circ p_G = \mathcal{F}_{\mathscr{D}} \circ \texttt{dom} \tag{18}$$

Then we can just apply Corollary 1, taking $h = p_G$ and $g = \mathcal{F}_{\mathscr{D}}$.

We derive $\mathcal{F}_{\mathscr{D}}$ by a two-staged derivation. First, we find a function k, such that

$$f \circ p_G = k \circ f \tag{19}$$

Second, we find $\mathcal{F}_{\mathscr{D}}$ such that

$$(\texttt{in}/) \circ k = \mathcal{F}_{\mathscr{D}} \circ (\texttt{in}/) \tag{20}$$

One can then see that for $\mathcal{F}_{\mathscr{D}}$ defined in such a way we have:

$$\begin{aligned}
&\texttt{dom} \circ p_G \\
=\ &\wr \text{ by Lemma 1 } \wr \\
&(\texttt{in}/) \circ f \circ p_G \\
=\ &\wr \text{ by (19) } \wr \\
&(\texttt{in}/) \circ k \circ f \\
=\ &\wr \text{ by (20) } \wr \\
&\mathcal{F}_{\mathscr{D}} \circ (\texttt{in}/) \circ f \\
=\ &\wr \text{ by Lemma 1 } \wr \\
&\mathcal{F}_{\mathscr{D}} \circ \texttt{dom}
\end{aligned}$$

and hence satisfy the requirement from equation (18).

Informally, we obtain the function k from (19) by "pushing" the lower adjoint f under the function definition p_G, a well-known "recipe" within the abstract interpretation community [24]:

$$\begin{aligned}
&f(p_G(\mathcal{X})) \\
=\ &\wr \text{ by definition (17) } \wr \\
&\{\sigma : \sigma \in p_G(\mathcal{X}) : \langle \texttt{last}(\sigma), \sigma \rangle\} \\
=\ &\wr \text{ by definition (10) } \wr \\
&\{\sigma, v : \sigma \in \mathcal{X} \wedge (\texttt{last}(\sigma) \to v) : \langle \texttt{last}(\sigma v), \sigma v \rangle\} \\
=\ &\wr \text{ one-point rule, definition of } \texttt{last} \wr \\
&\{\sigma, u, v : \sigma \in \mathcal{X} \wedge \texttt{last}(\sigma) = u \wedge (u \to v) : \langle v, \sigma v \rangle\} \\
=\ &\wr \text{ by definition (17) } \wr \\
&\{\sigma, u, v : \langle u, \sigma \rangle \in f(\mathcal{X}) \wedge (u \to v) : \langle v, \sigma v \rangle\} \\
=\ &\wr \text{ taking } k(\mathcal{X}) = \{\sigma, u, v : \langle u, \sigma \rangle \in \mathcal{X} \wedge (u \to v) : \langle v, \sigma v \rangle\} \wr \\
&k(f(\mathcal{X}))
\end{aligned}$$

Hence the following lemma:

Lemma 3.

$$f \circ p_G = k \circ f,$$

where $k : \langle \wp(\texttt{last}), \subseteq\rangle \to \langle \wp(\texttt{last}), \subseteq\rangle$ *is defined as follows:*

$$k(\mathcal{X}) = \{\sigma, u, v : \langle u, \sigma\rangle \in \mathcal{X} \wedge (u \to v) : \langle v, \sigma v\rangle\} \tag{21}$$

Now, we obtain $\mathcal{F}_{\mathscr{D}}$ using the same technique as in the previous derivation. Assume $R \in \wp(\texttt{last})$, then for all $u, v \in V$,

$$\begin{aligned}
& u\ (\texttt{in}/k(R))\ v \\
= {} & \wr \text{ by definition (4) } \wr \\
& \langle \forall \sigma : v\ k(R)\ \sigma : u\ \texttt{in}\ \sigma\rangle \\
= {} & \wr \text{ by definition of } k \text{ (21) } \wr \\
& \langle \forall \sigma : \langle \exists w : w \to v : w\ R\ \sigma\rangle : u\ \texttt{in}\ \sigma v\rangle \\
= {} & \wr \text{ by definition of } \texttt{in} \text{ (14) } \wr \\
& \langle \forall \sigma : \langle \exists w : w \to v : w\ R\ \sigma\rangle : u = v \vee u\ \texttt{in}\ \sigma\rangle \\
= {} & \wr \text{ by distributivity and range splitting } \wr \\
& u = v \vee \langle \forall w : w \to v : \langle \forall \sigma : w\ R\ \sigma : u\ \texttt{in}\ \sigma\rangle\rangle \\
= {} & \wr \text{ taking } [\ v\ \texttt{pred}\ w \equiv w \to v\ ] \wr \\
& u = v \vee u\ ((\texttt{in}/R)/\texttt{pred})\ v \\
= {} & \wr \text{ taking } \mathcal{F}_{\mathscr{D}}(\mathcal{X}) = \texttt{id} \cup \mathcal{X}/\texttt{pred} \wr \\
& u\ \mathcal{F}_{\mathscr{D}}(\texttt{in}/R)\ v
\end{aligned}$$

The presented derivation proves the following lemma:

Lemma 4.

$$(\texttt{in}/) \circ k = \mathcal{F}_{\mathscr{D}} \circ (\texttt{in}/),$$

where $\mathcal{F}_{\mathscr{D}} : \langle \wp(V \times V), \supseteq\rangle \to \langle \wp(V \times V), \supseteq\rangle$ *is defined by*

$$\mathcal{F}_{\mathscr{D}}(\mathcal{X}) = \texttt{id} \cup \mathcal{X}/\texttt{pred} \tag{22}$$

with `id` *denoting the identity relation and* `pred` *defined as* $[\ v\ \texttt{pred}\ u \equiv u \to v\ ]$.

We now have all the ingredients to express $\texttt{dom}(\mathsf{P}_G)$ in terms of $\mathcal{F}_{\mathscr{D}}$:

Theorem 1.

$$\texttt{dom}(\mathsf{P}_G) = \mathsf{lfp}_{\supseteq}(\lambda\mathcal{X}.\texttt{dom}(\{v_0\}) \cap (\texttt{id} \cup \mathcal{X}/\texttt{pred})) \tag{23}$$

where the least fixed point $\mathsf{lfp}_{\supseteq}$ *is computed with respect to the partial order* $\langle \wp(V \times V), \supseteq\rangle$.

Proof. First, we note that for all $\mathcal{X} \in \wp(V^+)$

$$\begin{aligned}
&\mathtt{dom}(\{v_0\} \cup p_G(\mathcal{X}))\\
=\ &\wr \text{ by corollary 1 and distributivity of adjoints } \int\\
&\mathtt{dom}(\{v_0\}) \cap \mathtt{dom}(p_G(\mathcal{X}))\\
=\ &\wr \text{ by the properties of } \mathcal{F}_{\mathscr{D}} \text{ (18) } \int\\
&\mathtt{dom}(\{v_0\}) \cap \mathcal{F}_{\mathscr{D}}(\mathtt{dom}(\mathcal{X}))
\end{aligned}$$

Applying Corollary 1, one can see that

$$\mathtt{dom}(\mathtt{P}_G) = \mathsf{lfp}_{\supseteq}(\lambda\mathcal{X}.\mathtt{dom}(\{v_0\}) \cap \mathcal{F}_{\mathscr{D}}(\mathcal{X})), \tag{24}$$

where the least fixed point $\mathsf{lfp}_{\supseteq}$ is computed with respect to the $\supseteq$ ordering. Finally, unfolding the definition of $\mathcal{F}_{\mathscr{D}}$ (22) concludes the proof of the theorem. $\square$

3.5 Dominance equations

From a practical point of view, one is usually more interested in computing a representation of the dominance relation as a map Dom, such that $\mathrm{Dom}(v) = \{u : u\ \mathtt{dom}(\mathtt{P}_G)\ v : u\}$. In this section we construct equivalent data-flow equations and iterative algorithms based on this representation, on the definition of the dominance functional $\mathcal{F}_{\mathscr{D}}$ (22), and on the result of Theorem 1. We thereby bridge the computation of dominance as a least fixed point of the path functional and the more traditional approaches [15].

First, we notice that

$$\begin{aligned}
&u\ \mathtt{dom}(\{v_0\})\ v\\
=\ &\wr \text{ definition (13) } \int\\
&\langle \forall \sigma : \sigma \in \{v_0\} \wedge \mathtt{last}(\sigma) = v : u\ \mathtt{in}\ \sigma \rangle\\
=\ &\wr \text{ since } \sigma \in \{v_0\} \iff \sigma = v_0 \text{ and } u\ \mathtt{in}\ v_0 \iff u = v_0 \int\\
&v = v_0 \Rightarrow u = v_0
\end{aligned}$$

Therefore, we obtain

$$\begin{aligned}
&u\ \mathtt{dom}(\mathtt{P}_G)\ v_0\\
=\ &\wr \text{ definition (12) } \int\\
&u\ \mathtt{dom}(\{v_0\} \cup p_G(\mathtt{P}_G))\ v_0\\
=\ &\wr \text{ since } \mathtt{dom} \text{ is distributive } \int\\
&u\ \mathtt{dom}(\{v_0\})\ v_0 \wedge u\ \mathtt{dom}(p_G(\mathtt{P}_G))\ v_0\\
=\ &\wr \text{ by the observation above, taking } v = v_0,\ [\ u\ \mathtt{dom}(\mathtt{P}_G)\ u\ ] \int\\
&u = v_0
\end{aligned}$$

So, we have an equivalence

$$[\ u\ \texttt{dom}(\mathtt{P}_G)\ v_0 \iff u = v_0\ ] \tag{25}$$

Also, for $v \neq v_0$,

$$\begin{aligned}
&\ u\ \texttt{dom}(\mathtt{P}_G)\ v \\
=&\ \wr\ \text{by Theorem 1, since } \mathsf{lfp} \text{ is a fixed-point operator}\ \wr \\
&\ u\ (\texttt{dom}(\{v_0\}) \cap (\texttt{id} \cup \texttt{dom}(\mathtt{P}_G)/\texttt{pred}))\ v \\
=&\ \wr\ \text{by assumption } v \neq v_0 \text{, so } u\ \texttt{dom}(\{v_0\})\ v\ \wr \\
&\ u\ (\texttt{id} \cup \texttt{dom}(\mathtt{P}_G)/\texttt{pred})\ v \\
=&\ \wr\ \text{by definitions of } / \text{ (4) and } \texttt{pred}\ \wr \\
&\ u = v \vee [\ \forall w : w \to v : u\ \texttt{dom}(\mathtt{P}_G)\ w\ ]
\end{aligned}$$

So, we obtain the second equivalence

$$[\ u\ \texttt{dom}(\mathtt{P}_G)\ v \iff u = v \vee \langle \forall w : w \to v : u\ \texttt{dom}(\mathtt{P}_G)\ w \rangle\ ] \tag{26}$$

Taking $\mathrm{Dom} = \texttt{dom}(\mathtt{P}_G)$ not as a relation, but as a *function* of type $V \to \wp(V)$ defined as $[\ u \in \mathrm{Dom}(v) \equiv u\ \texttt{dom}(\mathtt{P}_G)\ v\ ]$ and the equivalences (25) and (26), we discover the following equivalent data-flow equations for Dom [2]:[8]

$$\begin{aligned}
\mathrm{Dom}(v_0) &= \{v_0\} \\
\mathrm{Dom}(v) &= \bigcap_{w \in \texttt{pred}(v)} \mathrm{Dom}(w) \cup \{v\}
\end{aligned} \tag{27}$$

The statement of Theorem 1 can also be exploited to obtain a simple iterative algorithm for computing the least fixed point of the functional $\mathcal{F}_{\mathscr{D}}$ using Kleene iteration. Figure 1 presents such an algorithm, writing $\texttt{dom}(\{v_0\})$ for the map $\lambda v.(v = v_0\ ?\ \{v_0\} : V)$.

```
1: for $v \in V$ do
2:     $\mathrm{Dom}[v] \leftarrow V$
3: $\mathrm{Dom'} \leftarrow \texttt{dom}(\{v_0\}) \cap \mathcal{F}_{\mathscr{D}}(\mathrm{Dom})$
4: while $\mathrm{Dom} \neq \mathrm{Dom'}$ do
5:     $\mathrm{Dom} \leftarrow \mathrm{Dom'}$
6:     $\mathrm{Dom'} \leftarrow \texttt{dom}(\{v_0\}) \cap \mathcal{F}_{\mathscr{D}}(\mathrm{Dom})$
```

Fig. 1: A straightforward algorithm for computing dominance

Initially, the dominance set for every node is the entire set of nodes, according to the lattice $\langle \wp(V \times V), \supseteq \rangle$ (i.e., $\bot = V \times V$). This dominance set is then being

[8] In order to mimic the traditional presentation [2], we consider `pred` as a function of type $V \to \wp(V)$ defined as $[\ w \in \texttt{pred}(v) \equiv w \to v\ ]$.

```
 1: for $v \in V$ do
 2:     $\text{Dom}[v] \leftarrow V$
 3: $\text{Dom}[v_0] \leftarrow \{v_0\}$
 4: Changed $\leftarrow$ true
 5: while Changed do
 6:     Changed $\leftarrow$ false
 7:     for $v \in V$ do
 8:         newSet $\leftarrow \left(\bigcap_{w \in \texttt{pred}(v)} \text{Dom}[w]\right) \cup \{v\}$
 9:         if newSet $\neq \text{Dom}[v]$ then
10:             $\text{Dom}[v] \leftarrow$ newSet
11:             Changed $\leftarrow$ true
```

Fig. 2: An optimized iterative dominator algorithm [15]

"shrunk", as the algorithm proceeds to consider more paths. In the output of the algorithm every node is dominated by itself. The initial node v_0 in particular is dominated only by itself. All disconnected nodes in the graph are dominated by all nodes.

This algorithm can be optimized further although we make no attempt to calculate our way to these changes. For example, rather than maintaining two dominance maps Dom and Dom' one can make do with a single map. In each iteration one then needs to keep track of stabilization by other means than map comparison, e.g., using a Boolean flag to signal changes to an entry. By unfolding $\mathcal{F}_{\mathscr{D}}$ and making these changes we arrive at the classic algorithm from Figure 2 (see Cooper et al. [15] for more details on the implementation).

3.6 Complexity

The complexity of the derived algorithm is polynomial: the *height* of the lattice of dominance functions is $\mathcal{O}(|V|^2)$, which is an upper bound on the number of iterations. Each iteration of the first algorithm in Figure 1 requires (1) an $\mathcal{O}(|V|^2)$-time equality test between two lattice elements and (2) computing an intersection for each node over all its predecessors in $\mathcal{F}_{\mathscr{D}}$ which takes $\mathcal{O}(|V| \times |E|)$ operations. As a consequence the algorithm takes $\mathcal{O}(|V|^2(|V|^2 + |V| \times |E|)) = \mathcal{O}(|V|^4 + |V|^3 \times |E|)$ time. The optimized algorithm in Figure 2 uses a constant time stabilization test, but still requires computing an intersection over all predecessors for each node. As a result it has $\mathcal{O}(|V|^3 \times |E|)$ worst case time complexity.

The bottleneck of the optimized algorithm is the strategy by which it chooses a node to process in line 7 of Figure 2. By instead iterating through the vertices in *reverse postorder* [3] (i.e., a node is visited before all its successor nodes have been visited), we can avoid a general fixed-point computation. By this strategy we can obtain a $\mathcal{O}(|V| \times |E|)$ time algorithm. By a clever choice of data structures, representing sets using dominator trees, this can be improved to $\mathcal{O}(|V|^2)$ [15].

Even linear time dominance algorithms exist [4], but the O-notation for these hide a non-negligible constant factor. For practical purposes they do not fare as well as a well-engineered iterative algorithm [32]. We refer to Cooper, Harvey, and Kennedy [15] for a historical account of dominator algorithms.

4 Calculating a Shortest Path Algorithm

In this section, we calculate an algorithm solving the single-source shortest path problem for a weighted graph with non-negative edge costs. We augment the definition of directed graphs from Section 3.1 with a function assigning weights to edges. The shortest distance from the source to a target node is then formulated for sets of finite weighted paths and an iterative algorithm is derived by fixed-point fusion. Finally, we modify the property to compute the actual shortest paths and not only the shortest distances.

4.1 Weighted graphs and paths

Definition 4 (Weighted rooted graph). *A* weighted *rooted graph $G_w = \langle V, E, v_0, W\rangle$ is a rooted directed graph $\langle V, E, v_0\rangle$ with a* **weight** *function $W : E \to \mathbb{N}$.*

For nodes $u, v \in V$, we use the notation $(u \xrightarrow{w} v)$ to indicate the edge $\langle u, v\rangle \in E$ and $W(\langle u, v\rangle) = w$. A ***weighted path*** $\tau \in V_w^+$ is a non-empty sequence of interleaving nodes and weights $\tau = u_0 w_1 \dots u_{n-1} w_n u_n$, starting and ending by a node, such that for all $i \in 1 \dots n$, $(u_{i-1} \xrightarrow{w_i} u_i)$.

Definition 5 (Weight of a path [16]). *The* ***weight*** *of a weighted path $\tau = u_0 w_1 \dots u_{n-1} w_n u_n$ is the sum of the weights of its constituent edges:*

$$\|\tau\| = \sum_{i=1}^{n} w_i \tag{28}$$

In the remainder of this section we consider a *fixed* weighted graph $G_w = \langle V, E, v_0, W\rangle$.

4.2 The single-source shortest path property for finite paths

In this section we will focus on the ***single-source shortest-path problem***, which can be defined as follows:

Given a node u_0 and a set of weighted paths $\mathcal{X} = \{\tau : \tau = u_0 w_1 \dots u_{n-1} w_n u_n : \tau\}$, for each v, such that $\tau_v = u_0 \dots v \in \mathcal{X}$, what is the minimum of $\|\tau_v\|$?

Again, our goal is to compute an iterative algorithm for the defined property directly from its definition. In order to do so, we first define the *shortest-path weight* for a set of paths similarly to the canonical property by Cormen et al. [16].

Definition 6 (Shortest-path weight). *Given a set of weighted paths $\mathcal{X}$, then the* ***shortest-path weight*** *from u to v in $\mathcal{X}$ is*

$$\mathtt{dist}(\mathcal{X})(u,v) = \min\{\tau : \tau \in \mathcal{X} \ \wedge \ \tau = u \dots v : \|\tau\|\}, \ \textit{where}$$

$$\min(\emptyset) = \infty.$$

By overloading notation, the single-source shortest-path weight from v_0 to any other node in $\mathcal{X}$ is defined naturally using the function `last` (11) for weighted paths:

$$\mathtt{dist}(\mathcal{X}) = \lambda v. \min\{\tau : \tau \in \mathcal{X} \ \wedge \ \mathtt{last}(\tau) = v : \|\tau\|\}. \tag{29}$$

As in the canonical definition [16], we define the shortest-path weights as a function from a set of finite paths to natural numbers extended with infinity. Still, an arbitrary weighted graph can contain a possibly infinite number of paths from a node u to v. We connect the world of weighted graphs with sets of weighted paths by redefining the path functional from Section 2 for the single-source weighted paths of a weighted graph G_w.

Definition 7 (Weighted finite path functional). *Given a weighted graph $G_w = \langle V, E, v_0, W\rangle$, a weighted finite path functional $p_{G_w} : \wp(V_w^+) \to \wp(V_w^+)$ is defined as follows:*

$$p_{G_w}(\mathcal{X}) = \{\tau, w, v : \tau \in \mathcal{X} \wedge (\mathtt{last}(\tau) \xrightarrow{w} v) : \tau w v\}. \tag{30}$$

Similarly to Section 3.1, $\langle \wp(V_w^+), \subseteq\rangle$ is a complete lattice, so the set of all weighted single-source finite paths in the graph is defined as the following least fixed point:

$$\mathtt{P}_{G_w} = \mathsf{lfp}(\lambda \mathcal{X}. \{v_0\} \cup p_{G_w}(\mathcal{X})) \tag{31}$$

Again by a simple inductive argument any finite weighted path starting in v_0 belongs to $\mathtt{P}_{G_w}$. In this setting, $\mathtt{dist}(\mathtt{P}_{G_w})$ specifies the single-source shortest path property for the whole graph. In the remainder of this section we will derive an algorithm to compute it using fixed-point fusion.

4.3 A Galois connection between sets of finite paths and the shortest path weights

The function `dist` defined in Section 4.2 maps a set of paths to a function, mapping a node to a non-negative weight or infinity (in case a node is unreachable from v_0), so the codomain of `dist` is $\mathscr{E} = V \to \mathbb{N} \cup \{\infty\}$. In order to make it a complete lattice we extend natural arithmetic to infinity:

$$\begin{gathered} \forall n \in \mathbb{N} \ : \ n + \infty = \infty + n = \infty + \infty = \infty \\ \forall n \in \mathbb{N} \ : \ n < \infty \\ \infty \le \infty \end{gathered}$$

Next, we introduce a partial order and the least upper bound on elements δ of $\mathscr{E}$:

$$[\ \delta_1 \dot{\geq} \delta_2 \equiv \forall u \in V\ :\ \delta_1(u) \geq \delta_2(u)\ ] \tag{32}$$

$$[\ \delta_1 \sqcup \delta_2 = \lambda u. \min\{\delta_1(u), \delta_2(u)\}\ ] \tag{33}$$

Finally, one can observe that $\langle \mathscr{E}, \dot{\geq} \rangle$ is a complete lattice with the meet operation provided by (33), $\bot_{\mathscr{E}} = \lambda v.\infty$ and $\top_{\mathscr{E}} = \lambda v.0$. This follows, e.g, from realizing that $\langle \mathbb{N} \cup \{\infty\}, \geq \rangle$ is a complete lattice[9] that can be lifted into a complete lattice over functions with the above pointwise operations.

In order to build the Galois connection between $\langle \wp(V_w^+), \subseteq \rangle$ and $\langle \mathscr{E}, \dot{\geq} \rangle$ using `dist` as a lower adjoint, we need to show that `dist` is distributive with respect to $\sqcup$.

Lemma 5.
$$\left[\bigsqcup_i \mathtt{dist}(\mathcal{X}_i) = \mathtt{dist}(\bigcup_i \mathcal{X}_i) \right]$$

Proof. Let a sequence $\mathcal{X}_i \in \wp(V_w^+)$ be given

$$\begin{aligned}
& \bigsqcup_i \mathtt{dist}(\mathcal{X}_i) \\
= & \wr \text{ by definition of } \sqcup \wr \\
& \lambda u. \min_i(\mathtt{dist}(\mathcal{X}_i)(u)) \\
= & \wr \text{ by definition of } \mathtt{dist} \wr \\
& \lambda u. \min_i(\min\{\tau : \tau \in \mathcal{X}_i \ \wedge\ \mathtt{last}(\tau) = u : \|\tau\|\}) \\
= & \wr \min \text{ is associative and commutative } \wr \\
& \lambda u. \min\{\tau : \tau \in \bigcup_i \mathcal{X}_i \ \wedge\ \mathtt{last}(\tau) = u : \|\tau\|\} \\
= & \wr \text{ by definition of } \mathtt{dist} \wr \\
& \mathtt{dist}(\bigcup_i \mathcal{X}_i)
\end{aligned}$$

□

Recall from Section 2.2 that Lemma 5 guarantees the existence of a Galois connection between the two complete lattices, including a unique upper adjoint $\overline{\mathtt{dist}}$:

$$\langle \wp(V_w^+), \subseteq \rangle \underset{\mathtt{dist}}{\overset{\overline{\mathtt{dist}}}{\leftrightarrows}} \langle \mathscr{E}, \dot{\geq} \rangle$$

[9] The construction corresponds roughly to half an *interval domain* formalized by Cousot and Cousot as a complete product lattice $(\{-\infty\} \cup \mathbb{Z}) \times (\mathbb{Z} \cup \{\infty\})$ [20].

4.4 A shortest-path functional

In this section, we extract an algorithm to compute the shortest-path weight function corresponding to all finite paths in the graph. In order to do so, first, we derive the ***shortest-path functional*** $\mathcal{F}_\delta$ by the "pushing" the lower adjoint `dist` under p_{G_w}:

$$
\begin{aligned}
& \mathtt{dist}(p_{G_w}(\mathcal{X})) \\
= & \wr \text{ by the definition of } p_{G_w} \text{ (30) } \int \\
& \mathtt{dist}(\{\tau, w, v : \tau \in \mathcal{X} \wedge (\mathtt{last}(\tau) \xrightarrow{w} v) : \tau w v\}) \\
= & \wr \text{ by definition of } \mathtt{dist} \text{ (29) } \int \\
& \lambda v.\min\{\tau, w : \tau \in \mathcal{X} \wedge (\mathtt{last}(\tau) \xrightarrow{w} v) : \|\tau w v\|\} \\
= & \wr \text{ by definition of } \|\tau w v\| \text{ (28) } \int \\
& \lambda v.\min\{\tau, w : \tau \in \mathcal{X} \wedge (\mathtt{last}(\tau) \xrightarrow{w} v) : \|\tau\| + w\} \\
= & \wr \text{ taking } u = \mathtt{last}(\tau) \int \\
& \lambda v.\min\{\tau, u, w : \tau \in \mathcal{X} \wedge (u \xrightarrow{w} v) \wedge \mathtt{last}(\tau) = u : \|\tau\| + w\} \\
= & \wr \text{ by the property of min } \int \\
& \lambda v.\min\{u, w : (u \xrightarrow{w} v) : \overbrace{\min\{\tau : \tau \in \mathcal{X} \wedge \mathtt{last}(\tau) = u : \|\tau\|\}}^{\mathtt{dist}(\mathcal{X})(u)} + w\} \\
= & \wr \text{ by folding definition of } \mathtt{dist} \text{ (29) } \int \\
& \lambda v.\min\{u, w : (u \xrightarrow{w} v) : \mathtt{dist}(\mathcal{X})(u) + w\} \\
= & \wr \text{ taking } \mathtt{pred}(v) = \{u : u \xrightarrow{w} v : u\} \text{ and } W(\langle u, v\rangle) = w \int \\
& \lambda v.\min\{u : u \in \mathtt{pred}(v) : \mathtt{dist}(\mathcal{X})(u) + W(\langle u, v\rangle)\} \\
= & \wr \text{ defining } \mathcal{F}_\delta(\mathcal{Y}) = \lambda v.\min\{u : u \in \mathtt{pred}(v) : \mathcal{Y}(u) + W(\langle u, v\rangle)\} \int \\
& \mathcal{F}_\delta(\mathtt{dist}(\mathcal{X}))
\end{aligned}
$$

The derivation above proves the following lemma:

Lemma 6.

$$\mathtt{dist} \circ p_{G_w} = \mathcal{F}_\delta \circ \mathtt{dist}$$

where $\mathcal{F}_\delta$ is of type $\langle \mathscr{E}, \dot{\geq} \rangle \to \langle \mathscr{E}, \dot{\geq} \rangle$ is defined for all $\mathcal{X}$ by

$$\mathcal{F}_\delta(\mathcal{X}) = \lambda v.\min\{u : u \in \mathtt{pred}(v) : \mathcal{X}(u) + W(\langle u, v\rangle)\} \tag{34}$$

We can now notice that $\mathtt{dist}(\{v_0\}) = \lambda v.(v = v_0 \ ? \ 0 : \infty)$, so the following theorem follows naturally:

Theorem 2.

$$\mathtt{dist}(\mathsf{P}_{G_w}) = \mathsf{lfp}_{\dot{\geq}}\,(\lambda\mathcal{X}.(\lambda v.(v = v_0 \ ? \ 0 : \infty)) \sqcup \mathcal{F}_\delta(\mathcal{X})) \tag{35}$$

where the least fixed point $\mathsf{lfp}_{\dot{\geq}}$ is computed with respect to the ordering $\dot{\geq}$ over $\mathscr{E}$, starting from $\bot_{\mathscr{E}} = \lambda v.\infty$.

Proof. Similarly to the proof of Theorem 1, using distributivity of `dist`, Lemma 6, fixed-point fusion (2) and inlining $\texttt{dist}(\{v_0\})$. □

```
for $v \in V$ do
    $\delta(v) \leftarrow \infty$
$\delta' \leftarrow \texttt{dist}(\{v_0\}) \sqcup \mathcal{F}_\delta(\delta)$
while $\delta' \neq \delta$ do
    $\delta \leftarrow \delta'$
    $\delta' \leftarrow \texttt{dist}(\{v_0\}) \sqcup \mathcal{F}_\delta(\delta)$
```

Fig. 3: A straightforward algorithm for single-source shortest paths

Figure 3 provides a first iterative algorithm for computing the least fixed point of the functional $\mathcal{F}_\delta$ using Kleene iteration. Again the algorithm has room for improvement.

```
for $u \in V$ do
    $\delta[u] \leftarrow \infty$
$\delta[v_0] \leftarrow 0$
Changed $\leftarrow$ true
while Changed do
    Changed $\leftarrow$ false
    for $v \in V$ do
        for $u \in \texttt{pred}(v)$ do
            if $\delta[u] + W[u,v] < \delta[v]$ then
                $\delta[v] \leftarrow \delta[u] + W[u,v]$
                Changed $\leftarrow$ true
```

Fig. 4: An optimized imperative single-source shortest path algorithm

By unfolding $\mathcal{F}_\delta$ and maintaining only a single δ-map as in Section 3.5 we arrive at the single-source shortest-path algorithm in Figure 4. The resulting algorithm is strikingly similar to Bellman's iterative algorithm [11] for computing shortest paths: as Bellman's algorithm proceeds by computing a "*monotone sequence*" of "*successive approximations*" so does the derived algorithm. The algorithms differ in that Bellman assumes that all nodes are connected, which allows him initialize the distance to a node with the weight of the direct edge from the source node. For an account of the early history of shortest path algorithms we refer to Schrijver [37].

4.5 Complexity

As Bellman's algorithm [11] the derived algorithm has polynomial time complexity. One can see that the lattice $\langle \mathscr{E}, \dot{\geq} \rangle$ is *noetherian*, i.e., it satisfies the *ascending chain condition* [29] (i.e., every strictly ascending chain $x_1 \dot{\geq} x_2 \dot{\geq} \ldots$ of elements eventually terminates), which guarantees termination of the iterative algorithm, since $\mathcal{F}_\delta$ is monotone. Now let the constant L be the maximal weight of an edge between any two nodes in a given graph. For each node an initial path from the source node cannot contain cycles. Moreover its distance from the source node cannot be improved more than $L \times |V|$ times by a strictly increasing chain. Therefore, for a fixed graph, the length of a corresponding ascending chain in $\langle \mathscr{E}, \dot{\geq} \rangle$ is $\mathcal{O}(|V|^2)$ which bounds the number of while-loop iterations.

Both the first algorithm in Figure 3 and the optimized algorithm in Figure 4 iterate through the predecessors of each node, which takes $\mathcal{O}(|V| + |E|)$ operations for each while-loop iteration. In addition the first algorithm requires an $\mathcal{O}(|V|)$ time stabilization test. Therefore, the worst-case time complexity of both algorithms is $\mathcal{O}(|V|^3 + |V|^2 \times |E|)$, or $\mathcal{O}(|V|^2 \times |E|)$ for a connected graph.

The bottleneck of the optimized algorithm is again the non-optimized iteration in lines 7–11. Since $u \in \texttt{pred}(v)$ if and only if $v \in \texttt{next}(u)$, looping through all nodes u, v such that $u \in \texttt{pred}(v)$ is equivalent to looping through all nodes u, v such that $v \in \texttt{next}(u)$. We can therefore rewrite the for-loops into:

```
for $u \in V$ do
  for $v \in \texttt{next}(u)$ do
    if $\delta[u] + W[u,v] < \delta[v]$ then
      $\delta[v] \leftarrow \delta[u] + W[u,v]$
      Changed $\leftarrow$ true
```

Using an observation from Dijkstra's algorithm, we can process the nodes with less distance from v_0 first. As a consequence *each edge* will be examined *only once*, which leads to the original complexity $\mathcal{O}(|V|^2 + |E|) = \mathcal{O}(|V|^2)$. By further improving the algorithm to quickly locate the next node to process (employing a binary min-heap), we obtain the complexity $\mathcal{O}((|V| + |E|) \times \log(|V|))$ (which is an improvement for sparse graphs [16]).

4.6 Computing the shortest paths

Usually, one wants to compute not only shortest-path weights, but the vertices on shortest paths as well. Traditionally, the representation for shortest paths is implemented by a ***predecessor*** map π. In the canonical literature on graph algorithms [16], for a given graph node v, $\pi(v)$ is either another node or NIL, which means that the node is either the source or that it is unreachable. The shortest-paths algorithms traditionally set the values of π so that the chain of predecessors, originating at a vertex v, runs backwards along *some* shortest path from v_0 to v. In practice, it means that there might be several shortest paths from v_0 to v, however, the canonical algorithm chooses one of them arbitrarily [16].

In order to compute the predecessors for the shortest path, we will use the shortest-path weight property from Section 4.2. The shortest path predecessors of v with respect to the set of finite paths $\mathcal{X}$ are then defined as the predecessors of v on paths from v_0 with the minimal possible weight:

$$\texttt{dist}^{\pi}(\mathcal{X}) = \lambda v.\langle \texttt{dist}(\mathcal{X})(v), \{\tau, u, w : \tau uwv \in \mathcal{X} \ \wedge \ \|\tau\| = \texttt{dist}(\mathcal{X})(v) : u\}\rangle \tag{36}$$

where the codomain of $\texttt{dist}^{\pi}$ is

$$\mathscr{P} = V \to (\mathbb{N} \cup \{\infty\}) \times \wp(V) \tag{37}$$

To derive an algorithm to compute the shortest path predecessors for a given graph, we formulate $\mathscr{P}$ as a complete lattice with an order $\sqsubseteq$, build a Galois connection between $\langle \wp(V_w^+), \subseteq\rangle$ and $\langle \mathscr{P}, \sqsubseteq\rangle$, and employ fixed-point fusion.

In order to simplify the notation, in the remainder of this section we use $\downarrow_1$ and $\downarrow_2$ to refer to the first and second projections of a pair, respectively. The partial order and meet operations on elements π_1, π_2 of $\mathscr{P}$ use a function-lifted lexicographical ordering with respect to componentwise orders $\geq$ and $\subseteq$:

$$\left[\begin{aligned} \pi_1 \sqsubseteq \pi_2 \equiv \forall u \in V : \pi_1(u)\downarrow_1 > \pi_2(u)\downarrow_1 \ \ \vee \\ (\pi_1(u)\downarrow_1 = \pi_2(u)\downarrow_1 \ \wedge \ \pi_1(u)\downarrow_2 \subseteq \pi_2(u)\downarrow_2) \end{aligned}\right] \tag{38}$$

$$[\ \pi_1 \sqcup \pi_2 = \lambda u.\phi(\pi_1(u), \pi_2(u))\ ], \text{ where}$$

$$\phi(\langle m_1, r_1\rangle, \langle m_2, r_2\rangle) = \begin{cases} \langle m_2, r_1\rangle & \text{if } m_1 > m_2 \\ \langle m_1, r_2\rangle & \text{if } m_2 > m_1 \\ \langle m_1, r_1 \cup r_2\rangle & \text{otherwise} \end{cases} \tag{39}$$

One can see, that $\langle \mathscr{P}, \sqsubseteq\rangle$ is a complete lattice with $\bot_{\mathscr{P}} = \lambda u.\langle \infty, \emptyset\rangle$. Similarly to Section 4.3, in order to build a Galois connection between $\langle \wp(V_w^+), \subseteq\rangle$ and $\langle \mathscr{P}, \sqsubseteq\rangle$, using $\texttt{dist}^{\pi}$ as a lower adjoint, we show again that $\texttt{dist}^{\pi}$ is distributive with respect to $\sqcup$:

Lemma 7.

$$\left[\ \bigsqcup_i \texttt{dist}^{\pi}(\mathcal{X}_i) = \texttt{dist}^{\pi}(\bigcup_i \mathcal{X}_i)\ \right]$$

Proof. Similar to the proof of Lemma 5, using case analysis on the arguments to the helper function ϕ (39). □

The computation of the functional $\mathcal{F}_{\pi}$ for the shortest-path predecessors, such that

$$\texttt{dist}^{\pi} \circ p_{G_w} = \mathcal{F}_{\pi} \circ \texttt{dist}^{\pi} \tag{40}$$

is similar to the derivation from Section 4.3, using Lemma 7. The final result is stated by the following lemma:

Lemma 8.

$$\texttt{dist}^{\pi} \circ p_{G_w} = \mathcal{F}_{\pi} \circ \texttt{dist}^{\pi}$$

where $\mathcal{F}_\pi$ is of type $\langle \mathscr{P}, \sqsubseteq\rangle \to \langle \mathscr{P}, \sqsubseteq\rangle$ is defined for all $\mathcal{X}$ by

$$\mathcal{F}_\pi(\mathcal{X}) = \lambda v.\langle m, r\rangle,\ \text{ where } m = \min\{u : u \in \mathtt{pred}(v) : \mathcal{X}(u)\downarrow_1 + W(\langle u, v\rangle)\}$$

$$r = \left\{ u \;\middle|\; \begin{array}{l} u \in \mathtt{pred}(v) \\ \mathcal{X}(u)\downarrow_1 < \infty \\ \mathcal{X}(u)\downarrow_1 + W(\langle u, v\rangle) = m \end{array} \right\} \tag{41}$$

Thus, the sets of predecessors in the single-source shortest paths are then computed as a least fixed point according to the following theorem:

Theorem 3.

$$\mathtt{dist}^\pi(\mathsf{P}_{G_w}) = \mathsf{lfp}_\sqsubseteq\,(\lambda\mathcal{X}.(\lambda v.\langle(v = v_0\ ?\ 0 : \infty), \emptyset\rangle) \sqcup \mathcal{F}_\pi(\mathcal{X})) \tag{42}$$

where the least fixed point $\mathsf{lfp}_\sqsubseteq$ is computed with respect to the ordering $\sqsubseteq$ over $\mathscr{P}$, starting from $\bot_{\mathscr{P}} = \lambda u.\langle\infty, \emptyset\rangle$.

Proof. Similarly to the proof of Theorem 2, using distributivity of `dist`, Lemma 8, fixed-point fusion (2) and inlining $\mathtt{dist}^\pi(\{v_0\}) = \lambda v.\langle(v = v_0\ ?\ 0 : \infty), \emptyset\rangle$ □

Note that unlike traditional algorithms for the single-source shortest path problem [11, 28], our algorithm computes *all* possible shortest paths from the source node. The complexity of the algorithm is determined by the height of the lattice $\langle \mathscr{P}, \sqsubseteq\rangle$, which is $\mathcal{O}(|V|^3)$. However, updating the minimum and the set of predecessors can be performed within the same loop (lines 8–11 in Figure 4):

```
for $v \in V$ do
  for $u \in \mathtt{pred}(v)$ do
    $d \leftarrow \delta[u] + W[u, v]$
    if $d \le \delta[v]$ then
      $\delta[v] \leftarrow d$
      if $d < \delta[v]$ then
        $\pi[v] \leftarrow \{u\}$
      else
        $\pi[v] \leftarrow \pi[v] \cup \{u\}$
      Changed $\leftarrow$ true
```

This gives the same complexity boundary as in Section 4.5: $\mathcal{O}(|V|^3 \times |E|)$ in the worst case. By rewriting the algorithm with `next()` instead of `pred()` and applying observations from Dijkstra's algorithm analysis, one can obtain the complexity bound $\mathcal{O}(|V|^2)$ for the optimized iteration through the set of nodes.

5 Related Work

Two different schools have been working in parallel for the last forty years: the school of program calculation and the school of static program analysis. The intrinsic goal of the first school is to derive algorithms from the specification of properties of interest. The second school was historically interested in computing a *sound* approximation of a property of a program semantics. In this section we give a brief overview of these two lines of research which we have attempted to bridge in the present paper.

Calculational approaches to graph algorithms A number of approaches have been applied to derive graph algorithms since the seventies, originating in formulating path problems in terms of linear algebra. Carré [14] presented an algebraic structure to solve extremal network routing problems, such that a function is minimized or maximized on a particular path in a graph. He showed how extremal problems from this class can be expressed in terms of matrix equations and solved using a toolset from linear algebra. Later, Backhouse and Carré [8] showed the correspondence of the algebra for extremal graph problems and the algebra of regular languages. The idea was later extended to derive the exact implementation of Dijkstra's shortest path algorithm [9].

In the beginning of the nineties ideas from domain theory were applied to compute extremal properties on paths of graphs using fixed-point computations: Van den Eijnde [39] considered computation of path properties in graphs using monotone operators, satisfying certain restrictions and called these operators *conservative.* Van den Eijnde formulated a generalized fixed-point theorem, stating computation of a least fixed point of a monotone functional as a Kleene iteration. The property of interest was then defined as an *under-approximation* of the monotone function. As an example, this approach was applied to the ascending reachability problem. In contrast to our work, Van den Eijnde did not apply the Galois connection machinery to define the properties and prove them appropriate for an algorithm derivation. All the used toolset was later formalized as the *fixed-point calculus* [1]. The interplay between Galois connections and fixed points has later been established by Backhouse [6].

Abstract interpretation and distributive frameworks In parallel with the above line of research, Cousot and Cousot developed and refined the abstract interpretation framework [20,21]. In their 1979 paper [22], they mention various instances of distributive frameworks for imperative program analysis as particular cases of abstract interpretation, i.e., constant propagation, trace (or path) reachability properties, where Galois connections are defined appropriately [22]. In the same work, they prove a connection between properties, defined as *meet-over-all paths* and ones described by monotone functions: the former is generally more precise than the latter but the two are identical in a distributive framework. Ten years later, Cai and Paige describe a nondeterministic iterative schema that in the case of finite iteration generalizes the “chaotic iteration” of Cousot and Cousot for computing fixed points of monotone functions efficiently (in particular, *incrementally*) and show how to apply this technique to design fast non-numerical algorithms, such as variable reachability and cycle detection in a program flow graph [13]. Whereas the current paper illustrates how to get from a graph specification to a provably correct (but not necessarily O-optimal) algorithm, we believe that such chaotic iteration techniques may be the key to derive optimized versions of our calculated graph algorithms in a more principled manner.

Cousot and Cousot [22] initially formalized programs as flow graphs, but the framework was later generalized to *transition systems* [17, 23] which are not limited to describing formal semantics. Since then the abstract interpretation framework has been used to formalize other concepts than static analyses, e.g., program transformations [25] and to connect various forms of formal semantics [19].

Cooper, Harvey and Kennedy [15] point out that the equations to compute dominance form a distributive framework [31]. This fact allows them to state that the iterative algorithm for dominance computation will discover the maximal fixed-point solution and halt. Notably, the equations for Dom, presented by Cooper, Harvey and Kennedy in [15] are given *as is*, i.e., with no connection to the definition of dominance in terms of paths. In contrast we justify these equations by deriving them and a corresponding algorithm directly from the definition.

Backhouse [6, Section 6.2] used shortest paths as a motivating example for introducing fixed-point fusion in his lecture notes. In a later work on the shortest-path problem, Backhouse applied the fixed-point fusion theorem to a set of all paths, considered as a context-free language [7, Example 57], which gave the same solution as we obtained. We have nevertheless chosen to include the detailed development along with our complexity boundary discussion, as a second example of the technique.

Future work A natural next step is to incorporate more benefits of point-free style, such as those provided by relational compositions and factors for the systematic calculation of program analyses, as well as make use of tool support [38] for deriving graph algorithms.

6 Conclusion

In this work we explored two classical graph problems, formulated in terms of finite paths through a graph: dominance and the single-source shortest paths. Applying the toolset traditional to fixed-point calculus and semantics-based program analysis, we derived iterative, polynomial-time algorithms for both properties. We formalized definitions of the properties as adjoints in appropriate Galois connections. By fusing these with a least fixed point of a monotone path functional, we obtained polynomial-time algorithms for computing the properties directly.

The derived algorithms obtained are strikingly similar to independently discovered algorithms from the literature. Their calculations constitute constructive correctness proofs in contrast to, e.g., an invariant argument for Dijkstra's algorithm by contradiction [16]. The derivations further witness the wide applicability of the toolset behind fixed-point calculus and abstract interpretation.

Acknowledgements

We are grateful to Olivier Danvy for comments, which helped to improve the presentation of the paper, and to Jeremy Gibbons for suggestions on both formalism and terminology. We sincerely acknowledge the MPC 2012 reviewers, who *all* provided *excellent* feedback on the submission. In particular, we want to thank Reviewer #1 for suggesting the idea of using factors and showing how to apply it to compute the dominance algorithm,[10] which drastically simplified the derivations in Section 3. Finally, we want to express our gratitude to Shin-Cheng Mu for his dedication to bring out the best of the paper.

References

1. C. Aarts, R. C. Backhouse, E. A. Boiten, H. Doornbos, N. van Gasteren, R. van Geldrop, P. F. Hoogendijk, E. Voermans, and J. van der Woude. Fixed-point calculus. *Information Processing Letters*, 53:131–136, 1995.
2. F. E. Allen. Control flow analysis. *SIGPLAN Not.*, 5:1–19, July 1970.
3. F. E. Allen and J. Cocke. Graph theoretic constructs for program control flow analysis. Technical Report IBM Research Report RC 3923, Thomas J. Watson Research Center, Yorktown Heights, NY, USA, 1972.
4. S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM J. Comput.*, 28(6):2117–2132, 1999.
5. A. W. Appel. *Modern Compiler Implementation in {C, Java, ML}*. Cambridge University Press, New York, 1998.
6. R. C. Backhouse. Galois connections and fixed point calculus. In *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, volume 2297 of *Lecture Notes in Computer Science*, pages 89–148. Springer, 2002.
7. R. C. Backhouse. Regular algebra applied to language problems. *J. Log. Algebr. Program.*, 66(2):71–111, 2006.
8. R. C. Backhouse and B. A. Carré. Regular algebra applied to path-finding problems. *Journal of the Institute of Mathematics and Applications*, 15:161–186, 1975.
9. R. C. Backhouse, J. P. H. W. van den Eijnde, and A. J. M. van Gasteren. Calculating path algorithms. *Sci. Comput. Program.*, 22(1-2):3–19, 1994.
10. R. Barbuti, C. Bernardeschi, and N. De Francesco. Checking security of Java bytecode by abstract interpretation. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 229–236, Madrid, Spain, Mar. 2002. ACM.
11. R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
12. R. Bird and O. de Moor. *The Algebra of Programming.* Prentice-Hall, 1996.
13. J. Cai and R. Paige. Program derivation by fixed point computation. *Sci. Comput. Program.*, 11(3):197–261, 1989.
14. B. A. Carré. An algebra for network routing problems. *J. Inst. Maths Applics.*, 7:273–294, 1971.
15. K. D. Cooper, T. J. Harvey, and K. Kennedy. A simple, fast dominance algorithm. Technical report, Rice University Houston, Texas, USA, 2001.
16. T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.

[10] The degree of elaboration of the review made us speechless for a while.

17. P. Cousot. Semantic foundations of program analysis. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, 1981.
18. P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999.
19. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Comput. Sci.*, 277(1–2):47–103, 2002.
20. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.
21. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In R. Sethi, editor, *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, Jan. 1977.
22. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In B. K. Rosen, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, Jan. 1979.
23. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.
24. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, Aug. 1992.
25. P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In J. C. Mitchell, editor, *Proceedings of the 29th Annual ACM Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, Jan. 2002.
26. P. Cousot and R. Cousot. Basic concepts of abstract interpretation. In R. Jacquart, editor, *Building the Information Society*, pages 359–366. Kluwer Academic Publishers, 2004.
27. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, England, second edition, 2002.
28. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
29. D. Dummit and R. Foote. *Abstract algebra*. Prentice Hall, 1999.
30. M. Fluet and S. Weeks. Contification using dominators. In X. Leroy, editor, *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP'01)*, pages 2–13, Firenze, Italy, Sept. 2001.
31. J. B. Kam and J. D. Ullman. Global data flow analysis and iterative algorithms. *J. ACM*, 23:158–171, January 1976.
32. T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1:121–141, January 1979.
33. J. Midtgaard and T. Jensen. A calculational approach to control-flow analysis by abstract interpretation. In M. Alpuente and G. Vidal, editors, *Static Analysis, 15th International Symposium, SAS 2008*, volume 5079 of *Lecture Notes in Computer Science*, pages 347–362, Valencia, Spain, July 2008. Springer-Verlag.
34. M. Might. Abstract interpreters for free. In R. Cousot and M. Martel, editors, *SAS'10: Proceedings of the 17th international conference on Static analysis*, volume 6337 of *Lecture Notes in Computer Science*, pages 407–421, Perpignan, France, 2010. Springer-Verlag.

35. A. Milanova and J. Vitek. Static dominance inference. In J. Bishop and A. Vallecillo, editors, *Proceedings of the 49th international conference on Objects, models, components, patterns (TOOLS 2011)*, volume 6705 of *Lecture Notes in Computer Science*, pages 211–227, Zurich, Switzerland, 2011. Springer-Verlag.
36. R. T. Prosser. Applications of boolean matrices to the analysis of flow diagrams. In *Proceeding of the Eastern Joint IRE-AIEE-ACM Computer Conference*, pages 133–138, Boston, Massachusetts, 1959. ACM.
37. A. Schrijver. On the History of Combinatorial Optimization (till 1960). In K. Aardal, G. L. Nemhauser, and R. Weismantel, editors, *Handbook of Discrete Optimization*, pages 1–68, 2005.
38. P. F. Silva and J. N. Oliveira. 'Galculator': functional prototype of a Galois-connection based proof assistant. In S. Antoy and E. Albert, editors, *PPDP'08: Proceedings of the 10th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 44–55, July 2008.
39. J. P. H. W. van den Eijnde. Conservative fixpoint functions on a graph. In R. S. Bird, C. Morgan, and J. Woodcock, editors, *Second International Conference on Mathematics of Program Construction (MPC 1992)*, volume 669 of *Lecture Notes in Computer Science*, pages 80–99, Oxford, U.K., July 1992. Springer.
40. A. J. M. van Gasteren. *On the shape of mathematical arguments*. Springer-Verlag New York, Inc., 1990.