An Analysis of Repeated Graph Search

Roland Backhouse [0000−0002−0140−8089]

School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK.
rcb@cs.nott.ac.uk

Abstract. Graph-searching algorithms typically assume that a node is given from which the search begins but in many applications it is necessary to search a graph repeatedly until all nodes in the graph have been "visited". Sometimes a priority function is supplied to guide the choice of node when restarting the search, and sometimes not. We call the nodes from which a search of a graph is (re)started the "delegate" of the nodes found in that repetition of the search and we analyse the properties of the delegate function. We apply the analysis to establishing the correctness of the second stage of the Kosaraju-Sharir algorithm for computing strongly connected components of a graph.

Keywords: directed graph, depth-first search, relation algebra, regular algebra, strongly connected component

1 Introduction

Graph-searching algorithms typically assume that a node is given from which the search begins but in many applications it is necessary to search a graph repeatedly until all nodes in the graph have been "visited". Sometimes a priority function is supplied to guide the choice of node when restarting the search, and sometimes not.

The determination of the strongly connected components of a (directed) graph using the two-stage algorithm attributed to R.Kosaraju and M.Sharir by Aho, Hopcroft and Ullman [1] is an example of both types of repeated graph search.

In the first stage, a repeated search of the given graph is executed until all nodes in the graph have been "visited". In this stage, the choice of node when restarting the search is arbitrary; it is required, however, that the search algorithm is depth-first. The output is a numbering of the nodes in order of completion of the individual searches.

In the second stage, a repeated search of the given graph —but with edges reversed— is executed; during this stage, the node chosen from which the search is restarted is the highest numbered node (as computed in the first stage) that has not been "visited" (during this second stage). Popular accounts of the algorithm [1, 9] require a depth-first search once more but, as is clear from Sharir's original formulation of the algorithm [16], this is not necessary: an arbitrary graph-searching algorithm can be used in the second stage of the algorithm. Each

individual search identifies a strongly connected component of the graph of which the node chosen to restart the search is a representative.

The task of constructing a complete, rigorous, calculational proof of the two-stage Kosaraju-Sharir algorithm is non-trivial. (Glück [13] calls it a "Herculean Task".) The task is made simpler by a proper separation of concerns: both stages use a repeated graph search, the first stage requires depth-first search but the second does not. So what are the properties of repeated graph search (in general), and what characterises depth-first search (in particular)?

The current paper is an analysis of repeated graph search in which we abstract from the details of the Kosaraju-Sharir algorithm. That is, we assume the existence of a "choice" function from the nodes of the graph to the natural numbers that is used to determine which node is chosen from which to restart the search. We call the nodes from which a search of a graph is (re)started the "delegate" of the nodes found in that repetition of the search and we analyse the properties of the delegate function assuming, first, that the choice function is arbitrary (thus allowing different nodes to have the same number) and, second, that it is injective (i.e. different nodes are have different numbers).

The properties we identify are true irrespective of the searching algorithm that is used, contrary to popular accounts of graph searching that suggest the properties are peculiar to depth-first search. For example, all the nodes in a strongly connected component of a graph are assigned the same delegate (irrespective of the graph searching algorithm used) whereas Cormen, Leiserson and Rivest's account [9, theorem 23.13, p.490] suggests that this is a characteristic property of depth-first search.

The primary contribution of this paper is the subject of sections 3 and 4. The definition of "delegate" (a function from nodes to nodes) "according to a given choice function" is stated in section 3.1; an algorithm to compute each node's delegate is presented in section 3.2 and further refined in section 3.3. The algorithm is generic in the sense that no ordering is specified for the choice of edges during the search. (In breadth-first search, edges are queued and the choice is first-in, first-out; in depth-first search, edges are stacked and the choice is first-in, last-out. Other orderings are, of course, possible.)

Section 3.4 explores the properties of the delegate function when the choice function is injective. Section 4 applies the analysis of the delegate function to establishing the correctness of the second stage of the Kosaraju-Sharir algorithm. The proof adds insight into the algorithm by identifying clearly and precisely which elements of the so-called "parenthesis theorem" and the classification of edges in a depth-first search [9, 10] are vital to the identification of strongly connected components.

With the goal of achieving the combination of concision and precision, our development exploits so-called "point-free" relation algebra. This is briefly summarised in section 2.

2 Relation Algebra

For the purposes of this paper, a (directed) graph G is a homogeneous, binary relation on a finite set of nodes. One way to reason about graphs —so-called "pointwise" reasoning— is to use predicate calculus with primitive terms the booleans expressing whether or not the relation G holds between a given pair of nodes. In other words, a graph is a set of pairs —the edge set of the graph— and a fundamental primitive is the membership relation expressing whether or not a given pair is an element of a given graph. In so-called "point-free" relation algebra, on the other hand, relations are the primitive elements and the focus is on the algebraic properties of the fundamental operations on relations: converse, composition, etc. Because our focus here is on paths in graphs —algebraically the reflexive, transitive closure of a graph— we base our calculations firmly on point-free relation algebra.

A relation algebra is a combination of three structures with interfaces connecting the structures. The first structure is a powerset: the homogeneous binary relations on a set A are elements of the powerset $2^{A\times A}$ (i.e. subsets of $A\times A$) and thus form a complete, universally distributive, complemented lattice. We use familiar notations for, and properties of, set union and set intersection without further ado. (Formally, set union is the supremum operator of the lattice and set intersection is the infimum operator.) The complement of a relation R will be denoted by $\neg R$; its properties are also assumed known. The symbols $\perp\!\!\!\perp$ and $\top\!\!\!\top$ are used for the least and greatest elements of the lattice (the empy relation and the universal relation, respectively).

The second structure is composition: composition of the homogeneous binary relations on a set A forms a monoid with unit denoted in this paper by I_A (or sometimes just I if there is no doubt about the type of the relations under consideration). The interface with the lattice structure is that their combination forms a universally distributive regular algebra (called a "standard Kleene algebra" by Conway [8, p.27]). Although not a primitive, the star operator is, of course, a fundamental component of regular algebra. For relation R, the relation R^* is its reflexive-transitive closure; in particular, whereas graph G is interpreted as the edge relation, the graph G^* is interpreted as the path relation. The star operator can be defined in several different but equivalent ways (as a sum of powers or as a fixed-point operator). We assume familiarity with the different definitions as well as properties vital to (point-free) reasoning about paths in graphs such as the star-decomposition rule [3, 7, 5].

The third structure is converse. We denote the converse of relation R by $R^\cup$ (pronounced "R wok"). Converse is an involution (i.e. $(R^\cup)^\cup = R$, for all R). Its interface with the lattice structure is that it is its own adjoint in a Galois connection; its interface with composition is the distributivity property

$$(R\circ S)^\cup \;=\; S^\cup \circ R^\cup \quad .$$

Finally, the interface connecting all three structures is the so-called *modularity rule*: for all relations R, S and T,

$$(1) \qquad R\circ S \cap T \;\subseteq\; R\circ(S \cap R^\cup\circ T) \quad .$$

The (easily derived and equivalent) converse

$$(2) \qquad R \circ S \cap T \;\subseteq\; (R \cap T \circ S^{\cup}) \circ S$$

is also used later.

The axioms outlined above are applicable to homogeneous relations. They can, of course, be extended to heterogeneous relations by including type restrictions on the operators. (For example, the monoid structure becomes a category.) The structure is then sometimes known as an *allegory* [12]. We use $A \leftsquigarrow B$ to denote the type of a relation. The set A is called the *target* and the set B is called the *source* of the relation. A homogeneous relation has type $A \leftsquigarrow A$ for some set A.

Our use of heterogeneous relations in this paper is limited to functions (which we treat as a subclass of relations). Point-free relation algebra enables concise formulations of properties usually associated with functions. A relation R of type $A \leftsquigarrow B$ is *functional* if

$$R \circ R^{\cup} \;\subseteq\; I_A \quad ,$$

it is *injective* if

$$R^{\cup} \circ R \;\subseteq\; I_B \quad ,$$

it is *total* if

$$I_B \;\subseteq\; R^{\cup} \circ R \quad ,$$

and it is *surjective* if

$$I_A \;\subseteq\; R \circ R^{\cup} \quad .$$

We abbreviate "functional relation" to "function" and write $A \leftarrow B$ for the type.

(The arrowheads in $A \leftsquigarrow B$ and $A \leftarrow B$ indicate that we interpret relations as having outputs and inputs, where outputs are on the left and inputs are on the right. Our terminology reflects the choice we have made: the words "functional" and "injective", and simultaneously "total" and "surjective", can be interchanged to reflect an interpretation in which inputs are on the left and outputs are on the right.)

An idiom that occurs frequently in point-free relation algebra has the form

$$f^{\cup} \circ R \circ g$$

where f and g are functional and —often but not necessarily— total. Pointwise this expresses the relation on the source of f and the source of g that holds of x and y when $f.x \,[\![R]\!]\, g.y$. (In words, the value of f at x is related by R to the value of g at y.) The idiom occurs frequently below. For example,

$$f^{\cup} \circ < \circ s$$

is used later to express a relation between nodes a and b of a graph when $f.a < s.b$. This is interpreted as "the search from a finishes before the search from b starts", f and s representing finish and start times, respectively.

The "points" in our algebra are typically the nodes of a graph. Inevitably, we do need to refer to specific nodes from time to time. Points are modelled as "proper, atomic coreflexives".

A *coreflexive* is a relation that is a subset of the identity relation. The coreflexives, viewed as a subclass of the homogeneous relations of a given type, form a complete, universally distributive, complemented lattice under the infimum and supremum operators (which, as we have said, we denote by the symbols commonly used for set intersection and set union, respectively). We use lower-case letters p, q etc. to name coreflexives. So, a coreflexive is a relation p such that $p \subseteq I$. We use $\sim p$ to denote the complement in the lattice of coreflexives of the coreflexive p. This is not the same as the complement of p in the lattice of relations: the relation between them is given by the equation $\sim p = I \cap \neg p$.

Elsewhere, with a different application area, we use the word "monotype" instead of "coreflexive". (See, for example, [4, 14, 11].) We now prefer "coreflexive" because it is application-neutral. Others use the word "test" (eg. [13]).

In general, an *atom* in a lattice ordered by $\sqsubseteq$ and having least element $\bot\!\!\!\bot$ is an element x such that

$$\langle \forall y \;::\; y \sqsubseteq x \;\equiv\; y = x \;\vee\; y = \bot\!\!\!\bot \rangle \quad .$$

Note that $\bot\!\!\!\bot$ is an atom according to this definition. If p is an atom that is different from $\bot\!\!\!\bot$ we say that it is a *proper* atom. A lattice is said to be *atomic* if

$$\langle \forall y \;::\; y \neq \bot\!\!\!\bot \;\equiv\; \langle \exists x : atom.x \wedge x \neq \bot\!\!\!\bot : x \sqsubseteq y \rangle \rangle \quad .$$

In words, a lattice is atomic if every proper element includes a proper atom.

It is necessary to distinguish between atomic coreflexives and atomic relations. We use lower-case letters a, b to denote atomic coreflexives. Proper, atomic coreflexives model singleton sets in set theory; so, when applying the theory to graphs, proper, atomic coreflexive a models $\{u\}$ for some node u of the graph. Similarly, coreflexive p models a subset of the nodes, or, in the context of algorithm development, a predicate on nodes (which explains why they are sometimes called "tests").

A lattice with top element $\top\!\!\!\top$ and supremum operator $\sqcup$ is *saturated* (aka "full") if $\top\!\!\!\top$ is the supremum of the identity function on atoms, i.e. if

$$\top\!\!\!\top \;=\; \langle \sqcup x : atom.x : x \rangle \quad .$$

A powerset is atomic and saturated; since we assume that both the lattice of coreflexives and the lattice of relations form powersets, this is the case for both. The coreflexives are postulated to satisfy the *all-or-nothing rule* [13]:

$$\langle \forall\, a,b,R \;::\; a \circ R \circ b = \bot\!\!\!\bot \;\vee\; a \circ R \circ b = a \circ \top\!\!\!\top \circ b \rangle \quad .$$

Combined with the postulates about coreflexives, the all-or-nothing rule has the consequence that the lattice of relations is a saturated, atomic lattice; the proper atoms are elements of the form $a \circ \top\!\!\!\top \circ b$ where a and b are proper atoms of the lattice of coreflexives. In effect, the relation $a \circ \top\!\!\!\top \circ b$ models the

pair (a, b) in a standard set-theoretic account of relation algebra; the boolean $a \circ R \circ b = a \circ \top\!\!\top \circ b$ plays a role equivalent to the boolean $(a, b) \in [\![R]\!]$ (where $[\![R]\!]$ denotes the interpretation of R as a set of pairs).

The "domain" operators play a central role in relation algebra, particularly in its use in algorithm development. The *right domain* of a relation R is the coreflexive $R^{>}$ (read R "right") defined by $R^{>} = I \cap \top\!\!\top \circ R$. The *left domain* $R^{<}$ (read R "left") is defined similarly. The interpretation of $R^{>}$ is $\{x \mid \langle \exists y :: (y, x) \in [\![R]\!] \rangle\}$. The complement of the right domain of R in the lattice of coreflexives is denoted by $R^{>\bullet}$; similarly $R^{\bullet <}$ denotes the complement of the left domain of R. The left and right domains should not be confused with the source and/or target of a relation (in an algebra of heterogeneous relations).

We assume some familiarity with relation algebra (specifically set calculus, relational composition and converse, and their interfaces) as well as fixed-point calculus and Galois connections. For example, monotonicity properties of the operators, together with transitivity and anti-symmetry of the subset relation, are frequently used without specific mention. On the other hand, because the properties of domains are likely to be unfamiliar, we state the properties we use in the hints accompanying proof steps.

3 Repeated Search and Delegates

In this section, we explore a property of repeated application of graph-searching starting with an empty set of "seen" nodes until all nodes have been seen.

The algorithm we consider is introduced in section 3.2 and further refined in section 3.3. Roughly speaking, the algorithm repeatedly searches a given graph starting from a node chosen from among the nodes not yet seen so as to maximise a "choice function"; at each iteration, the graph searched is the given graph but restricted to edges connecting nodes not yet seen. The algorithm records the chosen nodes in a function that we call a "delegate function", the "delegate" of a node a being the node from which the search that "sees" a is initiated.

Rather than begin with the algorithm, we prefer to begin with a specification of what a repeated search of a graph is intended to implement. The formal specification of the delegate function is given in section 3.1.

Our formulation of the notion of a "delegate" is inspired by Cormen, Leiserson and Rivest's [9, p.490] discussion of a "forefather" function as used in depth-first search to compute strongly connected components of a graph. However, our presentation is more general than theirs. In particular, Cormen, Leiserson and Rivest assume that the choice function is injective. We establish some consequences of this assumption in section 3.4; this is followed in section 3.5 by a comparative discussion of our account and that of Cormen, Leiserson and Rivest.

Aside on Terminology I have chosen to use the word "delegate" rather than "forefather" because it has a similar meaning to the word "representative", as used in "a representative of an equivalence class". Tarjan [17], Sharir [16], Aho, Hopcroft and Ullman [1] and Cormen, Leiserson, Rivest and Stein [10, p.619] call

the representative of a strongly connected component of a graph the "root" of the component. This is a reference to the "forest" of "trees" that is (implicitly) constructed during any repeated graph search. In the two-stage algorithm, however, each stage is a repeated graph search and so to refer to the "root" could be confusing: which forest is meant? Using the word "representative" might also be confusing because it might (wrongly) suggest that the "representative" computed by an arbitrary repeated graph search is a representative of the equivalence class of strongly connected nodes in a graph. The introduction of novel terminology also has the advantage of forcing the reader to study its definition. **End of Aside**

3.1 Delegate Function

Suppose f is a total function of type $\mathbb{N} \leftarrow \mathsf{Node}$, where Node is a finite set of *nodes*. Suppose G is a graph with set of nodes Node. That is, G is a relation of type $\mathsf{Node} \leftrightsquigarrow \mathsf{Node}$. We call f the *choice function* (because it governs the choice of delegates).

A *delegate function on* G *according to* f is a relation φ of type $\mathsf{Node} \leftrightsquigarrow \mathsf{Node}$ with the properties that

$$(3) \qquad \varphi \circ \varphi^{\cup} \subseteq I_{\mathsf{Node}} \subseteq \varphi^{\cup} \circ \varphi \quad , \text{ and}$$

$$(4) \qquad \varphi \subseteq (G^{\cup})^* \quad \wedge \quad G^* \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f \quad .$$

The property (3) states that φ is a total function. Property (4), expressed pointwise and in words, states that for all nodes a and b, node a is the delegate of node b equivales the conjunction of (i) there is a path in G from b to a and (ii) among all nodes c such that there is a path from b to c, node a maximises the value of the choice function f. (The relation "$\geq$" on the right side of the second inclusion is the usual at-least ordering on numbers.)

Note that, because our main motivation for studying repeated graph search is to apply the results to understanding the second stage of the Kosaraju-Sharir algorithm for computing strongly connected components of a graph, the definition of the delegate function is that appropriate to a search of $G^{\cup}$ rather than a search of G.

Delegate functions have a couple of additional properties that we exploit later. These are formulated and proved in the lemma below.

Lemma 5 If φ is a delegate function on G according to f,

$$I \subseteq G^* \circ \varphi \quad \wedge \quad G^* \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f \circ \varphi \quad .$$

In words, there is a path in G from each node to its delegate, and if there is a path in G from node b to node c, the value of f at the delegate of b is at least the value of f at the delegate of c.

Proof First,

$$
\begin{array}{ll}
 & I \subseteq G^* \circ \varphi \\
\Leftarrow & \quad \{ \quad \varphi \text{ is total, i.e. } I \subseteq \varphi^\cup \circ \varphi \quad \} \\
 & \varphi^\cup \subseteq G^* \\
= & \quad \{ \quad \text{converse} \quad \} \\
 & \varphi \subseteq (G^*)^\cup \\
= & \quad \{ \quad (G^*)^\cup = (G^\cup)^* \text{ and definition of delegate: (4)} \quad \} \\
 & \mathsf{true} \ .
\end{array}
$$

Second,

$$
\begin{array}{ll}
 & G^* \subseteq (f \circ \varphi)^\cup \circ \geq \circ f \circ \varphi \\
\Leftarrow & \quad \{ \quad I \subseteq G^* \circ \varphi \text{ (see above)} \quad \} \\
 & G^* \circ G^* \circ \varphi \subseteq (f \circ \varphi)^\cup \circ \geq \circ f \circ \varphi \\
\Leftarrow & \quad \{ \quad G^* \circ G^* = G^* \text{ and monotonicity} \quad \} \\
 & G^* \subseteq (f \circ \varphi)^\cup \circ \geq \circ f \\
= & \quad \{ \quad \text{definition of delegate: (4)} \quad \} \\
 & \mathsf{true} \ .
\end{array}
$$

□

Lemma 6 If φ is a delegate function on G according to f,

$$\varphi \subseteq f^\cup \circ \geq \circ f \ .$$

In words, the delegate of a node has f-value that is at least that of the node.

Proof

$$
\begin{array}{ll}
 & \mathsf{true} \\
= & \quad \{ \quad \text{definition: (3) and (4)} \quad \} \\
 & \varphi \circ \varphi^\cup \subseteq I \ \wedge \ G^* \subseteq (f \circ \varphi)^\cup \circ \geq \circ f \\
\Rightarrow & \quad \{ \quad I \subseteq G^* \text{ and transitivity; converse} \quad \} \\
 & \varphi \circ \varphi^\cup \subseteq I \ \wedge \ I \subseteq \varphi^\cup \circ f^\cup \circ \geq \circ f \\
\Rightarrow & \quad \{ \quad \varphi \circ I = \varphi \text{, monotonicity of composition and transitivity} \quad \} \\
 & \varphi \subseteq f^\cup \circ \geq \circ f \ .
\end{array}
$$

□

3.2 Assigning Delegates

The basic structure of the algorithm for computing a delegate function is shown in fig. 1. It is a simple loop that initialises the coreflexive $seen$ (representing a set of nodes) to $\bot\!\!\!\bot$ (representing the empty set of nodes) and then repeatedly chooses a node a that has the largest f-value among the nodes that do not have a delegate and adds to $seen$ the coreflexive $\sim seen \circ (G^* \circ a)^<$; this coreflexive represents the nodes that do not have a delegate and from which there is a path to a in the graph. Simultaneous with the assignments to $seen$, the variable φ is initialised to $\bot\!\!\!\bot$ and subsequently updated by setting the φ-value of all newly "delegated" nodes to a.

$$
\begin{array}{l}
\{ \quad f \circ f^{\cup} \subseteq I_{\mathsf{N}} \ \wedge \ I_{\mathsf{Node}} \subseteq f^{\cup} \circ f \quad \} \\
\varphi, seen \ := \ \bot\!\!\!\bot, \bot\!\!\!\bot \ ; \\
\{ \quad \textbf{Invariant}\text{: (7) thru (14)} \quad \} \\
\textbf{while} \ \ seen \neq I_{\mathsf{Node}} \ \ \textbf{do} \\
\quad \textbf{begin} \\
\qquad \text{choose node } a \text{ such that } a \circ seen = \bot\!\!\!\bot \text{ and } \sim seen \circ \top\!\!\!\top \circ a \ \subseteq \ f^{\cup} \circ \leq \circ f \\
\quad ; \ \ s \ := \ \sim seen \circ (G^* \circ a)^< \\
\quad ; \ \ \varphi, seen \ := \ \varphi \cup a \circ \top\!\!\!\top \circ s \ , \ seen \cup s \\
\quad \textbf{end} \\
\{ \qquad \varphi \circ \varphi^{\cup} \subseteq I_{\mathsf{Node}} \subseteq \mathsf{equiv}.G \subseteq \varphi^{\cup} \circ \varphi \\
\quad \wedge \quad \varphi \subseteq (G^{\cup} \cap \varphi^{\cup} \circ \varphi)^* \ \wedge \ G^* \ \subseteq \ (f \circ \varphi)^{\cup} \circ \geq \circ f \\
\quad \wedge \quad \varphi = \varphi \circ \varphi \quad \}
\end{array}
$$

Fig. 1. Repeated Search. Outer Loop

For brevity in the calculations below, the temporary variable s (short for "seen") has been introduced. The sequence of assignments

$$
\begin{array}{ll}
 & s \ := \ \sim seen \circ (G^* \circ a)^< \\
; & \varphi, seen \ := \ \varphi \cup a \circ \top\!\!\!\top \circ s \ , \ seen \cup s
\end{array}
$$

is implemented by a generic graph-searching algorithm. The details of how this is done are given in section 3.3.

Apart from being a total function, we impose no restrictions on f. If f is a constant function (for example, if $f.a = 0$ for all nodes a), the "choice" is completely arbitrary.

The relation $\mathsf{equiv}.G$ in the postcondition of the algorithm is the equivalence relation defined by

$$
\mathsf{equiv}.G \ = \ G^* \cap (G^*)^{\cup} \ .
$$

If G is a graph, two nodes related by $\mathsf{equiv}.G$ are said to be in the same *strongly connected component* of G. The first clause of the postcondition thus asserts that the computed delegate relation φ is not only a total function, as required by (3), but also that all nodes in any one strongly connected component are assigned the same delegate.

The property

$$\varphi \subseteq (G^{\cup} \cap \varphi^{\cup} \circ \varphi)^*$$

in the postcondition is stronger than the requirement $\varphi \subseteq (G^{\cup})^*$ in (4). It states that there is a path from each node to its delegate comprising nodes that all have the same delegate. (More precisely, it states that there is a path from each node to its delegate such that successive nodes on the path have the same delegate. The equivalence of these two informal interpretations is formulated in lemma 25.)

Note the property $\varphi = \varphi \circ \varphi$ in the postcondition. Cormen, Leiserson and Rivest [9, p.490] require that the function f is injective and use this to derive the property from the definition of a delegate ("forefather" in their terminology). We don't impose this requirement but show instead that $\varphi = \varphi \circ \varphi$ is a consequence of the algorithm used to calculate delegates. For completeness, we also show that the property is a consequence of the definition of delegate under the assumption that f is injective: see lemma 18. Similarly, the property $\mathsf{equiv}.G \subseteq \varphi^{\cup} \circ \varphi$ can be derived from the definition of a delegate if f is assumed to be injective. Again for completeness, we also show that the property is a consequence of the definition of delegate under the assumption that f is injective: see lemma 19.

Termination of the loop is obvious: the coreflexive $seen$ represents a set of nodes that increases strictly in size at each iteration. (The chosen node a is added at each iteration.) The number of iterations of the loop body is thus at most the number of nodes in the graph, which is assumed to be finite. The principle task is thus to verify conditional correctness (correctness assuming termination, often called "partial" correctness).

The invariant properties of the algorithm are as follows:

(7) $\varphi^{>} = seen$,

(8) $\varphi \circ \varphi^{\cup} \subseteq seen$,

(9) $\varphi \subseteq (G^{\cup} \cap \varphi^{\cup} \circ \varphi)^*$,

(10) $\varphi = \varphi \circ \varphi$,

(11) $seen = (G^* \circ seen)^{<}$,

(12) $seen \circ \sqcap \circ \sim seen \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f$,

(13) $seen \circ G^* \circ seen \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f$,

(14) $seen \circ \mathsf{equiv}.G \circ seen \subseteq \varphi^{\cup} \circ \varphi$.

The postcondition

$$\varphi \circ \varphi^{\cup} \subseteq I_{\mathsf{Node}} \subseteq \varphi^{\cup} \circ \varphi$$

expresses the fact that, on termination, φ is functional and total; the claimed invariants (7) and (8) state that intermediate values of φ are total on *seen* and functional. The invariant (7) also guarantees that *seen* is a coreflexive. The invariants (9) and (10) are both conjuncts of the postcondition. The additional conjunct

$$\mathsf{equiv}.G \subseteq \varphi^{\cup} \circ \varphi$$

in the postcondition states that strongly connected nodes have the same delegate. The invariant (14) states that this is the case for nodes that have been assigned a delegate. Like (7) and (8), invariant (13) states that intermediate values of φ maximise f for those nodes for which a delegate has been assigned. It is therefore obvious that the postcondition is implied by the conjunction of the invariant and the termination condition. The additional invariants (11) and (12) are needed in order to establish the invariance of (13). It is straightforward to construct and check appropriate verification conditions. Full details are given in [6].

3.3 Incremental Computation

The algorithm shown in fig. 1 assigns to the variable s (the coreflexive representing) all the nodes that do not yet have a delegate and can reach the node a. The variable φ is also updated so that a becomes the delegate of all the nodes in the set represented by s. The assignments are implemented by a generic graph-searching algorithm. Fig. 2 shows the details.

The consecutive assignments in the body of the loop in fig. 1 (to s, and to φ and *seen*) are implemented by an inner loop together with initialising assignments. The assertions should enable the reader to verify that the two algorithms are equivalent: the variables s, $seen_0$ and φ_0 are auxiliary variables used to express the property that the inner loop correctly implements the two assignments that they replace in the outer loop; in an actual implementation the assignments to these variables may be omitted (or, preferably, included but identified as auxiliary statements that can be ignored by the computation proper).

It is straightforward to verify the correctness of this algorithm. Because it involves no new techniques, it is omitted here. Full details are included in [6].

A concrete implementation of the above graph-searching algorithm involves choosing a suitable data structure in which to store the unexplored edges represented by $\sim seen \circ (G \circ seen)^{<}$. Breadth-first search stores the edges in a queue (so newly added edges are chosen in the order that they are added), whilst depth-first search stores the edges in a stack (so the most recently added edge is chosen first). Other variations enable the solution of more specific path-finding problems. For example, if edges are labelled by distances, shortest paths from a given source can be found by storing edges in a priority queue. Topological

$\{ \quad a \circ seen = \amalg \wedge$ (7) thru (14) $\quad \}$

$/* \quad s\,,\ seen_0$ and φ_0 are auxiliary variables $\quad */$

$s,seen_0,\varphi_0 \ := \ a,seen,\varphi$

$\{ \quad \sim seen_0 \circ G \circ seen_0 \ = \ \amalg \quad \}$

; $\quad seen,\varphi \ := \ seen \cup a\ ,\ \varphi \cup a \circ \top\!\!\top \circ a$

; $\quad \{$ **Invariant**: $seen \ = \ s \cup seen_0 \ \wedge \ \varphi \ = \ \varphi_0 \cup a \circ \top\!\!\top \circ s$

Invariant: $a \subseteq s \subseteq \sim seen_0 \circ (G^* \circ a)^<$ $\quad \}$

while $\sim seen \circ G \circ seen \neq \amalg$ **do**

begin

choose node b such that $b \subseteq \sim seen \circ (G \circ seen)^<$

$\{ \quad b \subseteq \sim seen_0 \circ (G^* \circ a)^< \quad \}$

; $s \ := \ s \cup b$

; $seen,\varphi \ := \ seen \cup b\ ,\ \varphi \cup a \circ \top\!\!\top \circ b$

end

$\{ \quad s \ = \ \sim seen_0 \circ (G^* \circ a)^< \ \wedge \ seen \ = \ s \cup seen_0 \ \wedge \ \varphi \ = \ \varphi_0 \cup a \circ \top\!\!\top \circ s \quad \}$

$\{ \quad seen \ = \ seen_0 \cup (G^* \circ a)^< \ \wedge \ \varphi \ = \ \varphi_0 \ \cup \ a \circ \top\!\!\top \circ \sim seen_0 \circ (G^* \circ a)^< \quad \}$

Fig. 2. Repeated Search. Inner Loop.

search is also an instance: edges from each node are grouped together and an edge from a given node is chosen when the node has no unexplored incoming edges. We do not go into details any further.

For later discussion of the so-called "white-path theorem" [9, pp.482], we list below some consequences of the invariant properties. Lemmas 15 and 16 relate arbitrary paths to paths that are restricted to unseen nodes; lemma 17 similarly relates arbitrary paths to paths restricted to nodes that have been seen thus far.

Lemma 15 Assuming $seen \ = \ (G^* \circ seen)^<$ (i.e. (11)) and $a \circ seen = \amalg$, the following properties also hold:

$$\sim seen \circ G^* \circ seen \ = \ \amalg \ \wedge \ \sim seen \circ G^* \circ a \ = \ (\sim seen \circ G)^* \circ a \ .$$

(In words, the properties state that there are no paths from an unseen node to a seen node and, for all unseen nodes b there is a path in G from b to a equivales there is a path in G comprising unseen nodes from b to a.)

Proof First,

$\sim seen \circ G^* \circ seen$

$=$ $\quad \{$ domains: $[\ R = R^< \circ R\ ]$ with $R := G^* \circ seen$;

$$seen \;=\; (G^* \circ seen)^<\quad \}$$

$\sim seen \circ seen \circ G^* \circ seen$

$=$ $\{$ $\sim seen \circ seen = \bot\!\!\!\bot$ $\}$

$\bot\!\!\!\bot$.

Second,

$\sim seen \circ G^* \circ a$

$=$ $\{$ $I = seen \cup \sim seen$; distributivity, and star decomposition:

$[\ (R \cup S)^* = R^* \circ (S \circ R^*)^*\ ]$ with $R,S := seen \circ G\ ,\ \sim seen \circ G$ $\}$

$\sim seen \circ (seen \circ G)^* \circ (\sim seen \circ G \circ (seen \circ G)^*)^* \circ a$

$=$ $\{$ $(seen \circ G)^* \;=\; I \cup seen \circ G \circ (seen \circ G)^*$

distributivity and $\sim seen \circ seen = \bot\!\!\!\bot$ $\}$

$\sim seen \circ (\sim seen \circ G \circ (seen \circ G)^*)^* \circ a$

$=$ $\{$ $(seen \circ G)^* \;=\; I \cup seen \circ G \circ (seen \circ G)^*$

distributivity and $\sim seen \circ G^* \circ seen \;=\; \bot\!\!\!\bot$

(whence $\sim seen \circ G \circ seen \;=\; \bot\!\!\!\bot$) $\}$

$\sim seen \circ (\sim seen \circ G)^* \circ a$

$=$ $\{$ $(\sim seen \circ G)^* \;=\; I \cup \sim seen \circ G \circ (\sim seen \circ G)^*$

distributivity $\}$

$\sim seen \circ a \cup \sim seen \circ \sim seen \circ G \circ (\sim seen \circ G)^* \circ a$

$=$ $\{$ $\sim seen \circ a = a$ and $\sim seen \circ \sim seen = \sim seen$,

$(\sim seen \circ G)^* \;=\; I \cup \sim seen \circ G \circ (\sim seen \circ G)^*$

distributivity $\}$

$(\sim seen \circ G)^* \circ a$.

□

The following two lemmas concern the properties of the variable s which is assigned the value $\sim seen \circ (G^* \circ a)^<$ in fig. 1.

Lemma 16 Assuming properties (7) thru (14) and $a \circ seen = \bot\!\!\!\bot$,

$$s \;=\; ((\sim seen \circ G)^* \circ a)^< \ .$$

(In words, the coreflexive s represents the set of all nodes b such that there is a path in G comprising unseen nodes from b to a .)

Proof

s

$=$ { definition (see fig. 1) }

$\sim seen \circ (G^* \circ a)^<$

$=$ { domains: for all coreflexives p and all relations R,

$p \circ R^< \;=\; (p \circ R)^<$ with $p,R := \sim seen\,,\; G^* \circ a$ }

$(\sim seen \circ G^* \circ a)^<$

$=$ { lemma 15 }

$((\sim seen \circ G)^* \circ a)^<$.

□

Lemma 17 Assuming properties (7) thru (14) and $a \circ seen = \perp\!\!\!\perp$,

$$s \;=\; ((s \circ G)^* \circ a)^< \;.$$

(In words, the coreflexive s represents the set of all nodes b such that there is a path in G comprising nodes in s from b to a.)

Proof Applying lemma 16, the task is to prove that

$$((\sim seen \circ G)^* \circ a)^< \;=\; ((s \circ G)^* \circ a)^< \;.$$

Clearly, since $s \subseteq \sim seen$, the left side of this equation is at least the right side. So it suffices to prove the inclusion. This we do as follows.

$((\sim seen \circ G)^* \circ a)^< \;\subseteq\; ((s \circ G)^* \circ a)^<$

$\Leftarrow$ { fixed-point fusion }

$a \;\subseteq\; ((s \circ G)^* \circ a)^<$

$\wedge \quad (\sim seen \circ G \circ ((s \circ G)^* \circ a)^<)^< \;\subseteq\; ((s \circ G)^* \circ a)^<$

$=$ { first conjunct is clearly true;

$\sim seen$

$=$ { case analysis: $I \;=\; (G^* \circ a)^< \cup (G^* \circ a)^{\bullet <}$ }

$\sim seen \circ (G^* \circ a)^< \cup \sim seen \circ (G^* \circ a)^{\bullet <}$

$=$ { definition of s (see fig. 1) }

$s \cup \sim seen \circ (G^* \circ a)^{\bullet <}$ }

$((s \cup \sim seen \circ (G^* \circ a)^{\bullet <}) \circ G \circ ((s \circ G)^* \circ a)^<)^< \;\subseteq\; ((s \circ G)^* \circ a)^<$

$=$ { domains: $[\,(R \circ S^<)^< \;=\; (R \circ S)^<\,]$

with $R,S := (s \cup \sim seen \circ (G^* \circ a)^{\bullet <}) \circ G\,,\; (s \circ G)^* \circ a$;

distributivity }

$$
\begin{array}{ll}
 & (s \circ G \circ (s \circ G)^* \circ a)^< \ \subseteq \ ((s \circ G)^* \circ a)^< \\
\wedge & (\sim seen \circ (G^* \circ a) \bullet^< \circ G \circ (s \circ G)^* \circ a)^< \ \subseteq \ ((s \circ G)^* \circ a)^< \\
\Leftarrow & \quad \{ \quad \text{first conjunct is } \mathsf{true} \text{ (since } [\ R \circ R^* \subseteq R^*\ ] \text{ with } R := s \circ G\text{);} \\
 & \qquad \text{second conjunct: } G \circ (s \circ G)^* \subseteq G^* \text{ and domains} \quad \} \\
 & (\sim seen \circ (G^* \circ a) \bullet^< \circ (G^* \circ a)^<)^< \ \subseteq \ ((s \circ G)^* \circ a)^< \\
= & \quad \{ \quad \text{complements: } (G^* \circ a) \bullet^< \circ (G^* \circ a)^< = \perp\!\!\!\perp \quad \} \\
 & \mathsf{true} \ .
\end{array}
$$

□

3.4 Injective Choice

This section is a preliminary to the discussion in section 3.5. Throughout the section, we assume that f has type $\mathbb{N} \leftarrow \mathsf{Node}$. Also, the symbol I denotes I_{Node}: the identity relation on nodes.

Previous sections have established the existence of a delegate function φ according to choice function f with the only proviso being that f is total and functional. Moreover, the property $\varphi \circ \varphi = \varphi$ is an invariant of the algorithm for computing delegates. Cormen, Leiserson and Rivest [9] derive it from the other requirements assuming that f is also injective. For completeness, this is the point-free rendition of their proof.

Lemma 18 If f is a total, *injective* function and φ is a delegate function according to f, then

$$\varphi \circ \varphi = \varphi \ .$$

Proof

$$
\begin{array}{ll}
 & \varphi \circ \varphi = \varphi \\
\Leftarrow & \quad \{ \quad \text{assumption: } f \text{ is total and injective, i.e. } f^\cup \circ f = I \quad \} \\
 & f \circ \varphi \circ \varphi = f \circ \varphi \\
= & \quad \{ \quad \text{antisymmetry of } \geq \\
 & \qquad \text{and distributivity properties of total functions} \quad \} \\
 & I \subseteq (f \circ \varphi \circ \varphi)^\cup \circ \leq \circ\ f \circ \varphi \ \ \wedge \ \ I \subseteq (f \circ \varphi \circ \varphi)^\cup \circ \geq \circ\ f \circ \varphi \ .
\end{array}
$$

We establish the truth of both conjuncts as follows. First,

$$
\begin{array}{ll}
 & (f \circ \varphi \circ \varphi)^\cup \circ \leq \circ\ f \circ \varphi \\
= & \quad \{ \quad \text{converse} \quad \} \\
 & \varphi^\cup \circ (f \circ \varphi)^\cup \circ \leq \circ\ f \circ \varphi \\
\supseteq & \quad \{ \quad G^* \subseteq (f \circ \varphi)^\cup \circ \geq \circ\ f \circ \varphi \ \text{(lemma 5)}
\end{array}
$$

i.e. $(G^*)^\cup \subseteq (f \circ \varphi)^\cup \circ \leq \circ f \circ \varphi$

(distributivity properties of converse and $\geq^\cup = (\leq)$) }

$\varphi^\cup \circ (G^*)^\cup$

$\supseteq$ { $I \subseteq G^* \circ \varphi$ (lemma 5) and converse }

I .

Second,

$(f \circ \varphi \circ \varphi)^\cup \circ \geq \circ f \circ \varphi$

$=$ { converse }

$\varphi^\cup \circ (f \circ \varphi)^\cup \circ \geq \circ f \circ \varphi$

$\supseteq$ { definition of delegate: (4) and monotonicity }

$\varphi^\cup \circ G^* \circ \varphi$

$\supseteq$ { $I \subseteq G^*$ }

$\varphi^\cup \circ \varphi$

$\supseteq$ { φ is total (by definition: (3)) }

I .

□

As also shown above, the property $\mathsf{equiv}.G \subseteq \varphi^\cup \circ \varphi$ is an invariant of the algorithm. However, if f is a total, injective function, the property follows from the definition of a delegate, as we show below.

Lemma 19 If f is a total, injective function and φ is a delegate function according to f, strongly connected nodes have the same delegate. That is

$$\mathsf{equiv}.G \subseteq \varphi^\cup \circ \varphi \ .$$

Proof

$\mathsf{equiv}.G$

$=$ { definition }

$G^* \cap (G^*)^\cup$

$\subseteq$ { lemma 5 }

$(f \circ \varphi)^\cup \circ \geq \circ f \circ \varphi \ \cap \ ((f \circ \varphi)^\cup \circ \geq \circ f \circ \varphi)^\cup$

$=$ { converse }

$(f \circ \varphi)^\cup \circ \geq \circ f \circ \varphi \ \cap \ (f \circ \varphi)^\cup \circ \leq \circ f \circ \varphi$

$=$ { f and φ are total functions, distributivity }

$(f \circ \varphi)^\cup \circ (\geq \cap \leq) \circ f \circ \varphi$

$$
\begin{array}{ll}
= & \{ \quad \leq \text{ is antisymmetric, converse} \quad \} \\
& \varphi^{\cup} \circ f^{\cup} \circ f \circ \varphi \\
= & \{ \quad f \text{ is injective and total, i.e. } f^{\cup} \circ f = I \quad \} \\
& \varphi^{\cup} \circ \varphi \quad .
\end{array}
$$

□

The relation $\varphi \circ G^{\cup} \circ \varphi^{\cup}$ is a relation on delegates. Viewed as a graph, it is a homomorphic image of the graph $G^{\cup}$ formed by coalescing all the nodes with the same delegate into one node. Excluding self-loops, this graph is acyclic and topologically ordered by f, as we now show.

Definition 20 (Topological Order) A *topological ordering* of a homogeneous relation R of type A is a total, injective function ord from A to the natural numbers with the property that

$$R^{+} \subseteq ord^{\cup} \circ < \circ ord \quad .$$

□

A straightforward lemma is that the requirement on ord is equivalent to

$$R \subseteq ord^{\cup} \circ < \circ ord \quad .$$

Note that the less-than ordering relation on numbers is an implicit parameter of the definition of topological ordering. Sometimes it is convenient to use the greater-than ordering instead. In this way, applying basic properties of converse, it is clearly the case that a topological ordering of R is also a topological ordering of $R^{\cup}$.

Lemma 21 If f is a total, injective function and φ is a delegate function according to f, the graph $\varphi \circ G^{\cup} \circ \varphi^{\cup} \cap \neg I$ is acyclic with f as a topological ordering.

Proof It suffices to show that f is a topological ordering. The function f is, by assumption, a total, injective function of type $\mathbb{N} \leftarrow \mathsf{Node}$. Thus, by assumption, f satisfies the first requirement of being a topological ordering. (See definition 20.) Establishing the second requirement is achieved by the following calculation.

$$
\begin{array}{ll}
& \varphi \circ G^{\cup} \circ \varphi^{\cup} \cap \neg I \quad \subseteq \quad f^{\cup} \circ < \circ f \\
= & \{ \quad \text{shunting rule} \quad \} \\
& \varphi \circ G^{\cup} \circ \varphi^{\cup} \quad \subseteq \quad f^{\cup} \circ < \circ f \ \cup \ I \\
= & \{ \quad f \text{ is total and injective, i.e. } I = f^{\cup} \circ f \\
& \quad \text{distributivity and definition of } \leq \quad \} \\
& \varphi \circ G^{\cup} \circ \varphi^{\cup} \quad \subseteq \quad f^{\cup} \circ \leq \circ f
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \{ \quad \varphi \text{ is functional, i.e. } \varphi \circ \varphi^{\cup} \subseteq I \\
& \qquad \text{monotonicity, converse and transitivity} \quad \} \\
& G^{\cup} \subseteq (f \circ \varphi)^{\cup} \circ \leq \circ f \circ \varphi \\
= & \{ \quad \text{converse} \quad \} \\
& G \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f \circ \varphi \\
\Leftarrow & \{ \quad G \subseteq G^* \text{, transitivity} \quad \} \\
& G^* \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f \circ \varphi \\
= & \{ \quad \text{lemma 5} \quad \} \\
& \mathsf{true} \ .
\end{array}
$$

□

An important corollary of lemma 21 is that the finish timestamp of (repeated) depth-first search is a topological ordering of the strongly connected components of a graph. (See section 3.5 for further discussion of depth-first-search timestamps and lemma 21.)

The algorithm presented in fig. 1 shows that, viewed as a specification of the function φ, the equation (4) always has at least one solution. However, the algorithm is non-deterministic, which means that there may be more than one solution. We now prove that (4) has a unique solution in unknown φ if the function f is total and injective.

Lemma 22 Suppose f of type $\mathbb{N} \leftarrow \mathsf{Node}$ is a total and injective function, and φ and ψ are both total functions of type $\mathsf{Node} \leftarrow \mathsf{Node}$. Then

$$
\begin{array}{ll}
& \varphi = \psi \\
\Leftarrow & \quad (\varphi \subseteq (G^*)^{\cup} \ \wedge \ G^* \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f) \\
& \wedge \quad (\psi \subseteq (G^*)^{\cup} \ \wedge \ G^* \subseteq (f \circ \psi)^{\cup} \circ \geq \circ f) \quad .
\end{array}
$$

Proof Suppose ψ is a total function of type $\mathsf{Node} \leftarrow \mathsf{Node}$. Then

$$
\begin{array}{ll}
& \psi \subseteq (G^*)^{\cup} \ \wedge \ G^* \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f \\
\Rightarrow & \{ \quad \text{converse and transitivity} \quad \} \\
& \psi^{\cup} \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f \\
\Rightarrow & \{ \quad \psi \text{ is total, i.e. } I \subseteq \psi^{\cup} \circ \psi \text{;} \\
& \qquad \text{monotonicity and transitivity} \quad \} \\
& I \subseteq (f \circ \varphi)^{\cup} \circ \geq \circ f \circ \psi \quad .
\end{array}
$$

Interchanging φ and ψ, and combining the two properties thus obtained, we get that, if φ and ψ are both total functions of type $\mathsf{Node} \leftarrow \mathsf{Node}$,

$$
(\varphi \subseteq (G^*)^{\cup} \ \wedge \ G^* \subseteq (f \circ \psi)^{\cup} \circ \geq \circ f)
$$

$$
\begin{array}{ll}
 & \wedge \quad (\psi \subseteq (G^*)^\cup \quad \wedge \quad G^* \subseteq (f \circ \varphi)^\cup \circ \geq \circ f) \\
\Rightarrow & \quad \{ \quad \text{see above} \quad \} \\
 & \quad I \subseteq (f \circ \psi)^\cup \circ \geq \circ f \circ \varphi \\
 & \wedge \quad I \subseteq (f \circ \varphi)^\cup \circ \geq \circ f \circ \psi \\
= & \quad \{ \quad f,\ \varphi \text{ and } \psi \text{ are all total functions,} \\
 & \qquad \text{converse and distributivity} \quad \} \\
 & \quad I \subseteq (f \circ \psi)^\cup \circ ((\leq) \cap (\geq)) \circ f \circ \varphi \\
 & \wedge \quad I \subseteq (f \circ \varphi)^\cup \circ ((\leq) \cap (\geq)) \circ f \circ \psi \\
= & \quad \{ \quad \text{anti-symmetry of } (\leq) \quad \} \\
 & I \subseteq (f \circ \psi)^\cup \circ f \circ \varphi \quad \wedge \quad I \subseteq (f \circ \varphi)^\cup \circ f \circ \psi \\
\Rightarrow & \quad \{ \quad f,\ \varphi \text{ and } \psi \text{ are functional;} \\
 & \qquad \text{hence } f \circ \psi \circ (f \circ \psi)^\cup \subseteq I \text{ and } f \circ \varphi \circ (f \circ \varphi)^\cup \subseteq I \quad \} \\
 & f \circ \psi \subseteq f \circ \varphi \quad \wedge \quad f \circ \varphi \subseteq f \circ \psi \\
= & \quad \{ \quad \text{anti-symmetry} \quad \} \\
 & f \circ \psi = f \circ \varphi \\
\Rightarrow & \quad \{ \quad f \text{ is an injective, total function, i.e. } f^\cup \circ f = I \quad \} \\
 & \psi = \varphi \quad .
\end{array}
$$

The lemma follows by symmetry and associativity of conjunction.
□

Earlier, we stated that (9) formulates the property that there is a path from each node to its delegate on which successive nodes have the same delegate. Combined with (10) and the transitivity of equality, this means that there is a path from each node to its delegate on which all nodes have the same delegate. We conclude this section with a point-free proof of this claim. Since the claim is not specific to the delegate function, we formulate the underlying lemmas (lemmas 23 and 24) in general terms. The relevant property of the delegate function, lemma 25, is then a simple instance.

For readers wishing to interpret lemma 23 pointwise, the key is to note that, for total function h and arbitrary relation S, $h^\cup \circ h \cap S$ relates two points x and y if they are related by S and $h.x = h.y$. However, it is not necessary to do so: completion of the calculation in lemma 24 demands the proof of lemma 23 and this is best achieved by uninterpreted calculation. In turn, lemma 24 is driven by lemma 25 which expresses the delegate function φ as a least fixed point; crucially, this enables the use of fixed-point induction to reason about φ.

Lemma 23 If h is a total function,

$$h \cap R \circ (h^\cup \circ h \cap S) \;=\; h \cap (h \cap R) \circ S$$

for all relations R and S.

Proof By mutual inclusion:

$$
\begin{array}{ll}
 & h \cap (h \cap R) \circ S \\
\subseteq & \quad \{ \quad \text{modularity rule: (1)} \quad \} \\
 & (h \cap R) \circ ((h \cap R)^{\cup} \circ h \ \cap \ S) \\
\subseteq & \quad \{ \quad h \cap R \subseteq h \text{, monotonicity} \quad \} \\
 & (h \cap R) \circ (h^{\cup} \circ h \ \cap \ S) \\
\subseteq & \quad \{ \quad h \text{ is a total function, so } h \circ h^{\cup} \circ h = h \\
 & \qquad h \cap R \subseteq h \text{, distributivity and monotonicity} \quad \} \\
 & h \ \cap \ R \circ (h^{\cup} \circ h \ \cap \ S) \\
= & \quad \{ \quad \text{idempotency (preparatory to next step)} \quad \} \\
 & h \cap h \cap R \circ (h^{\cup} \circ h \ \cap \ S) \\
\subseteq & \quad \{ \quad \text{modularity rule: (2)} \quad \} \\
 & h \ \cap \ (h \circ (h^{\cup} \circ h \ \cap \ S)^{\cup} \ \cap \ R) \circ (h^{\cup} \circ h \ \cap \ S) \\
\subseteq & \quad \{ \quad h \text{ is a total function, so } h \circ h^{\cup} \circ h = h \\
 & \qquad (h^{\cup} \circ h \ \cap \ S)^{\cup} \subseteq h^{\cup} \circ h \text{,} \\
 & \qquad \text{distributivity and monotonicity} \quad \} \\
 & h \cap (h \cap R) \circ S \quad .
\end{array}
$$

□

Lemma 24 If h is a total function,

$$h \cap (h^{\cup} \circ h \cap R)^* \ = \ \langle \mu X \ :: \ h \cap (I \cup X \circ R) \rangle$$

for all relations R.

Proof We derive the right side as follows.

$$
\begin{array}{ll}
 & h \cap (h^{\cup} \circ h \cap R)^* \ = \ \mu g \\
\Leftarrow & \quad \{ \quad \text{fusion theorem} \quad \} \\
 & \langle \forall X :: h \cap (I \cup X \circ (h^{\cup} \circ h \cap R)) \ = \ g.(h \cap X) \rangle \\
= & \quad \{ \quad \text{distributivity, lemma 23 with } R,S := X,R \quad \} \\
 & \langle \forall X :: (h \cap I) \cup (h \cap (h \cap X) \circ R) \ = \ g.(h \cap X) \rangle \\
\Leftarrow & \quad \{ \quad \text{strengthening: } X := h \cap X \quad \} \\
 & \langle \forall X :: (h \cap I) \cup (h \cap X \circ R) \ = \ g.X \rangle
\end{array}
$$

$=$ { distributivity }

$\langle \forall X :: h \cap (I \cup X \circ R) = g.X \rangle$.

□

Lemma 25

$$\varphi \;=\; \langle \mu X :: \varphi \cap (I \;\cup\; X \circ G^{\cup}) \rangle \quad .$$

Proof

φ

$=$ { (9) (i.e., $\varphi \subseteq (G^{\cup} \cap \varphi^{\cup} \circ \varphi)^*$) }

$\varphi \cap (G^{\cup} \cap \varphi^{\cup} \circ \varphi)^*$

$=$ { lemma 24 }

$\langle \mu X :: \varphi \cap (I \;\cup\; X \circ G^{\cup}) \rangle$.

□

The significance of the equality in lemma 25 is the inclusion of the left side in the right side. (The converse is trivial.) Thus, in words, the lemma states that there is a path from each node to its delegate on which every node has the same delegate.

3.5 Summary and Discussion

We summarise the results of this section with the following theorem.

Theorem 26 Suppose f of type $\mathbb{N} \leftarrow \mathsf{Node}$ is a total function and G is a finite graph. Then the equation

$$\varphi \;::\; \varphi \circ \varphi^{\cup} \subseteq I_{\mathsf{Node}} \subseteq \varphi^{\cup} \circ \varphi \;\;\wedge\;\; \varphi \subseteq (G^*)^{\cup} \;\;\wedge\;\; G^* \;\subseteq\; (f \circ \varphi)^{\cup} \circ \geq \circ\, f$$

has a solution with the additional properties that the solution is a closure operator (i.e. a delegate is its own delegate):

$$\varphi \circ \varphi = \varphi \quad ,$$

strongly connected nodes have the same delegate:

$$\mathsf{equiv}.G \subseteq \varphi^{\cup} \circ \varphi$$

and there is a path from each node to its delegate on which successive nodes have the same delegate:

$$\varphi \;\subseteq\; (G^{\cup} \cap \varphi^{\cup} \circ \varphi)^* \quad .$$

More precisely, there is a path from each node to its delegate on which all nodes have the same delegate:

$$\varphi \;=\; \langle \mu X :: \varphi \cap (I \;\cup\; X \circ G^{\cup}) \rangle \quad .$$

Moreover, a delegate has the largest f-value

$$\varphi \subseteq f^{\cup} \circ \geq \circ f \quad .$$

If the function f is injective, the solution is unique; in this case, we call the unique solution *the* delegate function on G according to f. Moreover, f is a topological ordering of the nodes of the graph

$$\varphi \circ G^{\cup} \circ \varphi^{\cup} \cap \neg I$$

(the graph obtained from $G^{\cup}$ by coalescing all nodes with the same delegate and removing self-loops). This graph is therefore acyclic.
□

This paper is inspired by Cormen, Leiserson and Rivest's account of the "fore-father" function and its use in applying depth-first search to the computation of strongly connected components [9, pp.488–494]. However, our presentation is more general than theirs; in particular, we do not assume that the choice function is injective.

The motivation for our more general presentation is primarily to kill two birds with one stone. As do Cormen, Leiserson and Rivest, we apply the results of this section to computing strongly connected components: see section 4. This is one of the "birds". The second "bird" is represented by the case that the choice function is a constant function (for example, $f.a = 0$, for all nodes a). In this case, the choice of node a in the algorithm of fig. 1 reduces to the one condition $a \circ seen = \bot\!\!\!\bot$ (in words, a has not yet been seen) and the function f plays no role whatsoever. Despite this high level of nondeterminism, the specification of a delegate (see section 3.1) allows many solutions that are not computed by the algorithm. (For example, the identity function satisfies the specification.) The analysis of section 3.2 is therefore about the properties of a function that records the history of repeated searches of a graph until all nodes have been seen: the delegate function computed by repeated graph search records for each node b, the node a from which the search that sees b was initiated.

This analysis reveals many properties of graph searching that other accounts may suggest are peculiar to depth-first search. Most notable is the property that strongly connected nodes are assigned the same delegate. As shown in lemma 19, this is a necessary property when the choice function is injective; otherwise, it is not a necessary property but it is a property of the delegate function computed by repeated graph search, whatever graph-searching algorithm is used. The second notable property of repeated graph search is that there is a path from each node to its delegate on which all nodes have the same delegate. This is closely related to the property that Cormen, Leiserson and Rivest call the "white-path theorem" [9, pp.482], which we discuss shortly. Our analysis shows that the property is a generic property of repeated graph search and not specific to depth-first search.

In order to discuss the so-called "white-path theorem", it is necessary to give a preliminary explanation. Operational descriptions of graph-searching algorithms often use the colours white, grey and black to describe nodes. A white node is a node that has not been seen, a grey node is a node that has been seen

but not all edges from the node have been "processed", and a black node is a node that has been seen and all edges from the node have been "processed". The property "white", "grey" or "black" is, of course, time-dependent since initially all nodes are white and on termination all nodes are black.

Now lemmas 16 and 17 express subtley different versions of what is called the "white-path theorem". Suppose a search from node a is initiated in the *outer* loop of a repeated graph search. The search finds nodes on paths starting from a. There are three formally different properties of the paths that are found:

(i) The final node on the path is white at the time the search from a is initiated.
(ii) All nodes on the path are white at the time the search from a is initiated.
(iii) All nodes on the path are white at the time the search from their predecessor on the path is initiated.

In general, if nodes are labelled arbitrarily white or non-white, the sets of paths described by (i), (ii) and (iii) are different. (They are ordered by the subset relation, with (i) being the largest and (iii) the smallest.) However, in a repeated graph search, the sets of paths satisfying (i) and (ii) are equal. This is the informal meaning of lemma 15. Moreover, the right side of the assignment to s in fig. 1 is the set of nodes reached by paths satisfying (i); lemma 16 states that, in a repeated graph search, the nodes that are added by a search initiated from node a are the nodes that can be reached by a path satisfying (ii).

We claim —without formal proof— that it is also the case that, in a repeated graph search, all three sets of paths are equal. That is, the set of paths described by (iii) is also equal to the set of paths described by (i). We don't give a proof here because it is impossible to express formally without introducing additional auxiliary variables. Informally, it is clear from the implementation shown in fig. 2, in particular the choice of nodes b and c. The introduction of timestamps does allow us to prove the claim formally for depth-first search. See section 4.

Cormen, Leiserson and Rivest's [9, pp.482] "white-path theorem" states that it is a property of depth-first search that paths found satisfy (ii). Characteristic of depth-first search is that the property is true for all nodes, and not just nodes from which a search is initiated in the outer loop.

Finally, let us briefly remark on lemma 21. As we see later, not only can depth-first search be used to calculate the strongly connected components of a graph, in doing so it also computes a topological ordering of these components (more precisely a topological ordering of the homomorphic-image graph discussed in section 4). Lemma 21 is more general than this. It states that, if the choice function is injective, it is a topological ordering of the converse of the graph obtained by coalescing all the nodes with the same delegate and then omitting self-loops. In fact, this is also true of the delegate function computed as above. We leave its proof to the reader: remembering that during execution of the algorithm φ is partial with right domain $\varphi^{>}$, identify and verify an invariant that states that f is a topological ordering on a subgraph of $\varphi \circ G^{\cup} \circ \varphi^{\cup} \cap \neg I$.

4 Strongly Connected Components

Recall that if G is a relation, the relation $\mathsf{equiv}.G$ defined by

$$\mathsf{equiv}.G \;=\; G^* \cap (G^*)^\cup$$

is an equivalence relation; if G is a graph, two nodes related by $\mathsf{equiv}.G$ are said to be in the same *strongly connected component* of G.

An equivalence relation R on a set A is typically represented by a so-called *representative* function ρ of type $A \leftarrow A$ with the property that

$$R \;=\; \rho^\cup \circ \rho \quad .$$

For each element a of A, the element $\rho.a$ is called the *representative* of the equivalence class containing a. In words, two values a and b are equivalent (under R) iff they have the same representative.

The calculation of (a representative-function representation of the) strongly connected components of a given graph is best formulated as a two-stage process. In the first stage, a repeated depth-first search of the graph is executed; the output of this stage is a function f from nodes to numbers that records the order in which the search from each node finishes; we call it the *finish timestamp*. In the second stage a repeated search of the converse of the graph is executed using the function f as choice function.

Aside on Sharir's algorithm As mentioned in the introduction, Aho, Hopcroft and Ullman [1] attribute the algorithm to an unpublished 1978 document by R.Kosaraju and to M.Sharir [16]. Sharir's formulation of the algorithm supposes that a forest of trees is computed in the first stage; the ordering of the nodes is then given by a reverse postorder traversal of the trees in the forest. This is non-deterministic since the ordering of the trees is arbitrary. However, a well-known fact is that the use of the finish timestamp is equivalent to ordering the trees according to the reverse of the order in which they are constructed in the first stage; its use is also more efficient and much simpler to implement. Also, contrary to the suggestion in [1, 9, 10] and apparently not well-known, Sharir's formulation of the algorithm demonstrates that is not necessary to use depth-first search in the second stage: any graph searching algorithm will do. **End of Aside**

In this section, we establish the correctness of the second stage assuming certain properties of the first stage. Formally, we prove that the delegate function on G according to the timestamp f is a representative function for the strongly connected components of G.

The properties that we need involve the use of an additional function s from nodes to numbers that records the order in which the search in the first stage from each node starts. More precisely, the combination of the functions s and f records the order in which searches start and finish; the functions s and f are thus called the start and finish *timestamps*, respectively. Unlike f, which is used as a choice function in the second stage, the role of s is purely as an auxiliary variable. That is, the process of recording the start timestamp can be

omitted from the computation proper because it only serves to document the properties of depth-first search.

The properties of repeated depth-first search that we assume are four-fold. First, for all nodes a and b, if the search from a starts before the start of the search from b, and the search from a finishes after the search from b finishes there is a path from a to b:

$$(27) \qquad s^{\cup} \circ \leq \circ s \ \cap \ f^{\cup} \circ \geq \circ f \ \subseteq \ G^* \quad .$$

Second, for all nodes a and b, if the search from a starts strictly before the start of the search from b and finishes strictly before the finish of the search from b, the search from a finishes strictly before the search from b starts:

$$(28) \qquad s^{\cup} \circ < \circ s \ \cap \ f^{\cup} \circ < \circ f \ = \ f^{\cup} \circ < \circ s \quad .$$

Thirdly, if there is an edge from node a to node b in the graph, the search from node a finishes after the search from b starts:

$$(29) \qquad G \subseteq f^{\cup} \circ \geq \circ s \quad .$$

Finally, s and f are total, injective functions from the nodes to natural numbers.

Properties (27) and (28) are both consequences of the so-called "parenthesis theorem" [9, p.480] and [10, p.606]. (The "parenthesis theorem" bundles together the so-called "parenthesis structure" of the start and finish times with properties of paths in the graph.) Property (29) is a consequence of the classification of edges into tree/ancestor edges, fronds or vines [17]; see also [9, exercise 23.3-4, p.484] (after correction to include self-loops as in [10, exercise 22.3-5, p.611]). (Property (29) is sometimes stated in its contrapositive form: King and Launchbury [15], for example, formulate it as there being no "left-right cross edges".)

It may help to present further details of repeated depth-first search. The outer loop —the repeated call of depth-first search— takes the following form:

f,s := $\perp\!\!\!\perp$,$\perp\!\!\!\perp$;
while $s^{>} \neq I_{\mathsf{Node}}$ **do**
  begin
    choose node a such that $a \circ s^{>} = \perp\!\!\!\perp$
  ; $\mathsf{dfs}(a)$
  end
{ (27) $\wedge$ (28) $\wedge$ (29)
  $\wedge$ $s \circ s^{\cup} \subseteq I_{\mathbb{N}}$ $\wedge$ $s^{\cup} \circ s = I_{\mathsf{Node}} = f^{\cup} \circ f$ $\wedge$ $f \circ f^{\cup} \subseteq I_{\mathbb{N}}$ }

The implementation of $\mathsf{dfs}(a)$ is as follows:

$s \;:=\; s \cup \overline{(\mathsf{MAX}.s \uparrow \mathsf{MAX}.f)+1} \circ \top\!\!\top \circ a$

; **while** $a \circ G \circ s^{>\bullet} \neq \bot\!\!\bot$ **do**

begin

choose node b such that $a \circ \top\!\!\top \circ b \subseteq a \circ G \circ s^{>\bullet}$

; $\mathsf{dfs}(b)$

end

; $f \;:=\; f \cup \overline{(\mathsf{MAX}.s \uparrow \mathsf{MAX}.f)+1} \circ \top\!\!\top \circ a$

In the above, the current "time" is given by $\mathsf{MAX}.s \uparrow \mathsf{MAX}.f$ (assuming that $\mathsf{MAX}.\bot\!\!\bot$ is 0, by definition): the maximum value of the combined functions s and f; the overbar denotes the conversion of a number into a coreflexive representing the singleton set containing that number. The assignment to s thus increments the time by 1 and assigns to the node a the new time as starting time; similarly, the assignment to f increments the time by 1 and assigns to the node a the new time as finish time. The coreflexive $s^{>\bullet}$ is the complement of $s^{>}$; thus, it represents the set of nodes from which a search has not yet been started. The body of the inner loop is repeatedly executed while there remain edges in G from a to a node from which a search has not been started; the chosen node b is then one such node.

It is a very substantial exercise to verify the postcondition of repeated depth-first search since, in order to do so, additional invariant properties must be identified and verified. We have identified 16 different conjuncts in the invariant of depth-first search. Given that there are 5 components in its implementation (two assignments, one test, one choice and one recursive call), this means that there are at least 64 (sixteen times four) verification conditions that must be checked in order to verify repeated depth-first search: the recursive call can be ignored "by induction" but the repeated invocation of depth-first search also incurs additional verification conditions. Although many of these verification conditions are straight-forward, and might be taken for granted in an informal account of the algorithm, there is still much to be done. For more information, including a detailed comparison with [9], see [6].)

Suppose s and f are the start and finish timestamps computed by a repeated depth-first search of the graph G as detailed above. Suppose φ is the delegate function on G according to the timestamp f. (Recall that, as remarked immediately following its definition in section 3.1, the function φ is the function computed by the repeated search of $G^{\cup}$ in the second stage of the Kosaraju-Sharir algorithm.)

From theorem 26, we know that

$$\mathsf{equiv}.G \subseteq \varphi^{\cup} \circ \varphi \quad .$$

It remains to show that

$$\varphi^{\cup} \circ \varphi \subseteq \mathsf{equiv}.G \quad .$$

We do this by showing that $\varphi \subseteq \mathsf{equiv}.G$. That is, we show that the delegate of a node according to f is strongly connected to the node. The key is to use induction, the main difficulty being to identify a suitable induction hypothesis. This is done in the following lemma. Its proof combines two properties of delegates: (i) for each node, there is a path to its delegate on which all nodes have the same delegate and (ii) the delegate has the largest f-value.

Lemma 30

$$\varphi \;\subseteq\; \langle \mu X \;::\; f^{\cup} \circ \geq \circ f \cap (I \cup X \circ G^{\cup}) \rangle \quad .$$

Proof

$$\begin{array}{ll} & \varphi \\ = & \quad \{ \quad \text{lemma 25} \quad \} \\ & \langle \mu X :: \varphi \cap (I \cup X \circ G^{\cup}) \rangle \\ \subseteq & \quad \{ \quad \text{theorem 26 (specifically, } \varphi \subseteq f^{\cup} \circ \geq \circ f \text{)} \\ & \qquad \text{and monotonicity} \quad \} \\ & \langle \mu X \;::\; f^{\cup} \circ \geq \circ f \cap (I \cup X \circ G^{\cup}) \rangle \quad . \end{array}$$

□

Lemma 30 enables us to use fixed-point induction to establish a key lemma:

Lemma 31

$$\varphi \;\subseteq\; s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f \quad .$$

Proof

$$\begin{array}{ll} & \varphi \subseteq s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f \\ \Leftarrow & \quad \{ \quad \text{lemma 30} \quad \} \\ & \langle \mu X \;::\; f^{\cup} \circ \geq \circ f \cap (I \cup X \circ G^{\cup}) \rangle \;\subseteq\; s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f \\ \Leftarrow & \quad \{ \quad \text{fixed-point induction} \quad \} \\ & f^{\cup} \circ \geq \circ f \cap (I \cup (s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f) \circ G^{\cup}) \;\subseteq\; s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f \\ \Leftarrow & \quad \{ \quad \text{distributivity and } [\; R \cup S = R \cup (\neg R \cap S) \;] \\ & \qquad \text{with } R,S \;:=\; I \,,\, (s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f) \circ G^{\cup} \quad \} \\ & \quad f^{\cup} \circ \geq \circ f \cap I \;\subseteq\; s^{\cup} \circ \leq \circ s \\ & \wedge \quad f^{\cup} \circ \geq \circ f \cap \neg I \cap (s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f) \circ G^{\cup} \;\subseteq\; s^{\cup} \circ \leq \circ s \\ = & \quad \{ \quad \leq \text{ is reflexive and } s \text{ is total, so } I \subseteq s^{\cup} \circ \leq \circ s \\ & \qquad f \text{ is injective, so } f^{\cup} \circ \geq \circ f \cap \neg I = f^{\cup} \circ > \circ f \quad \} \\ & f^{\cup} \circ > \circ f \cap (s^{\cup} \circ \leq \circ s \cap f^{\cup} \circ \geq \circ f) \circ G^{\cup} \;\subseteq\; s^{\cup} \circ \leq \circ s \quad . \end{array}$$

We continue with the left-hand side of the inclusion.

$$
\begin{array}{ll}
 & f^{\cup} \circ > \circ f \ \cap \ (s^{\cup} \circ \leq \circ s \ \cap \ f^{\cup} \circ \geq \circ f) \circ G^{\cup} \\
\subseteq & \quad \{ \quad \text{assumption (29) and converse: } G^{\cup} \subseteq s^{\cup} \circ \leq \circ f \quad \} \\
 & f^{\cup} \circ > \circ f \ \cap \ (s^{\cup} \circ \leq \circ s \ \cap \ f^{\cup} \circ \geq \circ f) \circ (s^{\cup} \circ \leq \circ f) \\
\subseteq & \quad \{ \quad [\, R \cap S \subseteq R \,] \text{ with } R,S \ := \ s^{\cup} \circ \leq \circ s \ , \ f^{\cup} \circ \geq \circ f \\
 & \qquad \text{and monotonicity} \quad \} \\
 & f^{\cup} \circ > \circ f \ \cap \ s^{\cup} \circ \leq \circ s \circ s^{\cup} \circ \leq \circ f \\
\subseteq & \quad \{ \quad s \text{ is functional, so } s \circ s^{\cup} \subseteq I \ , \ \leq \text{ is transitive} \quad \} \\
 & f^{\cup} \circ > \circ f \ \cap \ s^{\cup} \circ \leq \circ f \\
= & \quad \{ \quad \text{assumption : (28), i.e. (taking converse and complements)} \\
 & \qquad s^{\cup} \circ \leq \circ f \ = \ s^{\cup} \circ \leq \circ s \ \cup \ f^{\cup} \circ \leq \circ f \quad \} \\
 & f^{\cup} \circ > \circ f \ \cap \ (s^{\cup} \circ \leq \circ s \ \cup \ f^{\cup} \circ \leq \circ f) \\
= & \quad \{ \quad f^{\cup} \circ > \circ f \ \cap \ f^{\cup} \circ \leq \circ f \ = \ \perp\!\!\!\perp \quad \} \\
 & f^{\cup} \circ > \circ f \ \cap \ s^{\cup} \circ \leq \circ s \\
\subseteq & \quad \{ \quad \text{monotonicity} \quad \} \\
 & s^{\cup} \circ \leq \circ s \quad .
\end{array}
$$

Combining the two calculations, the proof is complete.
□

Now we can proceed to show that every node is strongly connected to its delegate.

Lemma 32 Suppose φ is the delegate function on G according to the timestamp f. Then

$$\varphi \subseteq \mathsf{equiv}.G \quad .$$

Proof

$$
\begin{array}{ll}
 & \varphi \subseteq \mathsf{equiv}.G \\
= & \quad \{ \quad \text{definition of } \mathsf{equiv}.G \text{, distributivity} \quad \} \\
 & \varphi \subseteq G^{*} \wedge \varphi \subseteq (G^{*})^{\cup} \\
= & \quad \{ \quad \text{by definition of delegate (see theorem 26), } \varphi \subseteq (G^{*})^{\cup} \quad \} \\
 & \varphi \subseteq G^{*} \\
\Leftarrow & \quad \{ \quad (27) \text{ is a postcondition of repeated depth-first search} \quad \} \\
 & \varphi \ \subseteq \ s^{\cup} \circ \leq \circ s \ \cap \ f^{\cup} \circ \geq \circ f \\
\Leftarrow & \quad \{ \quad \text{lemma 31} \quad \} \\
 & \mathsf{true} \quad .
\end{array}
$$

□

Theorem 33 Suppose f is the finish timestamp computed by a repeated depth-first search of a graph G. Then the delegate function on G according to f is a representative function for strongly connected components of G. That is, if φ denotes the delegate function,

$$\varphi^{\cup} \circ \varphi \;=\; \mathsf{equiv}.G \quad .$$

Proof

$$\begin{array}{ll} & \varphi^{\cup} \circ \varphi \;=\; \mathsf{equiv}.G \\ = & \quad \{ \quad \text{anti-symmetry} \quad \} \\ & \mathsf{equiv}.G \subseteq \varphi^{\cup} \circ \varphi \;\;\wedge\;\; \varphi^{\cup} \circ \varphi \subseteq \mathsf{equiv}.G \\ \Leftarrow & \quad \{ \quad \text{theorem 26, lemma 32} \quad \} \\ & \mathsf{true} \;\;\wedge\;\; (\mathsf{equiv}.G)^{\cup} \circ \mathsf{equiv}.G \subseteq \mathsf{equiv}.G \\ = & \quad \{ \quad \mathsf{equiv}.G \text{ is symmetric and transitive} \quad \} \\ & \mathsf{true} \quad . \end{array}$$

□

5 Conclusion

In one sense, this paper offers no new results. Graph-searching algorithms have been studied extensively for decades and have long been a standard part of the undergraduate curriculum in computing science. The driving force behind this work has been to disentangle different elements of the correctness of the two-stage algorithm for determining the strongly connected components of a graph: our goal has been to clearly distinguish properties peculiar to depth-first search that are vital to the first stage of the algorithm from properties of repeated graph search that are exploited in its second stage. This is important because an algorithm to determine strongly connected components of a graph does not operate in a vacuum: the information that is gleaned is used to inform other computations. For example, Sharir [16] shows how to combine his algorithm with an iterative algorithm for data-flow analysis.

The primary contribution of the paper is, however, to show how the choice of an appropriate algebraic framework enables precise, concise calculation of algorithmic properties of graphs. Although with respect to graph algorithms (as opposed to relation algebra in general) the distinction between "point-free" and "pointwise" calculations has only been made relatively recently, this was the driving force behind the author's work on applying regular algebra to path-finding problems [2, 3].

The difference between point-free and pointwise calculations can be appreciated by noting that nowhere in our calculations is there an existential quantification or a nested universal quantification. Typical accounts of depth-first search make abundant use of such quantifications; the resulting formal statements are long and unwieldy, and calculations become (in our view) much harder to check: compare, for example, the concision of the three assumptions (27), (28) and (29) with the three assumptions made by King and Launchbury [15].

Of course, our discussion of the two-stage algorithm is incomplete because we have not formally established the properties of the first (depth-first search) stage that we assume hold in the second stage. (The same criticism is true of [15].) This we have done in [6]. Although the calculations are long — primarily because there is a large number of verification conditions to be checked — we expect that they would be substantially shorter and easier to check than formal pointwise justifications of the properties of depth-first search.

Acknowledgements Many thanks to the referees for their careful and detailed critique of the submitted paper. Thanks also for pointing out that explicit mention of the "forefather" function, studied in detail in [9], has been elided in [10].

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1982.
2. R.C. Backhouse. *Closure algorithms and the star-height problem of regular languages*. PhD thesis, University of London, 1975. Available at `https://spiral.imperial.ac.uk/bitstream/10044/1/22243/2/Backhouse-RC-1976-PhD-Thesis.pdf`.
3. R.C. Backhouse and B.A. Carré. Regular algebra applied to path-finding problems. *Journal of the Institute of Mathematics and its Applications*, 15:161–186, 1975.
4. R.C. Backhouse and J. van der Woude. Demonic operators and monotype factors. *Mathematical Structures in Computer Science*, 3(4):417–433, December 1993.
5. Roland Backhouse. Regular algebra applied to language problems. *Journal of Logic and Algebraic Programming*, 66:71–111, 2006.
6. Roland Backhouse, Henk Doornbos, Roland Glück, and Jaap van der Woude. Algorithmic graph theory: An exercise in point-free reasoning. `http://www.cs.nott.ac.uk/~pasrb2/MPC/BasicGraphTheory.pdf`, Also available online at ResearchGate, 2019.
7. Roland C. Backhouse, J.P.H.W. van den Eijnde, and A.J.M. van Gasteren. Calculating path algorithms. *Science of Computer Programming*, 22(1–2):3–19, 1994.
8. J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.
9. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Electrical Engineering and Computer Science Series, MIT Press, 1990.
10. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd edition*. MIT Electrical Engineering and Computer Science Series, MIT Press, 2009.

11. Henk Doornbos and Roland Backhouse. Reductivity. *Science of Computer Programming*, 26(1–3):217–236, 1996.
12. P.J. Freyd and A. Ščedrov. *Categories, Allegories*. North-Holland, 1990.
13. Roland Glück. Algebraic investigation of connected components. In P. Höfner, D. Pous, and G. Struth, editors, *Relational and Algebraic Methods in Computer Science – 16th International Conference, RAMiCS 2017*, volume 10226 of *Lecture Notes in Computer Science*, pages 109–126. Springer, May 15–18 2017.
14. Paul Hoogendijk and Roland C. Backhouse. Relational programming laws in the tree, list, bag, set hierarchy. *Science of Computer Programming*, 22(1–2):67–105, 1994.
15. David J. King and John Launchbury. Structuring depth-first search algorithms in Haskell. In *POPL '95. Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programmming Languages*, pages 344–354, 1995.
16. M. Sharir. A strong-connectivity algorithm and its application in data flow analysis. *Computers and Mathematics with Applications*, 7(1):67–72, 1981.
17. Robert Endre Tarjan. Depth first search and linear graph algorithms. *SIAM J. Computing*, pages 146–160, 1972.