Proof of an Abstract Algorithm for Graph Reachability

Jayadev Misra

May 7, 2019

Abstract

We present an abstract non-deterministic algorithm for identifying nodes of a directed graph that are reachable from a specific node. The algorithm encompasses all known algorithms for this problem, including breadth-first and depth-first search. Its correctness, therefore, implies that all such algorithms are correct.

Keywords: Graph algorithms, reachability in graphs, breadth-first search, depth-first search, proof of algorithms.

1 Reachability in graphs

We consider the classic problem of reachability in a graph. Given is a finite directed graph with a designated *root* node. It is required to identify all nodes that are reachable via a directed path from *root*.

There are many well-known algorithms for this problem, including breadth-first and depth-first search. The goal of this paper is to design and prove an algorithm that includes the known algorithms as special cases. The proposed algorithm is non-deterministic. We obtain different specific algorithms by limiting the non-deterministic options in its execution. In particular, we employ a set *vroot* in our algorithm to which nodes are added and from which nodes are deleted during execution. We do not specify the data structure for *vroot*. Implementing *vroot* as a queue, so that additions and deletions are performed at different ends of the list, yields the breadth-first search algorithm, whereas implementing *vroot* as a stack yields depth-first search.

We give a concise proof of correctness, which is not obvious. We also show a more non-deterministic version of the algorithm.

Formal definitions We include the material below for completeness.

Convention: We introduce a number of functions, *succ*, R_n, R, from nodes to sets of nodes. Extend the definitions so that each function has a *set* of nodes as argument and the the function value is the union of the results for individual nodes in the argument. That is, for S a set of nodes and x a node, $R(S) = (\cup x : x \in S : R(\{x\}))$. Therefore, for sets of nodes S and T, $R(S \cup T) = R(S) \cup R(T)$, and if $S \subseteq T$ then $R(S) \subseteq R(T)$.

For the given graph, $succ(\{x\})$ is the set of nodes to which node x has outgoing edges, $R(\{x\})$ the set of nodes reachable from x and $R_i(\{x\})$ the set of nodes reachable from x via a path (not necessarily a simple path) of length i, $i \geq 0$.

Formal definitions of R_i and R are given below, where i is universally quantified over naturals and x over nodes.

1. $R_i(\{\}) = \{\}$.
2. $R_0(\{x\}) = \{x\}$.
3. $R_{i+1}(\{x\}) = succ(R_i(\{x\}))$.
4. $R(\{x\}) = (\cup n : n \geq 0 : R_n(\{x\}))$.

Properties of reachability relation R

The following properties of R can be proved. Below, x is a node and S a set of nodes.

- P1. $R(\{\}) = \{\}$.
- P2. $R(S) = S \cup succ(R(S)) = S \cup R(succ(S))$.
- P3. $S \subseteq R(S)$ and $succ(R(S)) \subseteq R(S)$, from (P2).
- P4. $R(R(S)) = R(S)$.
- P5. $(succ(S) \subseteq S) \equiv (S = R(S))$

Properties P1 through P4 are proved easily from the definition. We prove property P5.

First, prove $(S = R(S)) \Rightarrow (succ(S) \subseteq S)$:

$$
\begin{array}{ll}
 & succ(S) \\
= & \{\text{From the antecedent, } S = R(S). \text{ Replace } S \text{ by } R(S).\} \\
 & succ(R(S)) \\
\subseteq & \{\text{from P2, } succ(R(S)) \subseteq R(S)\} \\
 & R(S) \\
= & \{\text{antecedent}\} \\
 & S
\end{array}
$$

Next, prove $(succ(S) \subseteq S) \Rightarrow (S = R(S))$: From P3, $S \subseteq R(S)$. So, it suffices to prove $(succ(S) \subseteq S) \Rightarrow (R(S) \subseteq S)$. We prove $R_n(S) \subseteq S$ for all n, $n \geq 0$, so $R(S) = (\cup n : n \geq 0 : R_n(S)) \subseteq S$.

The proof of $R_n(S) \subseteq S$ for all n is by induction on n. For $n = 0$, $R_0(S) = S$. Assume inductively that the result holds for some n, $n \geq 0$.

$$
\begin{array}{ll}
 & R_{n+1}(S) \\
= & \{\text{definition}\} \\
 & succ(R_n(S)) \\
\subseteq & \{\text{Induction: } R_n(S) \subseteq S\} \\
 & succ(S) \\
\subseteq & \{\text{antecedent: } succ(S) \subseteq S\} \\
 & S
\end{array}
$$

2 An abstract program for reachability

The strategy for computing the reachable nodes from *root* is to maintain two sets of nodes, *marked* and *vroot*. The nodes in *marked* have already been identified as being reachable from *root*. The nodes in *vroot* are analogous to *root* in that the remaining reachable nodes from *root* are those reachable from some node in *vroot*; i.e., $marked \cup R(vroot) = R(\{root\})$ will be a property of the program.

Initially, *marked* is empty and *vroot* contains only *root*. In each step, if *vroot* is non-empty, an arbitrary node x is chosen from *vroot*. if x is not in *marked* it is added to *marked* and all its successors are added to *vroot*, otherwise (x is in *marked*) x is removed from *vroot*. The program terminates when *vroot* is empty. Then, from $marked \cup R(vroot) = R(\{root\})$ we get $marked = R(\{root\})$, i.e., *marked* contains exactly the nodes reachable from *root*.

The program is given below. Labels **C1**, **C2**, **C3** and **C4** have been appended to commands to simplify subsequent explanation. Command **C4** is merely a *skip*. It has been added to streamline the proof.

$\{true\}$
C1:: $marked,\ vroot := \{\},\ \{root\}$ **;**
while $vroot \neq \{\}$ **do**
  choose x from $vroot$ **;**
  if $x \notin marked$ **then**
    C2:: $marked,\ vroot := marked \cup \{x\},\ vroot \cup succ(\{x\})$
  else **C3**:: $vroot := vroot - \{x\}$
  endif
enddo
$\{vroot = \{\}\}$
C4:: *skip*
$\{marked = R(\{root\})\}$

Note: It is customary to remove x from *vroot*, in command **C2**, as soon as its successors are added to *vroot*. We have not done so in order to get a more non-deterministic program. If required, x may be removed in the next step by executing **C3** with the appropriate choice of x.

Usefulness of the Absract program The given program admits of a number of possible implementations. In fact, an even more non-deterministic program, given in Section 4, has even more options for implementations. In the program here, suppose an execution of **C2** with some x is immediately followed by **C3** to remove x from *vroot*. Now, suppose *vroot* is implemented as a stack so that x is always chosen and removed from the top and nodes in $succ(\{x\})$ are added at the top. Then, the program implements depth-first search. And, if *vroot* is implemented as a queue so that x is always chosen and removed from the head and nodes in $succ(\{x\})$ are added at the tail, then the program implements breadth-first search.

3 Correctness

3.1 Partial Correctness

Invariant A first choice for invariant I is $marked \cup R(vroot) = R(\{root\})$. This predicate holds initially. Additionally, when the loop terminates, from $vroot = \{\}$ conclude that $marked = R(\{root\})$.

Unfortunately, $marked \cup R(vroot) = R(\{root\})$ can not be inductively proved to be invariant. This is because for **C3** it is not possible to show that all nodes in $R(\{x\})$ are still reachable from $vroot$ after removing x from $vroot$.

We propose invariant I to be a conjunction of four predicates.

$$
\begin{aligned}
& \{root\} \subseteq marked \cup vroot \\
\wedge\ & succ(marked) \subseteq marked \cup vroot \\
\wedge\ & vroot \subseteq R(\{root\}) \\
\wedge\ & marked \subseteq R(\{root\})
\end{aligned}
$$

It is possible to prove that each conjunct is an invariant, in the given order. We rewrite the invariant as follows and prove it formally.

$$I{::}\ \{root\} \cup succ(marked) \subseteq marked \cup vroot \subseteq R(\{root\})$$

Proofs of the Verification Conditions

C1. Setting $marked$ to $\{\}$ and $vroot$ to $\{root\}$ in I, we need to show $\{root\} \subseteq \{root\} \subseteq R(\{root\})$. The first inequality holds trivially, and the second from property (P3) of *reachable*.

C2. Show

$$
\begin{aligned}
& \{root\} \cup succ(marked) \subseteq marked \cup vroot \subseteq R(\{root\}) \\
& \wedge\ x \in vroot \wedge x \notin marked \\
\Rightarrow\ & \\
& \{root\} \cup succ(marked \cup \{x\}) \subseteq marked \cup \{x\} \cup vroot \cup succ(\{x\}) \\
& \subseteq R(\{root\}).
\end{aligned}
$$

We prove the conclusion in two parts corresponding to the two inequalities.

1. $\{root\} \cup succ(marked \cup \{x\}) \subseteq marked \cup \{x\} \cup vroot \cup succ(\{x\})$:

$$
\begin{aligned}
& \{root\} \cup succ(marked \cup \{x\}) \\
\subseteq\ & \{\text{expand } succ\} \\
& \{root\} \cup succ(marked) \cup succ(\{x\}) \\
\subseteq\ & \{\text{antecedent: } \{root\} \cup succ(marked) \subseteq marked \cup vroot\} \\
& marked \cup vroot \cup succ(\{x\}) \\
\subseteq\ & \{\text{set theory}\} \\
& marked \cup \{x\} \cup vroot \cup succ(\{x\})
\end{aligned}
$$

2. $marked \cup \{x\} \cup vroot \cup succ(\{x\}) \subseteq R(\{root\})$:

$$
\begin{aligned}
& marked \cup \{x\} \cup vroot \cup succ(\{x\}) \\
\subseteq\ & \{\text{Property P3: } succ(\{x\} \subseteq R(succ(\{x\})\} \\
& marked \cup vroot \cup \{x\} \cup R(succ(\{x\})) \\
=\ & \{\text{Property P2: } \{x\} \cup R(succ(\{x\})) = R(\{x\})\} \\
& marked \cup vroot \cup R(\{x\}) \\
\subseteq\ & \{\text{antecedent: } marked \cup vroot \subseteq R(\{root\})\ \}
\end{aligned}
$$

$$
\begin{array}{cl}
 & R(\{root\}) \cup R(\{x\}) \\
\subseteq & \{\{\text{from } x \in vroot,\ R(\{x\}) \subseteq R(vroot)\} \\
 & R(\{root\}) \cup R(vroot) \\
\subseteq & \{\text{from } vroot \subseteq R(\{root\}),\ R(vroot) \subseteq R(R(\{root\}))\} \\
 & R(\{root\}) \cup R(R(\{root\})) \\
= & \{\text{Property P4: } R(\{root\}) = R(R(\{root\}))\} \\
 & R(\{root\})
\end{array}
$$

C3. I is preserved because $marked$ and $marked \cup vroot$ are unchanged by **C3**, given $x \in vroot \wedge x \in marked$ as precondition.

C4. The verification condition is

$\{root\} \cup succ(marked) \subseteq marked \cup vroot \subseteq R(\{root\}) \wedge vroot = \{\}$
$\Rightarrow marked = R(\{root\})$.

$$
\begin{array}{cl}
 & R(\{root\}) \\
\subseteq & \{\text{antecedent: } \{root\} \subseteq marked.\ \text{So, } R(\{root\}) \subseteq R(marked)\} \\
 & R(marked) \\
= & \{\text{from } vroot = \{\},\ succ(marked) \subseteq marked.\ \text{Use Property P5}\} \\
 & marked \\
\subseteq & \{\text{antecedent: } marked \subseteq R(\{root\})\} \\
 & R(\{root\})
\end{array}
$$

Therefore, $R(\{root\}) \subseteq marked \subseteq R(\{root\})$, so $marked = R(\{root\})$.

Note: The reader may show that $marked \cup vroot = R(\{root\})$ follows from invariant I.

3.2 Termination

Let $unmarked$ be the set of reachable nodes from root that are not in $marked$, i.e., $unmarked = R(\{root\}) - marked$. Command **C2** adds node x to $marked$, effectively removing it from $unmarked$, thus reducing the set $unmarked$; observe that $x \in R(\{root\})$ from I. And, command **C3** removes a marked node from $vroot$, thus preserving $marked$ and, hence, $unmarked$, while reducing $vroot$. Therefore, each iteration decreases the metric $(unmarked, vroot)$ lexicographically, where the sets are ordered by subset ordering. Each set is finite, so this metric is well-founded, guaranteeing termination of the algorithm.

4 Introducing Further Non-determinism

The given program has exactly one point of non-determinism, in choosing a node from $vroot$. Its execution from then is deterministic. We add further non-determinism as follows: choose any node x of the graph, then choose the action to add x to $marked$ or remove x from $vroot$. If $x \notin vroot$ or $x \in marked$ and the "add" action has been chosen, do nothing. Analaogously, if $x \notin vroot$ or $x \notin marked$ and the "remove" action has been chosen, do nothing. Otherwise, do exactly as in the previous program.

The following program is written in the style of UNITY from Chandy and Misra[1]. A UNITY program has an initialization part followed by a set of commands. The program execution starts in a state that satisfes the initial condition. In each step an arbitrary command is chosen for execution and the execution runs forever under the requirement that every command be chosen eventually. Command execution in a step has no effect if the chosen command's guard is *false*, then the step is a *stutter*, otherwise the command body is executed. A state in which further execution does not cause any change is called a *fixed point*; the set of all fixed points is denoted by the predicate FP. Program termination is equivalent to reaching a fixed point. For the given program the correctness requirement is: (1) eventually a fixed point is reached, and (2) at any fixed point $marked = R(\{root\})$. It will still be necessary to prove an invariant so that the invariant and FP imply the desired condition in (2).

The program executes the same actions as before interspersed, possibly, with more stutter steps.

4.1 Program

$\{true\}$
U1:: **initially** $marked,\ vroot = \{\},\ \{root\}$
$(\forall x : x$ a node in the graph :
  U2:: $x \in vroot \wedge x \notin marked \quad \rightarrow$
    $marked,\ vroot := marked \cup \{x\},\ vroot \cup succ(\{x\})$
 [] **U3**:: $x \in vroot \wedge x \in marked \quad \rightarrow$
    $vroot := vroot - \{x\}$
)
$\{marked = R(\{root\})\}$

4.2 Correctness: Safety and Termination

Proof of partial correctness uses the same invariant I and is identical in all respects to the proof of Section 3.1. Proof of progress that the program eventually reaches a fixed point, is more involved because of the possibility of infinite stutter.

The FP for this program, when both guards are *false*, is

$(\forall x : x$ a node in the graph :
  $\neg(x \in vroot \wedge x \notin marked)$
 $\wedge\ \neg(x \in vroot \wedge x \in marked)$
)

which simplifies to $(\forall x : x$ a node in the graph : $x \notin vroot)$ or $vroot = \{\}$.

We show below that eventually $vroot = \{\}$. No more state changes are posssible beyond that point, and as in Section 3.1, $I \wedge vroot = \{\} \Rightarrow marked = R(\{root\})$.

UNITY Temporal Operators The following proof is written in UNITY logic using the temporal operators **en** and $\mapsto$ that esatblish a given predicate eventually. Here, "eventually" means after a finite number of steps, so the required predicate holds now or after one or more steps. For a full treatment of UNITY logic see Chandy and Misra[1] and

Misra [2]. Here, we explain just enough to prove that a fixed point is reached eventually.

For state predicates p and q, p **en** q asserts that (1) Safety: once predicate p is true it will continue to hold until q holds, and (2) Progress: eventually q is established by execution of some command. For (1) show for *every* command α, $\{p \wedge \neg q\}\ \alpha\ \{p \vee q\}$. For (2) show *some* command β for which $\{p \wedge \neg q\}\ \beta\ \{q\}$ because β will be executed eventually and establish q if it has not been true already.

Operator $\mapsto$ is a generalization of **en** where $p \mapsto q$ asserts that once p is true q will eventually be true. For this paper, the only facts about $\mapsto$ that we need are: (1) if p **en** q then $p \mapsto q$, (2) if $p \mapsto q$ and $p' \mapsto q'$ then $p \vee p' \mapsto q \vee q'$, and (3) (induction) given a metric m and a well-founded order $\prec$ over its domain, if for all possible values M in the domain of m $p \wedge m = M \mapsto m \prec M$ then $true \mapsto \neg p$.

Proofs of Elementary Progress properties We sketch the proof of two **en** properties informally because the details are easy to fill in. Below, $\prec$ denotes lexicographic order over pairs of sets, i.e., $(A, B) \preceq (A', B')$ means that $A \subseteq A'$ or $A = A' \wedge B \subseteq B'$, and $(A, B) \prec (A', B')$ is $(A, B) \preceq (A', B') \wedge (A, B) \neq (A', B')$.

We show for every node x in the graph (where S and T are arbitrary sets of nodes):

E1. $(unmarked, vroot) = (S, T) \wedge x \in vroot \wedge x \notin marked$ **en** $(unmarked, vroot) \prec (S, T)$:

Safety part for command **U3** follows easily because the command does not execute in the given state, so it preserves both predicates in the property. The progress part for command **U2** also implies the safety part. The progress proof is identical to that in Section 3.2.

E2. $(unmarked, vroot) = (S, T) \wedge x \in vroot \wedge x \in marked$ **en** $(unmarked, vroot) \prec (S, T)$:

Safety part for command **U2** follows because the command does not execute in the given state, so it preserves both predicates in the property. Execution of command **U3** establishes $unmarked = S \wedge vroot \subset T$ because the command does not modify $marked$, hence $unmarked$, and it reduces $vroot$ by removing x from it.

Proof of Termination We show $true \mapsto FP$; since $FP \equiv vroot = \{\}$, we show $true \mapsto vroot = \{\}$. Below, x is any node of the graph.

$$
\begin{array}{l}
(unmarked, vroot) = (S,T) \wedge x \in vroot \wedge x \notin marked \\
\qquad \mapsto \; (unmarked, vroot) \prec (S,T) \text{ ,from [E1]} \\
(unmarked, vroot) = (S,T) \wedge x \in vroot \wedge x \in marked \\
\qquad \mapsto \; (unmarked, vroot) \prec (S,T) \text{ ,from [E2]} \\
(unmarked, vroot) = (S,T) \wedge x \in vroot \\
\qquad \mapsto \; (unmarked, vroot) \prec (S,T) \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{, disjunction of above two} \\
(unmarked, vroot) = (S,T) \wedge (\exists x :: x \in vroot) \\
\qquad \mapsto \; (unmarked, vroot) \prec (S,T) \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{, disjunction over all } x \\
true \mapsto \neg(\exists x :: x \in vroot) \text{, induction} \\
true \mapsto vroot = \{\} \text{, from } \neg(\exists x :: x \in vroot) \equiv vroot = \{\}
\end{array}
$$

Acknowledgement Thanks to Leslie Lamport for discussions.

References

[1] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[2] Jayadev Misra. *A Discipline of Multiprogramming*. Monographs in Computer Science. Springer-Verlag New York Inc., New York, 2001.