## module *Reachable*

This module specifies an algorithm for computing the set of nodes in a directed graph that are reachable from a given node called *Root*. The algorithm is due to *Jayadev Misra*. It is, to my knowledge, a new variant of a fairly obvious breadth-first search for reachable nodes. I find this algorithm interesting because it is easier to implement using multiple processors than the obvious algorithm. Module *ParReach* describes such an implementation. You may want to read it after reading this module.

Module *ReachableProofs* contains a TLA+ proof of the algorithm's safety property–that is, partial correctness, which means that if the algorithm terminates then it produces the correct answer. That proof has been checked by *TLAPS*, the TLA+ proof system. The proof is based on ideas from an informal correctness proof by *Misra*.

In this module, reachability is expressed in terms of the operator *ReachableFrom*, where *ReachableFrom*($S$) is the set of nodes reachable from the nodes in the set $S$ of nodes. This operator is defined in module *Reachability*. That module describes a directed graph in terms of the constants *Nodes* and *Succ*, where *Nodes* is the set of nodes and *Succ* is a function with domain *Nodes* such that $Succ[m]$ is the set of all nodes $n$ such that there is an edge from $m$ to $n$. If you are not familiar with directed graphs, you should read at least the opening comments in module *Reachability*.

**extends** *Reachability*, *Integers*, *FiniteSets*

**constant** *Root*

**assume** $RootAssump \triangleq Root \in Nodes$

Reachable is defined to be the set of notes reachable from *Root*. The purpose of the algorithm is to compute *Reachable*.

$Reachable \triangleq ReachableFrom(\{Root\})$

---

***********************************************************************

The obvious algorithm for computing $Reachable(\{Root\})$ is as follows. There are two variables which, following *Misra*, we call *marked* and *vroot*. Each variable holds a set of nodes that are reachable from Root. Initially, $marked = \{\}$ and $vroot = \{Root\}$. While *vroot* is non-empty, the obvious algorithm removed an arbitrary node $v$ from *vroot*, adds $v$ to *marked* , and adds to *vroot* all nodes in $Succ[v]$ that are not in *marked* . The algorithm terminates when *vroot* is empty, which will eventually be the case if and only if $Reachable(\{Root\})$ is a finite set. When it terminates, *marked* equals $Reachable(\{Root\})$.

In the obvious algorithm, *marked* and *vroot* are always disjoint sets of nodes. *Misra*'s variant differs in that *marked* and *vroot* are not necessarily disjoint. While *vroot* is nonempty, it chooses an arbitrary node and does the following:

if $v$ is not in in *marked* then it performs the same action as the obvious algorithm except:
  (1) it doesn't remove $v$ from *vroot*, and
  (2) it adds all nodes in $Succ[v]$ to *vroot*, not just the ones not in *marked* .
 else it removes $v$ from *vroot*

Here is the algorithm's *PlusCal* code.

```
--fair algorithm Reachable {
  variables marked = {}; vroot = {Root} ;
  { a: while ( vroot ≠ {} )
        { with ( v ∈ vroot )
           { if ( v ∉ marked )
                { marked := marked ∪ {v} ;
```

$vroot \quad := vroot \cup Succ[v]$ g
**else** f $vroot := vroot \setminus \{v\}$ g
g
g
g
g

Here is the TLA+ translation of the *PlusCal* code.
***************************************************************************

BEGIN TRANSLATION

**variables** *marked*; *vroot*; *pc*

$Reachable \triangleq ReachableFrom(marked)$

$vars \triangleq \langle marked; vroot; pc \rangle$

$Init \triangleq$ Global variables
$\quad \land marked = \{\}$
$\quad \land vroot = \{Root\}$
$\quad \land pc = \text{“a”}$

$a \triangleq \land pc = \text{“a”}$
$\quad \land$ **if** $vroot \neq \{\}$
$\qquad$ **then** $\land \exists v \in vroot :$
$\qquad\qquad$ **if** $v \notin marked$
$\qquad\qquad\quad$ **then** $\land marked' = (marked \cup \{v\})$
$\qquad\qquad\qquad \land vroot' = (vroot \cup Succ[v])$
$\qquad\qquad\quad$ **else** $\land vroot' = vroot \setminus \{v\}$
$\qquad\qquad\qquad \land$ **unchanged** $marked$
$\qquad\quad \land pc' = \text{“a”}$
$\qquad$ **else** $\land pc' = \text{“Done”}$
$\qquad\quad \land$ **unchanged** $\langle marked; vroot \rangle$

$Next \triangleq a$
$\quad \lor$ Disjunct to prevent deadlock on termination
$\quad (pc = \text{“Done”} \land \textbf{unchanged } vars)$

$Spec \triangleq \land Init \land \Box[Next]_{vars}$
$\quad \land \mathrm{WF}_{vars}(Next)$

$Termination \triangleq \Diamond(pc = \text{“Done”})$

END TRANSLATION

---

Partial correctness is based on the invariance of the following four state predicates. I have sketched very informal proofs of their invariance, as well of proofs of the the two theorems that assert correctness of the algorithm. The module *ReachableProofs* contains rigorous, *TLAPS* checked TLA+ proofs of all except the last theorem. The last theorem asserts termination, which is a liveness property, and *TLAPS* is not yet capable of proving liveness properties.

$$\begin{aligned} TypeOK \;\triangleq\; & \land\ marked \in \textsf{subset}\ Nodes \\ & \land\ vroot \in \textsf{subset}\ Nodes \\ & \land\ pc \in \{\text{"a"}; \text{"Done"}\} \\ & \land\ (pc = \text{"Done"}) \Rightarrow (vroot = \{\}) \end{aligned}$$

The invariance of *TypeOK* is obvious. (I decided to make the obvious fact that $pc$ equals \Done" only if $vroot$ is empty part of the type-correctness invariant.)

$$\begin{aligned} Inv1 \;\triangleq\; & \land\ TypeOK \\ & \land\ \forall\, n \in marked : Succ[n] \subseteq (marked \cup vroot) \end{aligned}$$

The second conjunct of $Inv1$ is invariant because each element of $Succ[n]$ is added to $vroot$ when $n$ is added to $marked$ , and it remains in $vroot$ at least until it's added to $marked$ . I made *TypeOK* a conjunct of $Inv1$ to make $Inv1$ an inductive invariant, which made the TLA+ proof of its invariance a tiny bit easier to read.

$$Inv2 \;\triangleq\; (marked \cup ReachableFrom(vroot)) = ReachableFrom(marked \cup vroot)$$

Since $ReachableFrom(marked \cup vroot)$ is the union of $ReachableFrom(marked)$ and $ReachableFrom(vroot)$, to prove that $Inv2$ is invariant we must show $ReachableFrom(marked)$ is a subset of $marked \cup ReachabledFrom(vroot)$. For this, we assume that $m$ is in $ReachableFrom(marked)$ and show that it either is in $marked$ or is reachable from a node in $vroot$.

Since $m$ is in $ReachableFrom(marked)$, there is a path with nodes $p\_1$, $p\_2$, $: : :$ , $p\_j$ such that $p\_1$ is in $marked$ and $p\_j = m$. If all the $p\_i$ are in $marked$ , then $m$ is in $marked$ and we're done. Otherwise, choose $i$ such that $p\_1$, $: : :$ , $p\_i$ are in $marked$ , but $p\_(i+1)$ isn't in $marked$ . Then $p\_(i+1)$ is in $succ[p\_i]$, which by $Inv1$ implies that it's in $marked \cup vroot$. Since it isn't in $marked$ , it must be in $vroot$. The path with nodes $p\_(i+1)$, $: : :$ , $p\_j$ shows that $p\_j$, which equals $m$, is in $ReachableFrom(vroot)$. This completes the proof that $m$ is in $marked$ or $ReachableFrom(vroot)$.

$$Inv3 \;\triangleq\; Reachable = marked \cup ReachableFrom(vroot)$$

For convenience, let $R$ equal $marked \cup ReachableFrom(vroot)$. In the initial state, $marked = \{\}$ and $vroot = \{Root\}$, so $R$ equals Reachable and $Inv3$ is true. We have to show that each action $a$ step leaves $R$ unchanged. There are two cases:

$Case1$: The $a$ step adds an element $v$ of $vroot$ to $marked$ and adds to $vroot$ the nodes in $Succ[v]$, which are all in $ReachableFrom(vroot)$. Since $v$ itself is also in $ReachableFrom(vroot)$, the step leaves $R$ unchanged.

Case 2: The $a$ step removes from $vroot$ an element $v$ of $marked$ . Since $Inv1$ implies that every node in $Succ[v]$ is in $vroot$, the only element that this step removes from $ReachableFrom(vroot)$ is $v$, which the step adds to $marked$ . Hence $R$ is unchanged.

It is straightforward to use *TLC* to check that $Inv1 - Inv3$ are invariants of the algorithm for small graphs.

Partial correctness of the algorithm means that if it has terminated, then $marked$ equals $Reachable$. The algorithm has terminated when $pc$ equals \Done", so this theorem asserts partial correctness.

$$\textsf{theorem}\ Spec \Rightarrow \Box((pc = \text{"Done"}) \Rightarrow (marked = Reachable))$$

*TypeOK* implies $(pc =$ \Done"$) \Rightarrow (vroot = \{\})$. Since, $ReachableFrom(\{\})$ equals $\{\}$, $Inv3$ implies $(vroot = \{\}) \Rightarrow (marked = Reachable)$. Hence the theorem follows from the invariance of *TypeOK* and $Inv3$.

The following theorem asserts that if the set of nodes reachable from Root is finite, then the algorithm eventually terminates. Of course, this liveness property can be true only because *Spec* implies weak fairness of *Next*, which equals action *a* .

$$\textbf{theorem assume } IsFiniteSet(Reachable)$$
$$\textbf{prove } Spec \Rightarrow \Diamond(pc = \text{“Done”})$$

To prove the theorem, we assume a behavior satisfies *Spec* and prove that it satisfies $\Diamond(pc =$ \Done"). If $pc =$ \a" and $vroot = \{\}$, then an $a$ step sets $pc$ to \Done". Since invariance of *TypeOK* implies $\Box(pc \in \{$\a"; \Done"$\})$, weak fairness of $a$ implies that to prove $\Diamond(pc =$ \Done"), it suffices to prove $\Diamond(vroot = \{\})$.

We prove $\Diamond(root = \{\})$ by contradiction. We assume it's false, which means that $\Box(root \neq \{\})$ is true, and obtain a contradiction. From
$\Box TypeOK$, we infer that $\Box(root \neq \{\})$ implies $\Box(pc =$ \a"). By weak
fairness of action $a$ , $\Box(root \neq \{\})$ implies that there are an infinite number of $a$ steps. The assumption that *Reachable* is finite and $\Box Inv3$ imply that *marked* and *vroot* are always finite. Since *vroot* is always finite and nonempty, from any state there can be only a finite number of $a$ steps that remove an element from *vroot* until there is an $a$ step that adds a new element to *marked* . Since there are an infinite number of $a$ steps, there must be an infinite number of steps that add new elements to *marked* . This is impossible because *marked* is a finite set. Hence, we have the required contradiction.

*TLC* can quickly check these two theorems on models containing a half dozen nodes.

* Modification History
* Last modified *Wed Apr* 17 12:21:22 *PDT* 2019 by *lamport*
* Created *Thu Apr* 04 10:11:51 *PDT* 2019 by *lamport*