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.

The proof of *Misra*'s variant is also subtler than that of the obvious algorithm. 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 )
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad \textbf{\{}\ \mathit{marked} := \mathit{marked} \cup \{v\}\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad \mathit{vroot} \quad := \mathit{vroot} \cup \mathit{Succ}[v]\ \textbf{\}} \\
\qquad\qquad\qquad \textbf{else}\ \textbf{\{}\ \mathit{vroot} := \mathit{vroot} \setminus \{v\}\ \textbf{\}} \\
\qquad\qquad\quad \textbf{\}} \\
\qquad\qquad \textbf{\}} \\
\quad \textbf{\}} \\
\textbf{\}}
\end{array}
$$

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

BEGIN TRANSLATION

VARIABLES $marked, vroot, pc$

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

$$
\begin{array}{l}
Init \triangleq \text{Global variables} \\
\qquad \land\ marked = \{\} \\
\qquad \land\ vroot = \{Root\} \\
\qquad \land\ pc = \text{"a"}
\end{array}
$$

$$
\begin{array}{l}
a \triangleq \land\ pc = \text{"a"} \\
\quad\ \ \land\ \text{IF } vroot \neq \{\} \\
\qquad\quad \text{THEN } \land\ \exists\, v \in vroot : \\
\qquad\qquad\qquad\quad \text{IF } v \notin marked \\
\qquad\qquad\qquad\qquad \text{THEN } \land\ marked' = (marked \cup \{v\}) \\
\qquad\qquad\qquad\qquad\qquad\quad\ \ \land\ vroot' = (vroot \cup Succ[v]) \\
\qquad\qquad\qquad\qquad \text{ELSE } \land\ vroot' = vroot \setminus \{v\} \\
\qquad\qquad\qquad\qquad\qquad\quad\ \ \land\ \text{UNCHANGED } marked \\
\qquad\qquad\quad\ \ \land\ pc' = \text{"a"} \\
\qquad\quad \text{ELSE } \land\ pc' = \text{"Done"} \\
\qquad\qquad\quad\ \ \land\ \text{UNCHANGED } \langle marked, vroot \rangle
\end{array}
$$

$$
\begin{array}{l}
Next \triangleq a \\
\qquad\quad \lor\ \text{Disjunct to prevent deadlock on termination} \\
\qquad\qquad (pc = \text{"Done"} \land \text{UNCHANGED } vars)
\end{array}
$$

$$
\begin{array}{l}
Spec \triangleq \land\ Init \land \Box[Next]_{vars} \\
\qquad\quad \land\ \mathrm{WF}_{vars}(Next)
\end{array}
$$

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

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.

$$TypeOK \triangleq \land\ marked \in \text{SUBSET } Nodes$$
$$\qquad \land\ vroot \in \text{SUBSET } Nodes$$
$$\qquad \land\ pc \in \{\text{"a"}, \text{"Done"}\}$$
$$\qquad \land\ (pc = \text{"Done"}) \Rightarrow (vroot = \{\})$$

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.)

$$Inv1 \triangleq \land\ TypeOK$$
$$\qquad \land\ \forall\, n \in marked : Succ[n] \subseteq (marked \cup vroot)$$

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.

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

TypeOK implies $(pc = \text{"Done"}) \Rightarrow (vroot = \{\})$. 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* .

THEOREM ASSUME $IsFiniteSet(Reachable)$
PROVE $Spec \Rightarrow \Diamond(pc = \text{"Done"})$

To prove the theorem, we assume a behavior satisfies *Spec* and prove that it satisfies $\Diamond(pc = \text{"Done"})$. If $pc = \text{"a"}$ and $vroot = \{\}$, then an *a* step sets *pc* to "Done". Since invariance of *TypeOK* implies $\Box(pc \in \{\text{"a"}, \text{"Done"}\})$, weak fairness of *a* implies that to prove $\Diamond(pc = \text{"Done"})$, 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 $\Box(pc = \text{"a"})$. 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 Sun *Apr* 14 16:21:57 *PDT* 2019 by *lamport*
* Created *Thu Apr* 04 10:11:51 *PDT* 2019 by *lamport*