MODULE *ParReach*

This module describes an algorithm that is a multiprocess implementation of the algorithm in module *Reachable*. You should understand that algorithm before reading further in this module.

Here, this module's algorithm will be called the parallel algorithm and module *Reachable*'s algorithm will be called *Misra*'s algorithm. The parallel algorithm computes the same thing as *Misra*'s algorithm – namely, it terminates if and only if the set of nodes reachable from Root is finite, in which case the variable *marked* equals that reachable set when the algorithm terminates. The parallel algorithm does this by implementing *Misra*'s algorithm under a refinement mapping such that

- The parallel algorithm's *marked* variable implements the *Misra* algorithm's variable of the same name.

- The parallel algorithm has terminated if and only if *Misra*'s algorithm has.

EXTENDS *Reachability*, *Integers*, *FiniteSets*

The *Misra* algorithm's constants *Nodes* and *Succ* are imported from module Reachability. Its *Root* constant is declared here, as is the set *Procs* of processes that execute the parallel algorithm.

CONSTANT *Root*, *Procs*

ASSUME $RootAssump \triangleq Root \in Nodes$

ASSUME $ProcsAssump \triangleq \land Procs \neq \{\}$
$\qquad\qquad\qquad\qquad\qquad \land IsFiniteSet(Procs)$

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

This definition is copied from module *Reachable*. (We don't want to extend that module because it would lead to name conflicts with the TLA+ translation of the algorithm in this module.)

****

The algorithm executed by each process is the same as *Misra*'s algorithm, except that where the *Misra* algorithm adds to *marked* a node v in *vroot* and adds the nodes in $Succ[v]$ to *vroot* in a single step, the parallel algorithm first adds v to *marked* and then adds the nodes in $Succ[v]$ one at a time to *vroot* in separate steps.

Note that one process can add node v to *marked* , and then another process can remove v from *vroot* before the first process has added any nodes in $Succ[v]$ to *vroot*. This implies that a process can find *vroot* equal to $\{\}$ and terminate before *marked* contains all the nodes reachable from *Root*. This doesn't affect correctness of the parallel algorithm because, as long as *marked* doesn't contain all the nodes it should, there will be at least one process that hasn't terminated. However, it can cause inefficiency because processes can terminate while there is still useful work for them to do. In practice, the evaluation of the test $vroot \neq \{\}$ can be implemented in a way that makes it impossible or perhaps very unlikely to obtain the value FALSE if there are still nodes being added to *vroot*. Devising such an implementation is a nice exercise that will not be discussed here.

Here is the algorithm's *PlusCal* code. The initial values of the process-local variables u and *toVroot* don't matter. It's always a good idea to make them "type correct", and the initial value of *toVroot* is an obvious choice that also makes the refinement mapping a little simpler.

```
--algorithm ParallelReachability{
  variables marked = {}, vroot = {Root} ;
  fair process ( p ∈ Procs )
    variables u = Root, toVroot = {} ;
    { a: while ( vroot ≠ {} )
```

```
{   with ( v ∈ vroot ) { u := v } ;
  b: if ( u ∉ marked )
       {   marked := marked ∪ {u} ;
           toVroot := Succ[u] ;
        c: while ( toVroot ≠ {} )
             { with ( w ∈ toVroot ) {
                   vroot := vroot ∪ {w} ;
                   toVroot := toVroot \ {w} }
             }
       }
     else { vroot := vroot \ {u} }
  }
 }
}
```

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

BEGIN TRANSLATION

VARIABLES $marked$, $vroot$, pc, u, $toVroot$

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

$ProcSet \triangleq (Procs)$

$Init \triangleq$ Global variables

$$\begin{aligned}
&\land marked = \{\} \\
&\land vroot = \{Root\} \\
&\text{Process } p \\
&\land u = [self \in Procs \mapsto Root] \\
&\land toVroot = [self \in Procs \mapsto \{\}] \\
&\land pc = [self \in ProcSet \mapsto \text{“a”}]
\end{aligned}$$

$$\begin{aligned}
a(self) \triangleq\ &\land pc[self] = \text{“a”} \\
&\land \text{IF } vroot \neq \{\} \\
&\qquad \text{THEN } \land \exists v \in vroot : \\
&\qquad\qquad\qquad u' = [u \text{ EXCEPT } ![self] = v] \\
&\qquad\qquad\ \ \land pc' = [pc \text{ EXCEPT } ![self] = \text{“b”}] \\
&\qquad \text{ELSE } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}] \\
&\qquad\qquad\ \ \land u' = u \\
&\land \text{UNCHANGED } \langle marked, vroot, toVroot \rangle
\end{aligned}$$

$$\begin{aligned}
b(self) \triangleq\ &\land pc[self] = \text{“b”} \\
&\land \text{IF } u[self] \notin marked \\
&\qquad \text{THEN } \land marked' = (marked \cup \{u[self]\}) \\
&\qquad\qquad\ \ \land toVroot' = [toVroot \text{ EXCEPT } ![self] = Succ[u[self]]] \\
&\qquad\qquad\ \ \land pc' = [pc \text{ EXCEPT } ![self] = \text{“c”}] \\
&\qquad\qquad\ \ \land vroot' = vroot
\end{aligned}$$

$$
\begin{array}{l}
\qquad\qquad \text{ELSE} \quad \land\ vroot' = vroot \setminus \{u[self]\} \\
\qquad\qquad\qquad\quad\ \ \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"a"}] \\
\qquad\qquad\qquad\quad\ \ \land\ \text{UNCHANGED } \langle marked,\ toVroot \rangle \\
\qquad \land\ u' = u
\end{array}
$$

$$
\begin{array}{ll}
c(self) \triangleq & \land\ pc[self] = \text{"c"} \\
& \land\ \text{IF } toVroot[self] \neq \{\} \\
& \qquad \text{THEN} \quad \land\ \exists\, w \in toVroot[self] : \\
& \qquad\qquad\qquad \land\ vroot' = (vroot \cup \{w\}) \\
& \qquad\qquad\qquad \land\ toVroot' = [toVroot \text{ EXCEPT } ![self] = toVroot[self] \setminus \{w\}] \\
& \qquad\qquad\quad\ \ \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"c"}] \\
& \qquad \text{ELSE} \quad\ \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"a"}] \\
& \qquad\qquad\quad\ \ \land\ \text{UNCHANGED } \langle vroot,\ toVroot \rangle \\
& \land\ \text{UNCHANGED } \langle marked,\ u \rangle
\end{array}
$$

$$
p(self) \triangleq a(self) \lor b(self) \lor c(self)
$$

$$
\begin{array}{ll}
Next \triangleq & (\exists\, self \in Procs : p(self)) \\
& \quad \lor \quad \text{Disjunct to prevent deadlock on termination} \\
& \qquad ((\forall\, self \in ProcSet : pc[self] = \text{"Done"}) \land \text{UNCHANGED } vars)
\end{array}
$$

$$
\begin{array}{ll}
Spec \triangleq & \land\ Init \land \Box[Next]_{vars} \\
& \land\ \forall\, self \in Procs : \mathrm{WF}_{vars}(p(self))
\end{array}
$$

$$
Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{"Done"})
$$

END TRANSLATION

The formula *Inv* defined below is the inductive invariant needed to prove that the parallel algorithm implements the safety part of *Misra*'s algorithm. In addition to type correctness, it asserts two simple relations between the values of the local variables u and *toVroot* and the control state of the process:

- *toVroot* $= \{\}$ except when control is at c
- u is in $vroot \cup marked$ when control is at b. This is invariant because u is an element of *vroot* when control arrives at b, and an element is removed from *vroot* only after it is added to *marked* .

The fact that such a simple invariant is needed to prove that each step of the parallel algorithm implements a step allowed by *Misra*'s algorithm means that the easiest way to show that the parallel algorithm is correct, given the correctness of *Misra*'s algorithm, is to show that it implements *Misra*'s algorithm.

$$
\begin{array}{ll}
Inv \triangleq & \land\ marked \in \text{SUBSET } Nodes \\
& \land\ vroot \in \text{SUBSET } Nodes \\
& \land\ u \in [Procs \rightarrow Nodes] \\
& \land\ toVroot \in [Procs \rightarrow \text{SUBSET } Nodes] \\
& \land\ pc \in [Procs \rightarrow \{\text{"a"}, \text{"b"}, \text{"c"}, \text{"Done"}\}] \\
& \land\ \forall\, q \in Procs : \land\ (pc[q] \in \{\text{"a"}, \text{"b"}, \text{"Done"}\}) \Rightarrow (toVroot[q] = \{\}) \\
& \qquad\qquad\qquad\quad\ \land\ (pc[q] = \text{"b"}) \Rightarrow (u[q] \in vroot \cup marked)
\end{array}
$$

To define a refinement mapping from states of the parallel algorithm to states of *Misra*'s algorithm, we must define the expressions in this module that represent the values of the *Misra* algorithm's variables. The value of the variable *marked* of *Misra*'s algorithm is represented by the value of variable *marked* of the parallel algorithm. The values of the variables *vroot* and *pc* of *Misra*'s algorithm are represented by the expressions *vrootBar* and *pcBar* defined below.

For the parallel algorithm to implement *Misra*'s algorithm under this refinement mapping, the same *b* step that adds node *u* to *marked* must also add *Succ*[*u*] to *vrootBar*. The *c* steps that move individual elements of *Succ*[*u*] from *toVroot* must leave *vrootBar* unchanged. (Those steps implement stuttering steps of *Misra*'s algorithm.) This leads us to the following definition of *vrootBar*.

$$vrootBar \triangleq vroot \cup \text{UNION } \{toVroot[i] : i \in Procs\}$$

Since the variable *pc* of *Misra*'s algorithm always equals "a" or "Done", and we want it to equal "Done" if and only if the parallel algorithm has terminated, the definition of *pcBar* is obvious.

$$pcBar \triangleq \text{IF } \forall\, q \in Procs : pc[q] = \text{“Done”} \text{ THEN } \text{“Done”} \text{ ELSE } \text{“a”}$$

$$R \triangleq \text{INSTANCE } Reachable \text{ WITH } vroot \leftarrow vrootBar,\ pc \leftarrow pcBar$$

For every definition *Op* in module *Reachable*, this statement defines the operator *R*!*Op* in the current module to have as its definition the expression obtained from the definition of *Op* in module *Reachable* by substituting *vrootBar* for *vroot* and *pcBar* for *pc*.

The following theorem is the TLA+ assertion of the statement that the parallel algorithm implements *Misra*'s algorithm under the refinement mapping defined above. It can be checked by *TLC* for models large enough to make us reasonably confident that it is correct. (To check it, use models with behavior specification spec that check the temporal property *R*!*Spec*.)

$$\text{THEOREM } Spec \Rightarrow R!Spec$$

By the definition of formula *Spec* of module *Reachable*, this theorem follows from the following two theorems, the first showing that the parallel algorithm implements the safety part of *Misra*'s algorithm (which implies partial correctness) and the second showing that it implements the fairness part of that algorithm (which implies termination). Module *ParReachProofs* contains a *TLAPS* checked proof of the first theorem.

$$\text{THEOREM } Spec \Rightarrow R!Init \land \Box[R!Next]_{R!vars}$$
$$\text{THEOREM } Spec \Rightarrow \mathrm{WF}_{R!vars}(R!Next)$$

* Modification History
* Last modified Sun *Apr* 14 16:53:23 *PDT* 2019 by *lamport*
* Created *Mon Apr* 08 10:48:52 *PDT* 2019 by *lamport*