MODULE *ReachabilityTest*

The correctness of an algorithm often depends on properties of one or more data structures. For example, the algorithm of module $Reachable$ depends on properties of directed graphs. Assuming those properties, once we have found suitable invariants, proving correctness of the algorithm is often a matter of checking a large number of fairly simple details. Writing a proof checked by $TLAPS$ can be a cost-e ective way of making sure that the algorithm is correct{assuming that those properties are correct.

Writing $TLAPS$ checked proofs of properties of data structures is often di cult. When verifying correctness of an algorithm, it may not be a cost-e ective way of verifying correctness of those properties of data structures. Correctness of an algorithm algorithm rarely depends on subtle new mathematical properties of the data structures. The properties of a data structure that that the algorithm relies on are almost always well known or obvious. The incorrectness of the algorithm's correctness proof introduced by assuming a property of a data structure will most likely be the result of an incorrect TLA+ statement of a correct property. Checking the property on small models is an e ective way of catching such an error.

And remember that it's easier to get $TLAPS$ to prove something if it's true. Even if you intend to prove properties of a data structure, you should use TLC to check that those properties are true before you start writing a TLA+ proof.

The proof of correctness in module $ReachableProofs$ of the algorithm in module $Reachable$ uses

The first test to perform is to evaluate $Test$ with $SuccSet$ equal to the set of all possible values of $Succ$, using as large a set $Nodes$ as we can. We do this with a model in which $SuccSet$ equals the following set $SuccSet1$.

$SuccSet1 \triangleq [Nodes \to \text{SUBSET } Nodes]$

For 3 nodes, $SuccSet1$ has 3^(2^3) (about 6500) elements. Evaluating Test on my laptop takes TLC a few seconds{mostly start-up time.

For 4 nodes, $SuccSet1$ has 4^16 (about 4*10^9) elements. I expect it will take TLC quite a few hours, and perhaps days, to evaluate $Test$.

We'd like to evaluate $Test$ for more than 4 nodes, so we do it by using a model that sets $Succ$ to $SuccSet2(n)$, which we now define to be a set of n randomly chosen values of $Succ$. A randomly chosen value of $Succ$ is a function that assigns to each node n a randomly chosen subset of Nodes. We choose a randomly chosen subset of $Nodes$ by using the $RandomElement$ operator from the TLC module.

In the following definition, $RandomSucc$ is given an unused parameter because TLC tries to optimize its execution by evaluating a definition of a constant value once when it starts up and using that same value every time it has to evaluate the constant.

```
RandomSucc(x) ≜
  LET RECURSIVE RS(_, _)
      RS(F, D) ≜ IF D = {}
                 THEN F
                 ELSE LET d ≜ CHOOSE d ∈ D : TRUE
                          S ≜ RandomElement(SUBSET Nodes)
                      IN  RS(F @@ (d :> S), D \ {d})
  IN  RS(⟨⟩, Nodes)

SuccSet2(n) ≜ {RandomSucc(i) : i ∈ 1 .. n}
```

With $Succ$ set to $SuccSet2(n)$, it takes TLC an average of about $3n$ seconds to evaluate $Test$ on my laptop for a set of 5 nodes. It seems to me that an error in the definition of $ReachableFrom$ or in one of the lemmas in one of the lemmas $Reachability0-3$ would almost certainly be manifest in a graph with 5 nodes. So, had these lemmas not been proved, I would have performed as many tests as I could on a graph with 5 nodes.

* Modification History
* Last modified Sun Apr 14 15:36:03 PDT 2019 by $lamport$
* Created $Fri\ Apr$ 05 19:21:52 PDT 2019 by $lamport$