```
MODULE SnapSpec
CONSTANT Proc, Val

PUnion(Q) ≜ UNION {Q[p] : p ∈ Proc}

*****
--algorithm SnapSpec
{ variables  myVals = [i ∈ Proc ↦ {}],
             nextout = [i ∈ Proc ↦ {}] ;
  process ( Pr ∈ Proc )
    variable out = {} ;
    { A: while ( TRUE )
           {    with ( v ∈ Val ) { myVals[self] := myVals[self] ∪ {v} } ;
             B: with ( V ∈ {W ∈ SUBSET PUnion(myVals) :
                                  ∧ myVals[self] ⊆ W
                                  ∧ PUnion(nextout) ⊆ W} )
                  { nextout[self] := V } ;
             C: either out := nextout[self]
                or     goto B ;
           }
    }
}
****
BEGIN TRANSLATION
VARIABLES myVals, nextout, pc, out

vars ≜ ⟨myVals, nextout, pc, out⟩

ProcSet ≜ (Proc)

Init ≜  Global variables
        ∧ myVals = [i ∈ Proc ↦ {}]
        ∧ nextout = [i ∈ Proc ↦ {}]
        Process Pr
        ∧ out = [self ∈ Proc ↦ {}]
        ∧ pc = [self ∈ ProcSet ↦ "A"]

A(self) ≜  ∧ pc[self] = "A"
           ∧ ∃ v ∈ Val :
               myVals' = [myVals EXCEPT ![self] = myVals[self] ∪ {v}]
           ∧ pc' = [pc EXCEPT ![self] = "B"]
           ∧ UNCHANGED ⟨nextout, out⟩

B(self) ≜  ∧ pc[self] = "B"
           ∧ ∃ V ∈ {W ∈ SUBSET PUnion(myVals) :
                          ∧ myVals[self] ⊆ W
```

```
                                  ∧ PUnion(nextout) ⊆ W} :
                        nextout' = [nextout EXCEPT ![self] = V]
                  ∧ pc' = [pc EXCEPT ![self] = "C"]
                  ∧ UNCHANGED ⟨myVals, out⟩

C(self) ≜  ∧ pc[self] = "C"
           ∧ ∨ ∧ out' = [out EXCEPT ![self] = nextout[self]]
               ∧ pc' = [pc EXCEPT ![self] = "A"]
             ∨ ∧ pc' = [pc EXCEPT ![self] = "B"]
               ∧ out' = out
           ∧ UNCHANGED ⟨myVals, nextout⟩

Pr(self) ≜ A(self) ∨ B(self) ∨ C(self)

Next ≜ (∃ self ∈ Proc : Pr(self))

Spec ≜ Init ∧ □[Next]_vars

END TRANSLATION
```

It's always a good idea to write and have *TLC* check a type invariant.

```
TypeOK ≜  ∧ myVals ∈ [Proc → SUBSET Val]
          ∧ out ∈ [Proc → PUnion(myVals)]
```

Specification *BigSpec* is the same as the specification *Spec* produced by translating the *SnapSpec* algorithm's code except that it allows a step, enabled when some process p is at control point A, that moves some set of processes at control point C to control point B – just as if they had all simulateously executed statement C, choosing to perform the second clause of the enabled/or statement. Spec *BigSpec* has the same initial predicate as *Spec*. It's next-state action *BigNext* is the disjunction of the next-state action of algorithm *SnapSpec* with an action that describes the extra step the *BegSpec* allows.

```
BigNext ≜  ∨ Next
           ∨ ∃ p ∈ Proc :
               ∧ pc[p] = "A"
               ∧ ∃ P ∈ SUBSET (Proc \ {p}) :
                   ∧ ∀ q ∈ P : pc[q] = "C"
                   ∧ pc' = [q ∈ Proc ↦ IF q ∈ P THEN "B"
                                              ELSE pc[q]]
                   ∧ UNCHANGED ⟨myVals, nextout,  out⟩

BigSpec ≜ Init ∧ □[BigNext]_vars
```

```
\* Modification History
\* Last modified Fri Jul 20 11:48:06 PDT 2012 by lamport
\* Created Wed Jul 04 23:50:00 PDT 2012 by lamport
```