CS 6110 Software Correctness, Spring 2022
Lec5

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

URL: bit.ly/cs6110s22

SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

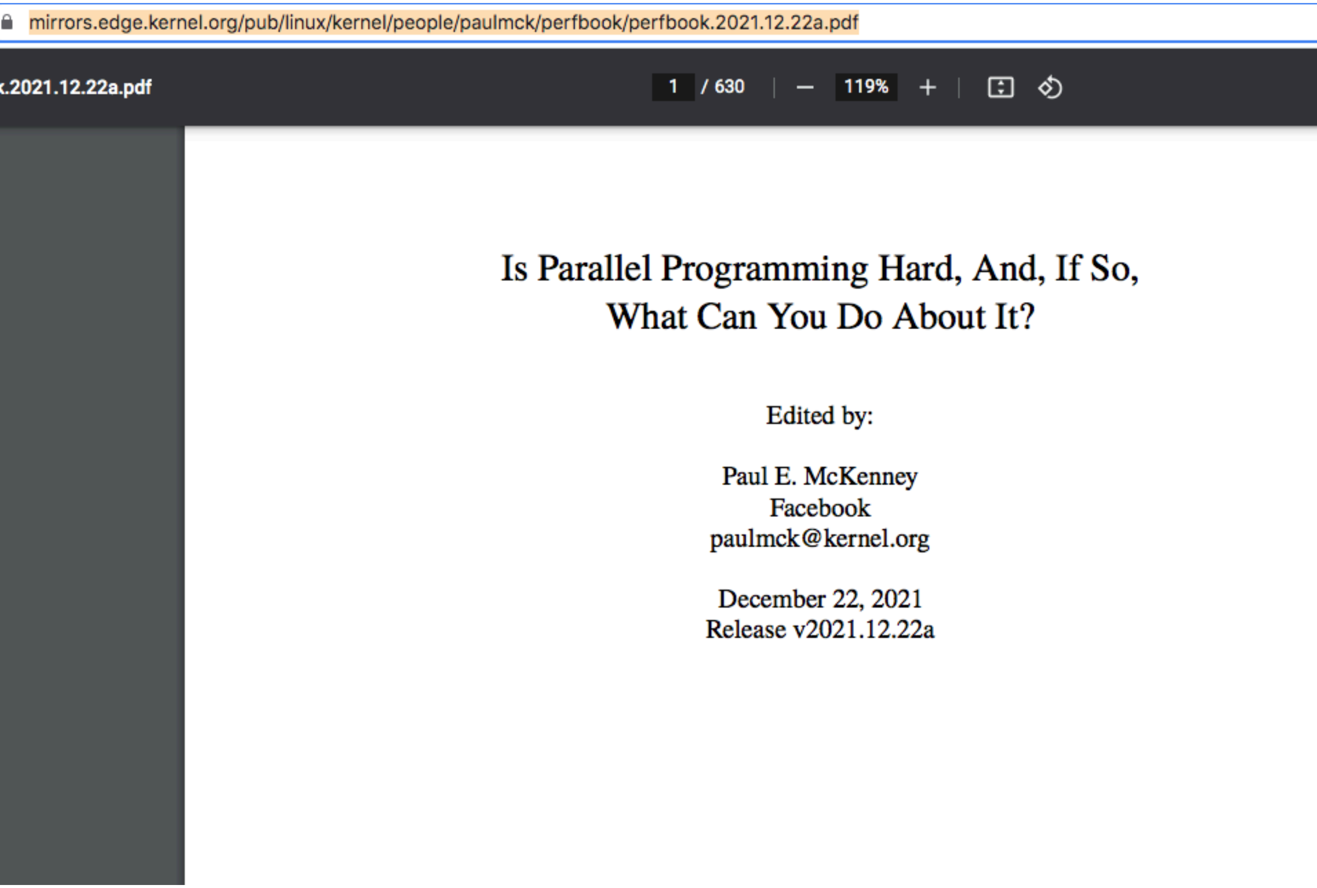
Slides for Lec5 : Agenda

- Bugs matter (loss of lives, property, wasted time, poor user experience..)
  - Therac incident https://en.wikipedia.org/wiki/Therac-25
    - Patients killed
  - Heartbleed https://dwheeler.com/essays/heartbleed.html
    - A huge mistake
- Protocols baked into code are inscrutably complex (hard to debug at that level)
  - Build models (like executable blueprints – or like scale model of an airplane in a wind tunnel)
    - Error-prone but can be gotten right - readable rigorous specification!
    - You'll see the styles that matter + the extras (checks in SPIN) that help
      - Murphi's rule-style shines! - will later show the power of Table-based transition systems! (like Excel sheets)
  - Then auto-generate C or protocols – have MULTIPLE models (so we can check all vantage-points)
  - After tuning/tweaking, switch to C-level verification
- Asg-2 is to take you through these paths
  - We will experience typing a model
  - We will sow a bug in Distributed Termination (work channel slack)
    - And experience how verification goes

Take it from the experts!

- Paul E. McKenney
    - Top Linux Developer at IBM (now works for Facebook)
    - Has authored a free book on parallel programming that he maintains
        - A fantastic collection on (shared memory) parallel programming
        - https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.2021.12.22a.pdf
- Look at its table of contents (next slide)

Take it from the experts!



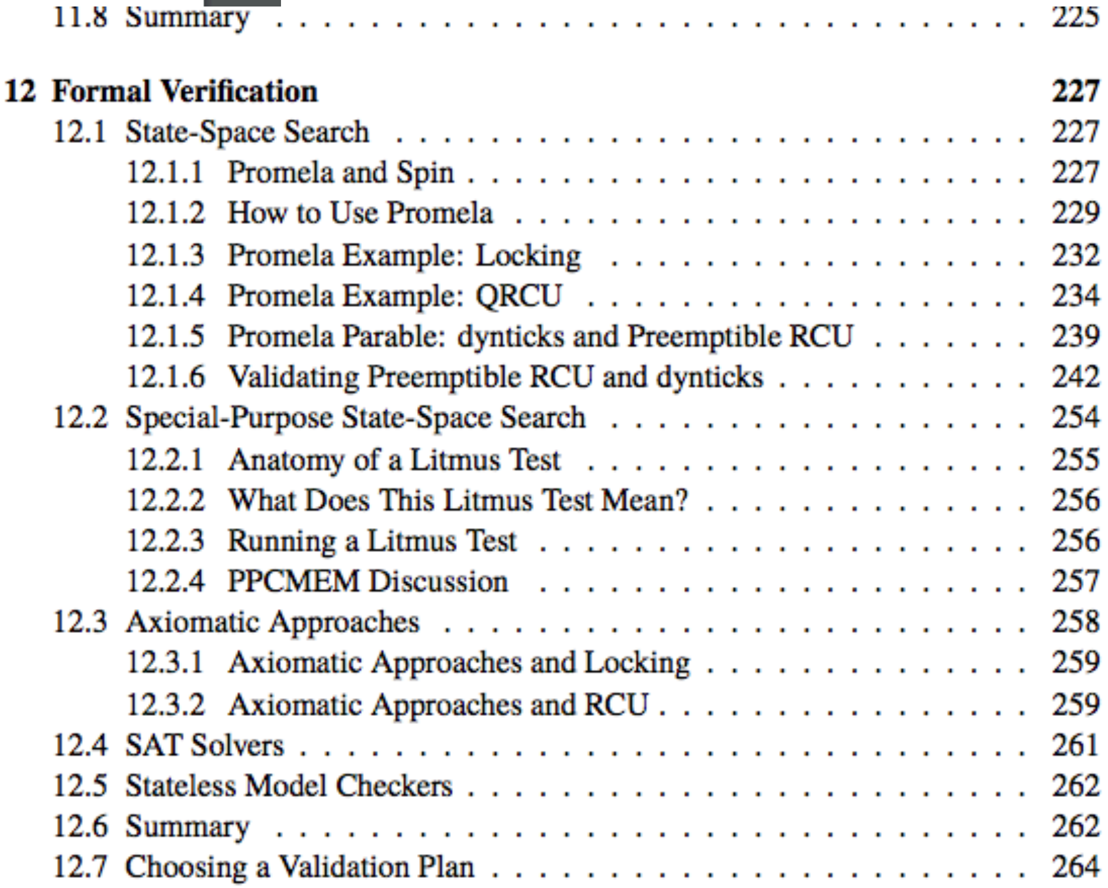
mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.2021.12.22a.pdf

Is Parallel Programming Hard, And, If So,
What Can You Do About It?

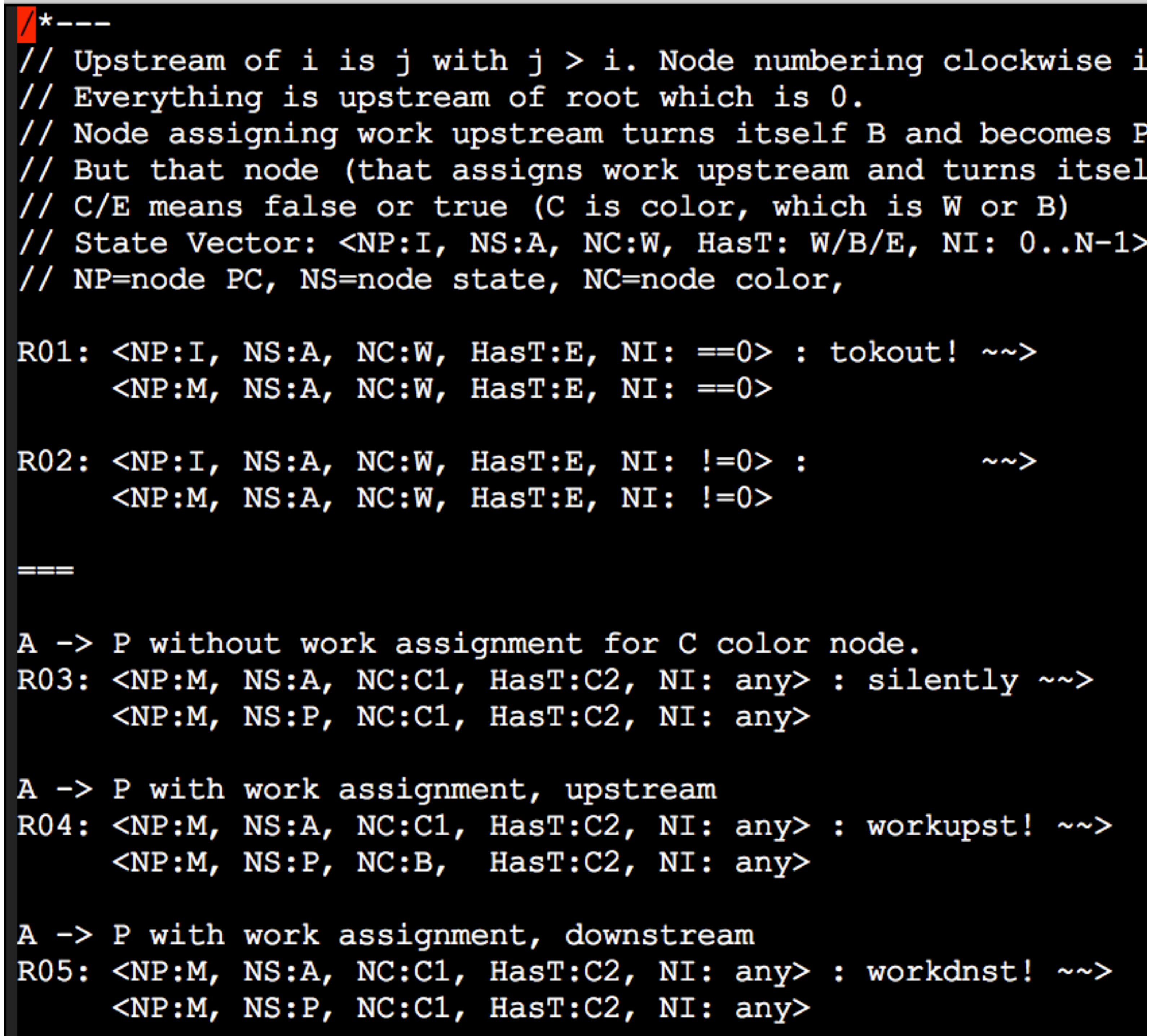
Edited by:

Paul E. McKenney
Facebook
paulmck@kernel.org

December 22, 2021
Release v2021.12.22a

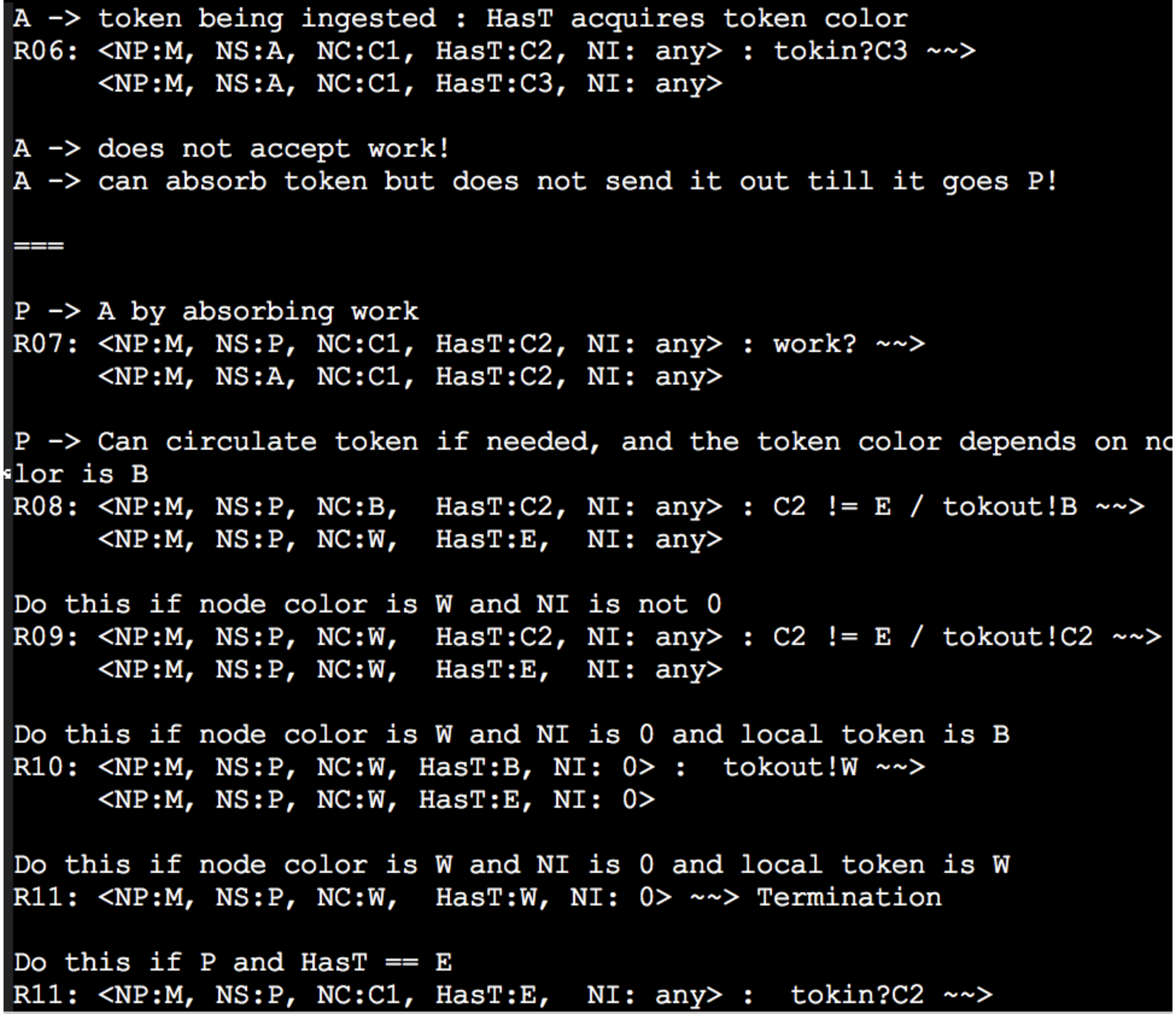


Dist Term



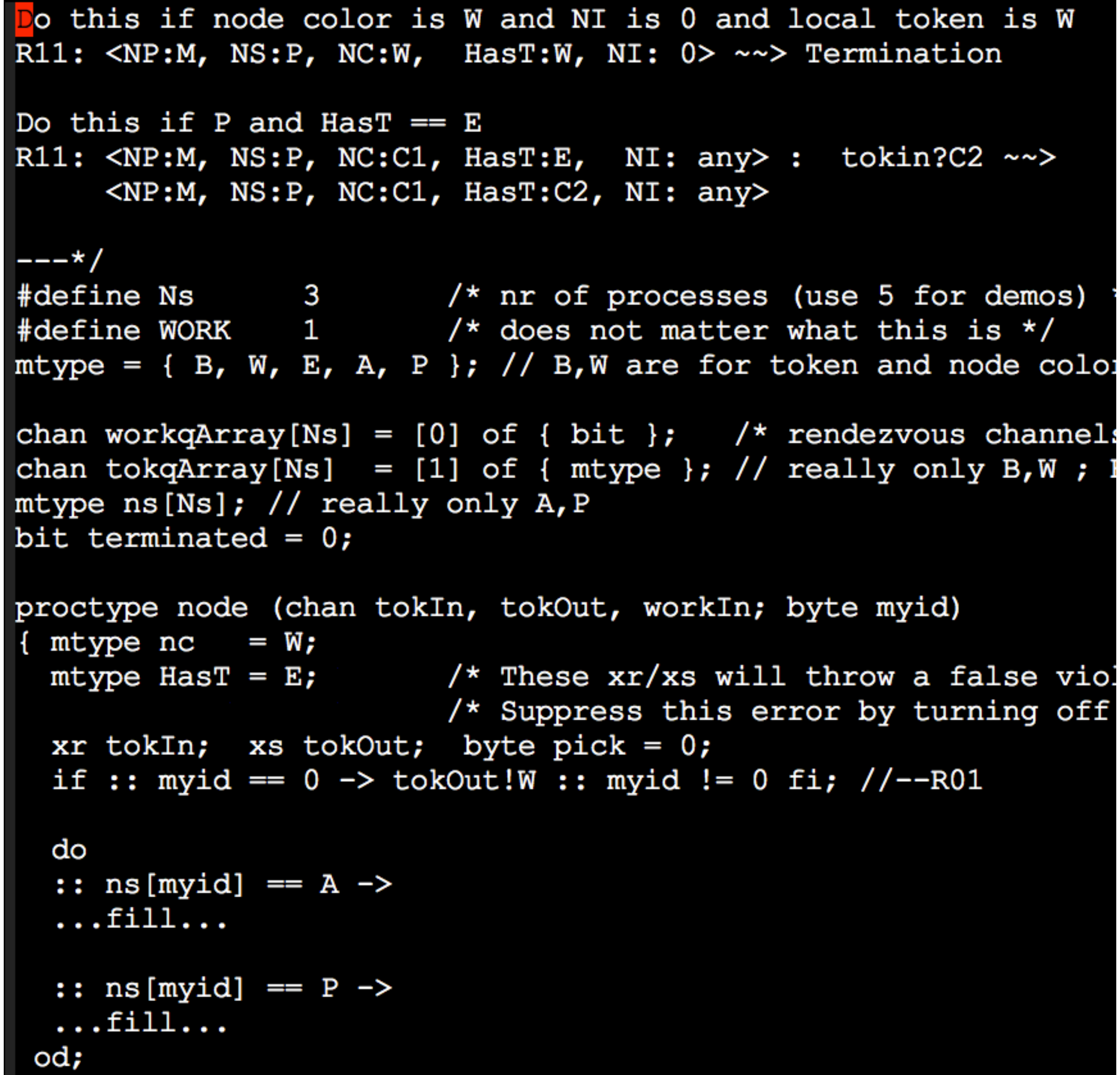
```
/*---
// Upstream of i is j with j > i. Node numbering clockwise i
// Everything is upstream of root which is 0.
// Node assigning work upstream turns itself B and becomes P
// But that node (that assigns work upstream and turns itsel
// C/E means false or true (C is color, which is W or B)
// State Vector: <NP:I, NS:A, NC:W, HasT: W/B/E, NI: 0..N-1>
// NP=node PC, NS=node state, NC=node color,

R01: <NP:I, NS:A, NC:W, HasT:E, NI: ==0> : tokout! ~~>
     <NP:M, NS:A, NC:W, HasT:E, NI: ==0>

R02: <NP:I, NS:A, NC:W, HasT:E, NI: !=0> :         ~~>
     <NP:M, NS:A, NC:W, HasT:E, NI: !=0>

===

A -> P without work assignment for C color node.
R03: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : silently ~~>
     <NP:M, NS:P, NC:C1, HasT:C2, NI: any>

A -> P with work assignment, upstream
R04: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : workupst! ~~>
     <NP:M, NS:P, NC:B,  HasT:C2, NI: any>

A -> P with work assignment, downstream
R05: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : workdnst! ~~>
     <NP:M, NS:P, NC:C1, HasT:C2, NI: any>
```

Dist Term



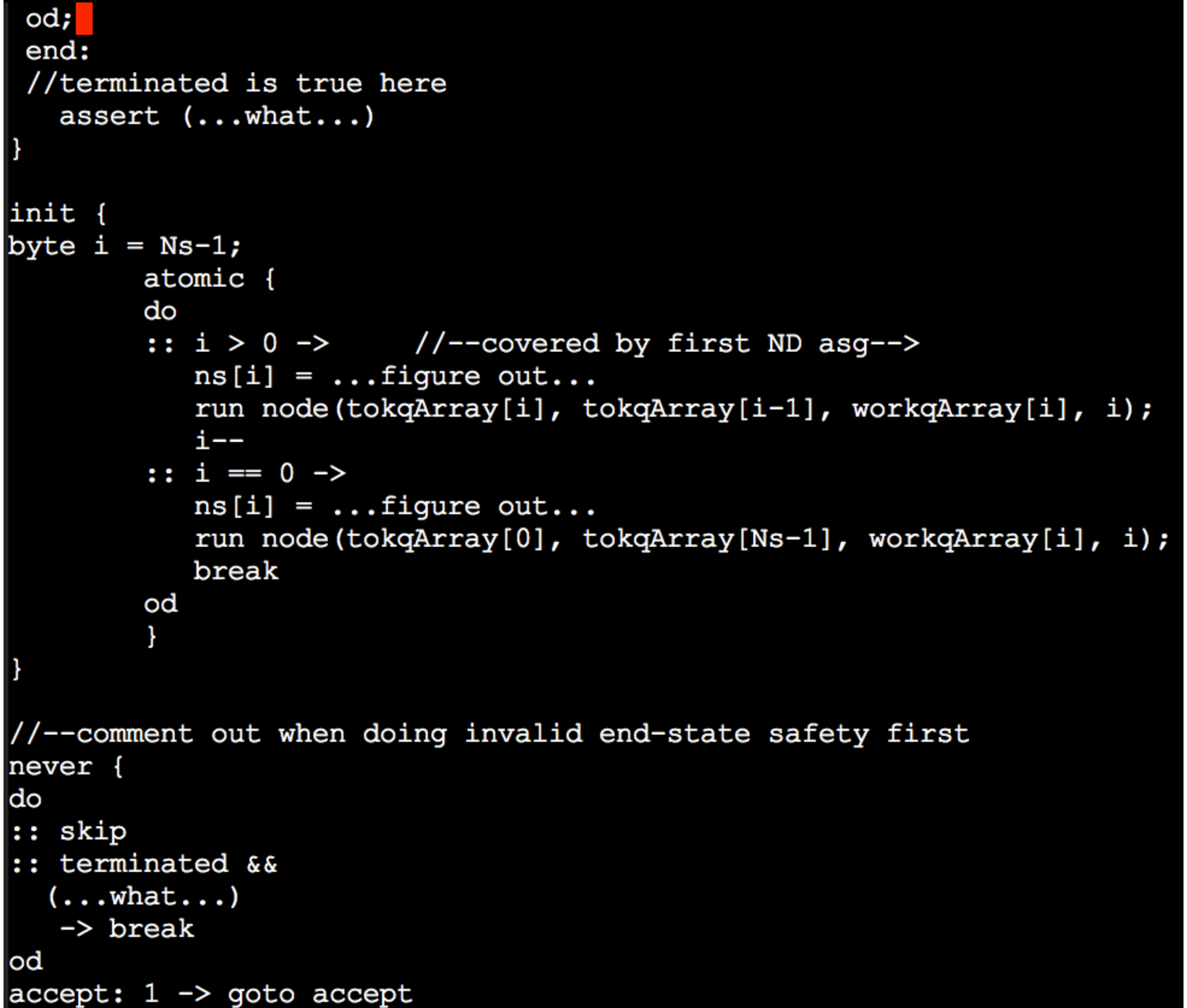
```
A -> token being ingested : HasT acquires token color
R06: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : tokin?C3 ~~>
     <NP:M, NS:A, NC:C1, HasT:C3, NI: any>

A -> does not accept work!
A -> can absorb token but does not send it out till it goes P!


===


P -> A by absorbing work
R07: <NP:M, NS:P, NC:C1, HasT:C2, NI: any> : work? ~~>
     <NP:M, NS:A, NC:C1, HasT:C2, NI: any>

P -> Can circulate token if needed, and the token color depends on no
lor is B
R08: <NP:M, NS:P, NC:B,  HasT:C2, NI: any> : C2 != E / tokout!B ~~>
     <NP:M, NS:P, NC:W,  HasT:E,  NI: any>

Do this if node color is W and NI is not 0
R09: <NP:M, NS:P, NC:W,  HasT:C2, NI: any> : C2 != E / tokout!C2 ~~>
     <NP:M, NS:P, NC:W,  HasT:E,  NI: any>

Do this if node color is W and NI is 0 and local token is B
R10: <NP:M, NS:P, NC:W, HasT:B, NI: 0> :  tokout!W ~~>
     <NP:M, NS:P, NC:W, HasT:E, NI: 0>

Do this if node color is W and NI is 0 and local token is W
R11: <NP:M, NS:P, NC:W,  HasT:W, NI: 0> ~~> Termination

Do this if P and HasT == E
R11: <NP:M, NS:P, NC:C1, HasT:E,  NI: any> :  tokin?C2 ~~>
```

Dist Term



```
Do this if node color is W and NI is 0 and local token is W
R11: <NP:M, NS:P, NC:W,  HasT:W, NI: 0> ~~> Termination

Do this if P and HasT == E
R11: <NP:M, NS:P, NC:C1, HasT:E,  NI: any> :  tokin?C2 ~~>
     <NP:M, NS:P, NC:C1, HasT:C2, NI: any>

---*/
#define Ns      3       /* nr of processes (use 5 for demos)
#define WORK    1       /* does not matter what this is */
mtype = { B, W, E, A, P }; // B,W are for token and node colo

chan workqArray[Ns] = [0] of { bit };   /* rendezvous channel
chan tokqArray[Ns]  = [1] of { mtype }; // really only B,W ;
mtype ns[Ns]; // really only A,P
bit terminated = 0;

proctype node (chan tokIn, tokOut, workIn; byte myid)
{ mtype nc   = W;
  mtype HasT = E;       /* These xr/xs will throw a false vio
                        /* Suppress this error by turning off
  xr tokIn;  xs tokOut;  byte pick = 0;
  if :: myid == 0 -> tokOut!W :: myid != 0 fi; //--R01

  do
  :: ns[myid] == A ->
  ...fill...

  :: ns[myid] == P ->
  ...fill...
 od;
```

Dist Term



```
 od;
 end:
 //terminated is true here
   assert (...what...)
}

init {
byte i = Ns-1;
        atomic {
        do
        :: i > 0 ->     //--covered by first ND asg-->
           ns[i] = ...figure out...
           run node(tokqArray[i], tokqArray[i-1], workqArray[i], i);
           i--
        :: i == 0 ->
           ns[i] = ...figure out...
           run node(tokqArray[0], tokqArray[Ns-1], workqArray[i], i);
           break
        od
        }
}

//--comment out when doing invalid end-state safety first
never {
do
:: skip
:: terminated &&
  (...what...)
   -> break
od
accept: 1 -> goto accept
```