

Solution for Exercise 4

Michał Kapałka

LPD, EPFL

November 3, 2008

The Splitter Object

- Only one operation: *splitter*
- Returns: *stop*, *left* or *right*
- If a **single** process executes *splitter*, then this process gets *stop*.
- If **two or more** processes invoke *splitter*, then not all get the same output.
- At most one process gets *stop*.

An Implementation of a Splitter

We use two registers:

- P (multi-valued), and
- S (binary, initialized to *false*)

upon *splitter_i*

$P \leftarrow i$

if S then return "right"

$S \leftarrow true$

if $P = i$ then return "stop"

return "left"

An Implementation of a Splitter

We use two registers:

- P (multi-valued), and
- S (binary, initialized to *false*)

upon *splitter* _{i}

$P \leftarrow i$

if S **then return** *"right"*

$S \leftarrow true$

if $P = i$ **then return** *"stop"*

return *"left"*

Non-adaptive Snapshot

upon *scan*_{*i*}

$t_1 \leftarrow \text{collect}(), t_2 \leftarrow t_1$

while *true* **do**

$t_3 \leftarrow \text{collect}()$

if $t_3 = t_2$ **then return** $\langle t_3[1].\text{val}, \dots, t_3[N].\text{val} \rangle$

for $k \leftarrow 1$ **to** N **do**

if $t_3[k].\text{ts} \geq t_1[k].\text{ts} + 2$ **then return** $t_3[k].\text{snapshot}$

$t_2 \leftarrow t_3$

procedure *collect*_{*i*}_{*j*}_{*k*}_{*l*}_{*m*}_{*n*}_{*o*}_{*p*}_{*q*}_{*r*}_{*s*}_{*t*}_{*u*}_{*v*}_{*w*}_{*x*}_{*y*}_{*z*}_{*aa*}_{*ab*}_{*ac*}_{*ad*}_{*ae*}_{*af*}_{*ag*}_{*ah*}_{*ai*}_{*aj*}_{*ak*}_{*al*}_{*am*}_{*an*}_{*ao*}_{*ap*}_{*aq*}_{*ar*}_{*as*}_{*at*}_{*au*}_{*av*}_{*aw*}_{*ax*}_{*ay*}_{*az*}_{*ba*}_{*bb*}_{*bc*}_{*bd*}_{*be*}_{*bf*}_{*bg*}_{*bh*}_{*bi*}_{*bj*}_{*bk*}_{*bl*}_{*bm*}_{*bn*}_{*bo*}_{*bp*}_{*bq*}_{*br*}_{*bs*}_{*bt*}_{*bu*}_{*bv*}_{*bw*}_{*bx*}_{*by*}_{*bz*}_{*ca*}_{*cb*}_{*cc*}_{*cd*}_{*ce*}_{*cf*}_{*cg*}_{*ch*}_{*ci*}_{*cj*}_{*ck*}_{*cl*}_{*cm*}_{*cn*}_{*co*}_{*cp*}_{*cq*}_{*cr*}_{*cs*}_{*ct*}_{*cu*}_{*cv*}_{*cw*}_{*cx*}_{*cy*}_{*cz*}_{*da*}_{*db*}_{*dc*}_{*dd*}_{*de*}_{*df*}_{*dg*}_{*dh*}_{*di*}_{*dj*}_{*dk*}_{*dl*}_{*dm*}_{*dn*}_{*do*}_{*dp*}_{*dq*}_{*dr*}_{*ds*}_{*dt*}_{*du*}_{*dv*}_{*dw*}_{*dx*}_{*dy*}_{*dz*}_{*ea*}_{*eb*}_{*ec*}_{*ed*}_{*ee*}_{*ef*}_{*eg*}_{*eh*}_{*ei*}_{*ej*}_{*ek*}_{*el*}_{*em*}_{*en*}_{*eo*}_{*ep*}_{*eq*}_{*er*}_{*es*}_{*et*}_{*eu*}_{*ev*}_{*ew*}_{*ex*}_{*ey*}_{*ez*}_{*fa*}_{*fb*}_{*fc*}_{*fd*}_{*fe*}_{*ff*}_{*fg*}_{*fh*}_{*fi*}_{*fj*}_{*fk*}_{*fl*}_{*fm*}_{*fn*}_{*fo*}_{*fp*}_{*fq*}_{*fr*}_{*fs*}_{*ft*}_{*fu*}_{*fv*}_{*fw*}_{*fx*}_{*fy*}_{*fz*}_{*ga*}_{*gb*}_{*gc*}_{*gd*}_{*ge*}_{*gf*}_{*gg*}_{*gh*}_{*gi*}_{*gj*}_{*gk*}_{*gl*}_{*gm*}_{*gn*}_{*go*}_{*gp*}_{*gq*}_{*gr*}_{*gs*}_{*gt*}_{*gu*}_{*gv*}_{*gw*}_{*gx*}_{*gy*}_{*gz*}_{*ha*}_{*hb*}_{*hc*}_{*hd*}_{*he*}_{*hf*}_{*hg*}_{*hh*}_{*hi*}_{*hj*}_{*hk*}_{*hl*}_{*hm*}_{*hn*}_{*ho*}_{*hp*}_{*hq*}_{*hr*}_{*hs*}_{*ht*}_{*hu*}_{*hv*}_{*hw*}_{*hx*}_{*hy*}_{*hz*}_{*ia*}_{*ib*}_{*ic*}_{*id*}_{*ie*}_{*if*}_{*ig*}_{*ih*}_{*ii*}_{*ij*}_{*ik*}_{*il*}_{*im*}_{*in*}_{*io*}_{*ip*}_{*iq*}_{*ir*}_{*is*}_{*it*}_{*iu*}_{*iv*}_{*iw*}_{*ix*}_{*iy*}_{*iz*}_{*ja*}_{*jb*}_{*jc*}_{*jd*}_{*je*}_{*jf*}_{*jj*}_{*jh*}_{*ji*}_{*jj*}_{*jk*}_{*jl*}_{*jm*}_{*jn*}_{*jo*}_{*jp*}_{*jq*}_{*jr*}_{*js*}_{*jt*}_{*ju*}_{*jv*}_{*jw*}_{*jx*}_{*ky*}_{*kz*}_{*la*}_{*lb*}_{*lc*}_{*ld*}_{*le*}_{*lf*}_{*lg*}_{*lh*}_{*li*}_{*lj*}_{*lk*}_{*ll*}_{*lm*}_{*ln*}_{*lo*}_{*lp*}_{*lq*}_{*lr*}_{*ls*}_{*lt*}_{*lu*}_{*lv*}_{*lw*}_{*lx*}_{*ly*}_{*lz*}_{*ma*}_{*mb*}_{*mc*}_{*md*}_{*me*}_{*mf*}_{*mg*}_{*mh*}_{*mi*}_{*mj*}_{*mk*}_{*ml*}_{*mm*}_{*mn*}_{*mo*}_{*mp*}_{*mq*}_{*mr*}_{*ms*}_{*mt*}_{*mu*}_{*mv*}_{*mw*}_{*mx*}_{*my*}_{*mz*}_{*na*}_{*nb*}_{*nc*}_{*nd*}_{*ne*}_{*nf*}_{*ng*}_{*nh*}_{*ni*}_{*nj*}_{*nk*}_{*nl*}_{*nm*}_{*nn*}_{*no*}_{*np*}_{*nq*}_{*nr*}_{*ns*}_{*nt*}_{*nu*}_{*nv*}_{*nw*}_{*nx*}_{*ny*}_{*nz*}_{*oa*}_{*ob*}_{*oc*}_{*od*}_{*oe*}_{*of*}_{*og*}_{*oh*}_{*oi*}_{*oj*}_{*ok*}_{*ol*}_{*om*}_{*on*}_{*oo*}_{*op*}_{*oq*}_{*or*}_{*os*}_{*ot*}_{*ou*}_{*ov*}_{*ow*}_{*ox*}_{*oy*}_{*oz*}_{*pa*}_{*pb*}_{*pc*}_{*pd*}_{*pe*}_{*pf*}_{*pg*}_{*ph*}_{*pi*}_{*pj*}_{*pk*}_{*pl*}_{*pm*}_{*pn*}_{*po*}_{*pp*}_{*pq*}_{*pr*}_{*ps*}_{*pt*}_{*pu*}_{*pv*}_{*pw*}_{*px*}_{*py*}_{*pz*}_{*qa*}_{*qb*}_{*qc*}_{*qd*}_{*qe*}_{*qf*}_{*qg*}_{*qh*}_{*qi*}_{*qj*}_{*qk*}_{*ql*}_{*qm*}_{*qn*}_{*qo*}_{*qp*}_{*qq*}_{*qr*}_{*qs*}_{*qt*}_{*qu*}_{*qv*}_{*qw*}_{*qx*}_{*qy*}_{*qz*}_{*ra*}_{*rb*}_{*rc*}_{*rd*}_{*re*}_{*rf*}_{*rg*}_{*rh*}_{*ri*}_{*rj*}_{*rk*}_{*rl*}_{*rm*}_{*rn*}_{*ro*}_{*rp*}_{*rq*}_{*rr*}_{*rs*}_{*rt*}_{*ru*}_{*rv*}_{*rw*}_{*rx*}_{*ry*}_{*rz*}_{*sa*}_{*sb*}_{*sc*}_{*sd*}_{*se*}_{*sf*}_{*sg*}_{*sh*}_{*si*}_{*sj*}_{*sk*}_{*sl*}_{*sm*}_{*sn*}_{*so*}_{*sp*}_{*sq*}_{*sr*}_{*ss*}_{*st*}_{*su*}_{*sv*}_{*sw*}_{*sx*}_{*sy*}_{*sz*}_{*ta*}_{*tb*}_{*tc*}_{*td*}_{*te*}_{*tf*}_{*tg*}_{*th*}_{*ti*}_{*tj*}_{*tk*}_{*tl*}_{*tm*}_{*tn*}_{*to*}_{*tp*}_{*tq*}_{*tr*}_{*ts*}_{*tt*}_{*tu*}_{*tv*}_{*tw*}_{*tx*}_{*ty*}_{*tz*}_{*ua*}_{*ub*}_{*uc*}_{*ud*}_{*ue*}_{*uf*}_{*ug*}_{*uh*}_{*ui*}_{*uj*}_{*uk*}_{*ul*}_{*um*}_{*un*}_{*uo*}_{*up*}_{*uq*}_{*ur*}_{*us*}_{*ut*}_{*uu*}_{*uv*}_{*uw*}_{*ux*}_{*uy*}_{*uz*}_{*va*}_{*vb*}_{*vc*}_{*vd*}_{*ve*}_{*vf*}_{*vg*}_{*vh*}_{*vi*}_{*vj*}_{*vk*}_{*vl*}_{*vm*}_{*vn*}_{*vo*}_{*vp*}_{*vq*}_{*vr*}_{*vs*}_{*vt*}_{*vu*}_{*vv*}_{*vw*}_{*vx*}_{*vy*}_{*vz*}_{*wa*}_{*wb*}_{*wc*}_{*wd*}_{*we*}_{*wf*}_{*wg*}_{*wh*}_{*wi*}_{*wj*}_{*wk*}_{*wl*}_{*wm*}_{*wn*}_{*wo*}_{*wp*}_{*wq*}_{*wr*}_{*ws*}_{*wt*}_{*wu*}_{*wv*}_{*ww*}_{*wx*}_{*wy*}_{*wz*}_{*xa*}_{*xb*}_{*xc*}_{*xd*}_{*xe*}_{*xf*}_{*xg*}_{*xh*}_{*xi*}_{*xj*}_{*xk*}_{*xl*}_{*xm*}_{*xn*}_{*xo*}_{*xp*}_{*xq*}_{*xr*}_{*xs*}_{*xt*}_{*xu*}_{*xv*}_{*xw*}_{*xx*}_{*xy*}_{*xz*}_{*ya*}_{*yb*}_{*yc*}_{*yd*}_{*ye*}_{*yf*}_{*yg*}_{*yh*}_{*yi*}_{*yj*}_{*yk*}_{*yl*}_{*ym*}_{*yn*}_{*yo*}_{*yp*}_{*yq*}_{*yr*}_{*ys*}_{*yt*}_{*yu*}_{*yv*}_{*yw*}_{*yx*}_{*yy*}_{*yz*}_{*za*}_{*zb*}_{*zc*}_{*zd*}_{*ze*}_{*zf*}_{*zg*}_{*zh*}_{*zi*}_{*zj*}_{*zk*}_{*zl*}_{*zm*}_{*zn*}_{*zo*}_{*zp*}_{*zq*}_{*zr*}_{*zs*}_{*zt*}_{*zu*}_{*zv*}_{*zw*}_{*zx*}_{*zy*}_{*zz*}

for $k \leftarrow 1$ **to** N **do**

$x[k] \leftarrow R[k]$

return x

Non-adaptive Snapshot (2)

```
procedure updatei(v)  
  ts  $\leftarrow$  ts + 1  
  snapshot  $\leftarrow$  scan()  
  R[i]  $\leftarrow$   $\langle$  ts, v, snapshot  $\rangle$ 
```

Adaptive Update

```
procedure update(v)  
  if myreg =  $\perp$  then  
     $\perp$  myreg  $\leftarrow$  obtain()  
  
  ts  $\leftarrow$  ts + 1  
  snapshot  $\leftarrow$  scan()  
  R[myreg]  $\leftarrow$   $\langle$  ts, v, snapshot  $\rangle$ 
```

Adaptive Scan

upon *scan*;

$t_1 \leftarrow \text{collect}()$, $t_2 \leftarrow t_1$

while *true* **do**

$t_3 \leftarrow \text{collect}()$

if $t_3 = t_2$ **then return** $\langle t_3[1].\text{val}, \dots, t_3[t_3.\text{length}].\text{val} \rangle$

for $k \leftarrow 1$ **to** $t_3.\text{length}$ **do**

if $t_3[k].\text{ts} \geq t_1[k].\text{ts} + 2$ **then return** $t_3[k].\text{snapshot}$

$t_2 \leftarrow t_3$

A Disallowed Solution

```
procedure obtain()  
   $\lfloor$  myreg  $\leftarrow$  C.fetch&inc()  
  
procedure collect()  
  for k  $\leftarrow$  1 to C.read() do  
     $\lfloor$  x[k]  $\leftarrow$  R[k]  
  return x
```

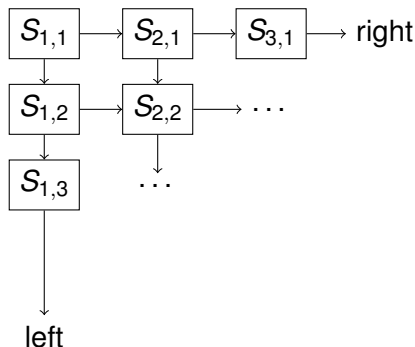
But we can use **only registers!**

The Splitter Object

- One operation: *splitter*
- Returns: *stop*, *left* or *right*
- If a **single** process executes *splitter*, then *stop* is returned.
- If **two or more** processes invoke *splitter*, then not all get the same output.
- At most one process gets *stop*.

Main Idea of Adaptive Snapshot

- Matrix of **registers** and **splitters**
- To obtain a register, a process must find a splitter that returns *stop*.
- Process starts from left top corner and follows the output of splitters.



The Obtain Operation

```
procedure obtain()  
   $x \leftarrow 1, y \leftarrow 1$   
  while true do  
     $s \leftarrow S[x, y].\text{splitter}()$   
    if  $s = \text{"stop"}$  then  $\text{myreg} \leftarrow \langle x, y \rangle$   
    else if  $s = \text{"left"}$  then  $y \leftarrow y + 1$   
    else  $x \leftarrow x + 1$ 
```

The Collect Operation

procedure *collect*

$C \leftarrow \langle \rangle$

$d \leftarrow 1$

while *diagonal d has a splitter that has been traversed* **do**

$C \leftarrow C \cdot \langle$ values of all
non- \perp registers on
diagonal d \rangle

$d \leftarrow d + 1$

return C

