A Solution for the Exercise 2

Michał Kapałka

EPFL, LPD

STiDC'06, 14.XI 2006

Common Mistakes

This should simulate a safe/regular register, but does not make sense (in fact, it's atomic, but why bother with `RW_INIT` then?)

```
macro RW_INIT
begin
  skip;
end macro

macro WRITE(value)
begin
  R := value;
end macro
```

Common Mistakes

Using objects that are not registers / queues:

```
label: r := r + 1;
```

```
label: DEQUEUE(res);
       if res = 1 then R := val; end if;
next:
```

```
macro WRITE(ts, val)
begin
  R[1] := ts;
  R[2] := val;
end macro
```

Common Mistakes

Not exactly a queue:

```
macro DEQUEUE(q)
begin
  if q = 0 then ret := "winner"; q := 1;
  else ret := "loser";
  end if;
end macro
```

Common Mistakes

Other mistakes:

- Statically proposed values
- First dequeue a value, then write to a register
- Returning values of macros/procedures in `V` instead of `V[self]`
- Using global variables as local for many processes without `self`.

Steps

- Slightly modify `update()` – dynamic mapping of processes to registers: `obtain()`.
- `scan()` uses `collect()` $\Rightarrow$ almost no changes.
- Implement `obtain()` using a splitter.
- Implement an adaptive `collect()` operation.
- Implement a splitter using registers.

The Update Operation

procedure `update`($value_i$)

$ts_i \leftarrow ts_i + 1$

$snap_i \leftarrow scan()$

$R[i] \leftarrow [ts \mapsto ts_i, val \mapsto value, snap \mapsto snap_i]$

The Update Operation

procedure `update(`$value_i$`)`

- **if** $myreg_i = \perp$ **then**
  - $myreg_i \leftarrow$ `obtain()`
- $ts_i \leftarrow ts_i + 1$
- $snap_i \leftarrow scan()$
- $R[myreg_i] \leftarrow [ts \mapsto ts_i, val \mapsto value, snap \mapsto snap_i]$

The Scan Operation

procedure `scan()`

$t1_i \leftarrow$ `collect()`, $t2_i \leftarrow t1_i$

while *true* **do**

$t3_i \leftarrow$ `collect()`

if $t3_i = t2_i$ **then return** $\langle t3_i[1].val, \ldots, t3_i[Len(t3_i)].val \rangle$

for $k \leftarrow 1$ **to** $Len(t3_i)$ **do**

if $t3_i[k].ts \geq t1_i[k].ts + 2$ **then return** $t3_i[k].snap$

$t2_i \leftarrow t3_i$

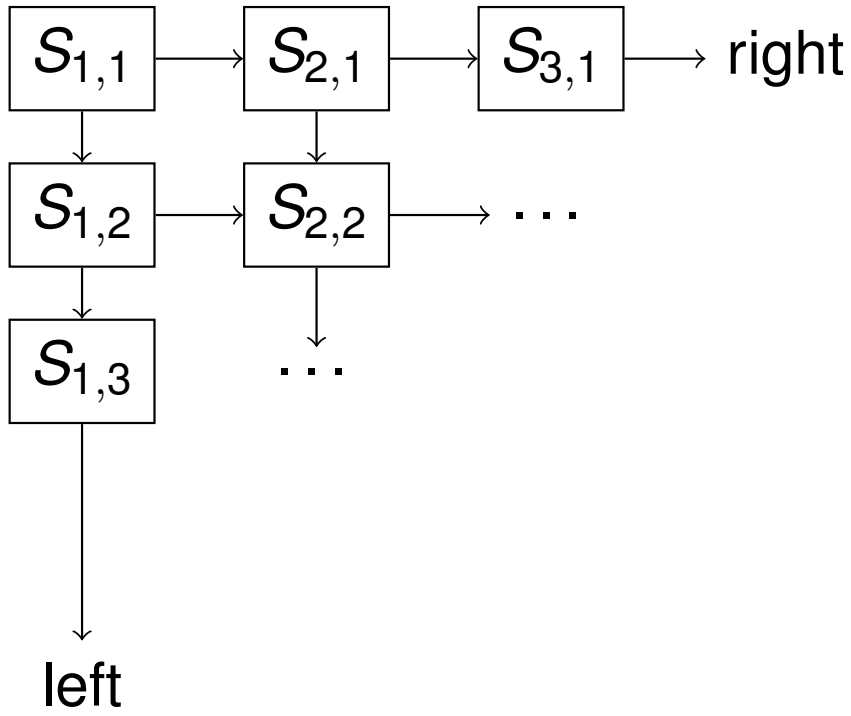
(We assume that $t1_i[k].ts = 0$ if $k > Len(t1_i)$.)

The Splitter Object

- Only 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.

Main Idea of Adaptive Snapshot

- We have a 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()`
- $x_i \leftarrow 1, y_i \leftarrow 1$
- **while** *true* **do**
  - $s_i \leftarrow$ `splitter`$(S[x_i, y_i])$
  - **if** $s_i =$ *"stop"* **then** $myreg_i \leftarrow \langle x_i, y_i \rangle$
  - **else if** $s_i =$ *"left"* **then** $y_i \leftarrow y_i + 1$
  - **else** $x_i \leftarrow x_i + 1$

The Collect Operation

procedure `collect()`

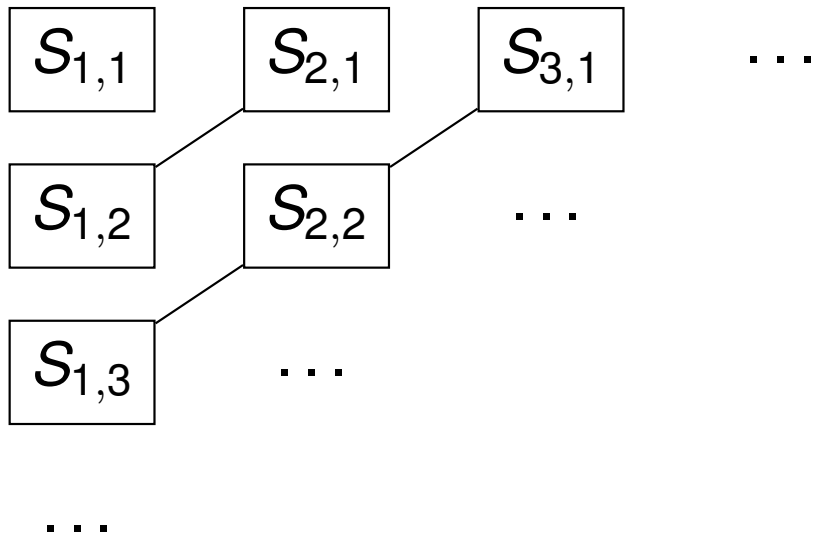
$C_i \leftarrow \langle\rangle$

$d_i \leftarrow 1$

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

$C_i \leftarrow C_i \cdot \langle$ *val* fields of all non-$\perp$ registers on diagonal $d_i \rangle$

$d_i \leftarrow d_i + 1$



An Implementation of a Splitter

procedure `splitter`(S_i)

$S_i.pid \leftarrow i$

if $S_i.flag$ **then return** *"right"*

$S_i.flag \leftarrow true$

if $S_i.pid = i$ **then return** *"stop"*

return *"left"*