

# COMP473: Formal Aspects of Concurrent Systems

Annie Luxton: 300046696

30 October 2003

## Questions

1.

Rewrite the following program that computes the sum  $2^0 + \dots + 2^{N-1}$ , where  $2^k$  is "2 to the power  $k$ " (so we know the required sum is  $2^N - 1$ ) using Action Systems.

The program can be written as:

```
x: integer Initially x=0
cobegin
  x := x+20 || x := x+21 || ... || x := x+2(N-1)
coend
```

where cobegin-coend bracket a set of statements to be executed in parallel, and || separates the parallel statements.

2.

Below is a leader-election algorithm:

```
0..N-1
X: integer 0..N-1
C: array[0..N-1] of 0..2 Initially 0
e: array[0..N-1] of 0..1 Initially 0
```

```
 $E_i(0)$ :
  enable: C[i]=0
  action: X := i; C[i] := 1
```

```
 $E_i(1)$ :
  enable:  $\forall k:: C[k] \geq 1$ 
  action: e[i] := (X=1); C[i] := 2
```

- a) Rewrite the leader-election algorithm using Action Systems and 2 loops.
- b) Rewrite the leader-election algorithm using Action Systems, 2 loops and auxiliary variables to ensure termination.
- c) Rewrite the leader-election algorithm using Action Systems, 1 loop and still ensuring termination using auxiliary variables.
- d) Rewrite the leader-election algorithm using Action Systems, 1 loop and auxiliary variables to ensure atomicity of each separate statement and that they are still executed in the correct order, as well as termination.
- e) Rewrite the leader-election algorithm using Action Systems, 1 loop and ensure termination without the addition of any auxiliary variables.

### 3.

Prove the relationship between different versions of the leader-election algorithm. In particular, prove the relationship between the version of the leader-election algorithm that uses 2 loops and the version of the leader-election algorithm that uses only 1 loop. Do this by showing the relationship between the following:

- a) The version of the leader-election algorithm that uses 2 loops (**2.a**) and the version of the leader-election algorithm that uses 2 loops and also ensures termination using auxiliary variables (**2.b**).
- b) The version of the leader-election algorithm that uses 2 loops and also ensures termination using auxiliary variables (**2.b**) and the version of the leader-election algorithm that uses only 1 loop, still ensuring termination using auxiliary variables (**2.c**).

# Answers

## 1.

Below is the program that computes the sum  $2^0 + \dots + 2^{N-1}$ , where  $2^k$  is "2 to the power  $k$ " written using Action Systems:

```
[var b:array[0..N-1] of 0..1;
  || i : 0..N-1 : b[i] := 0 || ;
  x := 0;
  *[i : 0..N-1 : b[i] = 0 → x := x + 2i; b[i] := 1]
] : x
```

## 2.

### 2.a)

Below is the leader-election algorithm written using Action Systems and 2 loops:

```
[var C:array[0..N-1] of 0..2;
  X : 0..N-1;
  || i : 0..N-1 : C[i] := 0 || ;
  || i : 0..N-1 : e[i] := false || ;
  *[i : 0..N-1 : C[i] = 0 → X := i; C[i] := 1];
  *[j : 0..N-1 : (∧ k : 0..N-1 : C[k] ≥ 1) → e[j] := (X = j); C[j] := 2]
] : e
```

### 2.b)

Below is the leader-election algorithm written using Action Systems, 2 loops and auxiliary variables to ensure termination:

```
[var C:array[0..N-1] of 0..2;
  X : 0..N-1;
  l:array[0..N-1] of 0..2;
  || i : 0..N-1 : C[i] := 0 || ;
  || i : 0..N-1 : e[i] := false || ;
  || i : 0..N-1 : l[i] := 0 || ;
  *[i : 0..N-1 : C[i] = 0 ∧ l[i] = 0 → X := i; C[i] := 1; l[i] := 1];
  *[j : 0..N-1 : (∧ k : 0..N-1 : C[k] ≥ 1) ∧ l[j] = 1 → e[j] := (X = j); C[j] := 2; l[j] := 2]
] : e
```

**2.c)**

Below is the leader-election algorithm written using Action Systems, only 1 loop and auxiliary variables to ensure termination:

```

[var C:array[0..N - 1] of 0..2;
  X : 0..N - 1;
  l:array[0..N - 1] of 0..2;
  || i : 0..N - 1 : C[i] := 0 || ;
  || i : 0..N - 1 : e[i] := false || ;
  || i : 0..N - 1 : l[i] := 0 || ;
  *
    [i : 0..N - 1 : C[i] = 0 ∧ l[i] = 0 → X := i; C[i] := 1; l[i] := 1]
    []
    [j : 0..N - 1 : (∧ k : 0..N - 1 : C[k] ≥ 1) ∧ l[j] = 1 → e[j] := (X = j); C[j] := 2; l[j] := 2]
  ]
]: e

```

**2.d)**

Below is the leader-election algorithm written using Action Systems, only 1 loop and auxiliary variables to ensure atomicity of each separate statement and that they are still executed in the correct order, as well as termination:

```

[var C:array[0..N - 1] of 0..2;
  X : 0..N - 1;
  l:array[0..N - 1] of 0..2;
  || i : 0..N - 1 : C[i] := 0 || ;
  || i : 0..N - 1 : e[i] := false || ;
  || i : 0..N - 1 : li := 0 || ;
  *
    [i : 0..N - 1 : C[i] = 0 ∧ li = 0 → X := i; li := 1]
    []
    [i : 0..N - 1 : C[i] = 0 ∧ li = 1 → C[0] := 1; li := 2]
    []
    [j : 0..N - 1 : (∧ k : 0..N - 1 : C[k] ≥ 1) ∧ lj = 2 → e[j] := (X = j); lj := 3]
    []
    [j : 0..N - 1 : (∧ k : 0..N - 1 : C[k] ≥ 1) ∧ lj = 3 → C[j] := 2; lj := 4]
  ]
]: e

```

2.e)

Below is the leader-election algorithm written using Action Systems, only 1 loop while still ensuring termination, this time without the addition of any auxiliary variables:

```
[var C:array[0..N - 1] of 0..2;
  X : 0..N - 1;
  || i : 0..N - 1 : C[i] := 0 || ;
  || i : 0..N - 1 : e[i] := false || ;
  *[
    [i : 0..N - 1 : C[i] = 0 → X := i; C[i] := 1]
    []
    [j : 0..N - 1 : (∧ k : 0..N - 1 : C[k] ≥ 1) ∧ e[X] = false → e[j] := (X = j); C[j] := 2]
  ]
]: e
```

### 3.

#### 3.a)

Proof of the relationship between the version of the leader-election algorithm that uses 2 loops (**2.a**) and the version of the leader-election algorithm that uses 2 loops but ensures termination using auxiliary variables (**2.b**). This requires showing that the algorithm in **2.a**) can be refined into the algorithm in **2.b**) using Back's rules, in particular Rule 8.

**Rule 8: Adding auxilliary variables:**

$$S \equiv [\text{var } x; S[x := h_1/\text{skip}^1, \dots, x := h_n/\text{skip}^n]] \quad (1)$$

To use this rule, we must show the following:

1.  $S$  does not contain any occurrence of  $x$ .

Proof:

In this case:

$$\begin{aligned} S \hat{=} & \\ & [\text{var } C:\text{array}[0..N-1] \text{ of } 0..2; \\ & \quad X : 0..N-1; \\ & \quad \| i : 0..N-1 : C[i] := 0 \| ; \\ & \quad \| i : 0..N-1 : e[i] := \text{false} \| ; \\ & \quad * [i : 0..N-1 : C[i] = 0 \longrightarrow X := i; C[i] := 1]; \\ & \quad * [j : 0..N-1 : (\wedge k : 0..N-1 : C[k] \geq 1) \longrightarrow e[j] := (X = j); C[j] := 2] \\ & ] : e \end{aligned}$$

And variables  $x$  and  $y$ :

$$\begin{aligned} x \hat{=} & C : \text{array}, X : \text{int} \\ y \hat{=} & C' : \text{array}, X' : \text{int}, l : \text{array} \end{aligned}$$

We can assume that initially we have a statement  $[\text{var } x; S]$  and we want to replace variables  $x$  with some other variables  $y$ , changing  $S$  to  $S'$  accordingly, such that:

$$[\text{var } x; S] \leq [\text{var } y; S']$$

This can be achieved by completing the following sequence of steps:

1. *Introduce new variables y:*

$$\text{Let } S_1 = S[i : 0..N - 1 : C'[i] := e_1/skip^1, X' := e_2/skip^2, i : 0..N - 1 : l[i] := e_3/skip^3]$$

where  $e_1, e_2, e_3$  depend on where in the algorithm they appear.

Applying Rule 8 and monotonicity of refinement, we have:

$$[var\ x; S] \leq [var\ x; [var\ y; S']]$$

2. *Introduce context assertion relating x and y:*

Apply context introduction of assertions on  $x$  and  $y$ . This gives us:

$$S_1 \leq S_1[\{l[i] = C[i] = 0\}; T_0/T_0, \{l[i] = C[i] = 1\}; T_1/T_1, \{l[i] = C[i] = 2\}; T_2/T_2]$$

where:

$$\begin{aligned} T_0 &\hat{=} \| i : 0..N - 1 : C[i] := 0 \| ; \| i : 0..N - 1 : e[i] := false \| && \text{(Initialisation)} \\ T_1 &\hat{=} *[i : 0..N - 1 : C[i] = 0 \longrightarrow X := i; C[i] := 1] && \text{(First loop)} \\ T_2 &\hat{=} *[j : 0..N - 1 : (\wedge k : 0..N - 1 : C[k] \geq 1) \longrightarrow e[j] := (X = j); C[j] := 2] && \text{(Second loop)} \end{aligned}$$

3. *Replace statements on x with statements on y:*

Let  $T'_0, T'_1, T'_2$  be statements that do not refer to variables in  $x$ :

$$\begin{aligned} T'_0 &\hat{=} \| i : 0..N - 1 : C'[i] := 0 \| ; \| i : 0..N - 1 : e[i] := false \| ; \\ &\quad \| i : 0..N - 1 : l[i] := 0 \| && \text{(Initialisation)} \\ T'_1 &\hat{=} *[i : 0..N - 1 : C'[i] = 0 \wedge l[i] = 0 \longrightarrow X' := i; C'[i] := 1; l[i] := 1] && \text{(First loop)} \\ T'_2 &\hat{=} *[j : 0..N - 1 : (\wedge k : 0..N - 1 : C'[k] \geq 1) \wedge l[j] = 1 \longrightarrow e[j] := (X' = j); \\ &\quad C'[j] := 2; l[j] := 2] && \text{(Second loop)} \end{aligned}$$

(Informally) using weakest preconditions, we can show that:

- $\{l[i] = C[i] = 0\}; T_0 \leq T'_0$

Must show that  $wp(T_0, R) \Rightarrow wp(T'_0, R)$ :

$$\begin{aligned}
wp(T_0, R) &= wp(\| i : 0..N - 1 : C[i] := 0 \| ; \| i : 0..N - 1 : e[i] := false \|, \\
&\quad \sum_{i=0}^{N-1} C[i] = 0 \wedge \sum_{i=0}^{N-1} e[i] = 0) \\
&= wp(\| i : 0..N - 1 : C[i] := 0 \|, wp(\| i : 0..N - 1 : e[i] := false \|, \\
&\quad \sum_{i=0}^{N-1} C[i] = 0 \wedge \sum_{i=0}^{N-1} e[i] = 0)) \\
&= wp(\| i : 0..N - 1 : C[i] := 0 \|, \sum_{i=0}^{N-1} C[i] = 0 \wedge \sum_{i=0}^{N-1} 0 = 0) \\
&= \sum_{i=0}^{N-1} 0 = 0 \wedge \sum_{i=0}^{N-1} 0 = 0
\end{aligned}$$

$$\begin{aligned}
wp(T'_0, R) &= wp(\| i : 0..N - 1 : C'[i] := 0 \| ; \| i : 0..N - 1 : e[i] := false \| ; \\
&\quad \| i : 0..N - 1 : l[i] := 0 \|, \sum_{i=0}^{N-1} C'[i] = 0 \wedge \sum_{i=0}^{N-1} e[i] = 0) \\
&= wp(\| i : 0..N - 1 : C'[i] := 0 \| ; \| i : 0..N - 1 : e[i] := false \|, \\
&\quad wp(\| i : 0..N - 1 : l[i] := 0 \|, \sum_{i=0}^{N-1} C'[i] = 0 \wedge \sum_{i=0}^{N-1} e[i] = 0)) \\
&= wp(\| i : 0..N - 1 : C'[i] := 0 \| ; \| i : 0..N - 1 : e[i] := false \|, \\
&\quad \sum_{i=0}^{N-1} C'[i] = 0 \wedge \sum_{i=0}^{N-1} e[i] = 0) \\
&= wp(\| i : 0..N - 1 : C'[i] := 0 \|, wp(\| i : 0..N - 1 : e[i] := false \|, \\
&\quad \sum_{i=0}^{N-1} C'[i] = 0 \wedge \sum_{i=0}^{N-1} e[i] = 0)) \\
&= wp(\| i : 0..N - 1 : C'[i] := 0 \|, \sum_{i=0}^{N-1} C'[i] = 0 \wedge \sum_{i=0}^{N-1} 0 = 0) \\
&= \sum_{i=0}^{N-1} 0 = 0 \wedge \sum_{i=0}^{N-1} 0 = 0
\end{aligned}$$

Therefore,  $wp(T_0, R) \Rightarrow wp(T'_0, R)$  and thus  $\{l[i] = C[i] = 0\}; T_0 \leq T'_0$ .

- $\{l[i] = C[i] = 1\}; T_1 \leq T'_1$

Must show that  $wp(T_1, R) \Rightarrow wp(T'_1, R)$ :

$$\begin{aligned}
wp(T_1, R) &= wp(*[i : 0..N - 1 : C[i] = 0 \longrightarrow X := i; C[i] := 1], \\
&\quad \sum_{i=0}^{N-1} C[i] = N \wedge \sum_{i=0}^{N-1} e[i] = 0) \quad \text{(At termination of loop)} \\
&= \sum_{i=0}^{N-1} 1 = N \wedge \sum_{i=0}^{N-1} e[i] = 0
\end{aligned}$$

$$\begin{aligned}
wp(T'_1, R) &= wp(*[i : 0..N - 1 : C'[i] = 0 \wedge l[i] = 0 \longrightarrow X' := i; C'[i] := 1; l[i] := 1], \\
&\quad \sum_{i=0}^{N-1} C'[i] = N \wedge \sum_{i=0}^{N-1} e[i] = 0) \quad \text{(At termination of loop)} \\
&= \sum_{i=0}^{N-1} 1 = N \wedge \sum_{i=0}^{N-1} e[i] = 0
\end{aligned}$$

Therefore,  $wp(T_1, R) \Rightarrow wp(T'_1, R)$  and thus  $\{l[i] = C[i] = 1\}; T_1 \leq T'_1$ .

- $\{l[i] = C[i] = 2\}; T_2 \leq T'_2$

Must show that  $wp(T_2, R) \Rightarrow wp(T'_2, R)$

$$\begin{aligned}
& wp(T_2, R) \\
&= * [j : 0..N-1 : (\wedge k : 0..N-1 : C[k] \geq 1) \longrightarrow e[j] := (X = j); C[j] := 2], \\
&\quad \sum_{j=0}^{N-1} C[j] \geq 2N \wedge \sum_{j=0}^{N-1} e[j] = 1 \wedge e[X] := 1 \quad (\text{At termination of loop}) \\
&= \sum_{j=0}^{N-1} 2 \geq 2N \wedge \sum_{j=0}^{N-1} (X = j) = 1 \wedge e[X] := 1
\end{aligned}$$

$$\begin{aligned}
& wp(T'_2, R) \\
&= wp(* [j : 0..N-1 : (\wedge k : 0..N-1 : C'[k] \geq 1) \wedge l[j] = 1 \longrightarrow e[j] := (X' = j); \\
&\quad C'[j] := 2; l[j] := 2], \\
&\quad \sum_{i=0}^{N-1} C'[i] \geq 2N \wedge \sum_{i=0}^{N-1} e[i] = 0 \wedge e[X'] := 1) \quad (\text{At termination of loop}) \\
&= \sum_{i=0}^{N-1} 2 = 2N \wedge \sum_{i=0}^{N-1} (X' = j) = 1 \wedge e[X'] := 1
\end{aligned}$$

Therefore,  $wp(T_2, R) \Rightarrow wp(T'_2, R)$  and thus  $\{l[i] = C[i] = 2\}; T_2 \leq T'_2$ .

By transitivity of refinement, we now have:

$$S_1 \leq S_1[T'_0/T_0, T'_1/T_1, T'_2/T_2] = S_2$$

#### 4. Remove old variables $x$ :

Assume that all the remaining occurrences of  $x$  are now in assignments to variables in  $x$ :

$$S_2 = S'_2[x := h_1/skip^1, \dots, x := h_m/skip^m]$$

where  $S'_2$  does not contain any occurrences of  $x$ .

Therefore, we have:

$$\begin{aligned}
S'_2 \cong & \\
& [\text{var } C':\text{array}[0..N-1] \text{ of } 0..2; \\
& \quad X' : 0..N-1; \\
& \quad l:\text{array}[0..N-1] \text{ of } 0..2; \\
& \quad \| i : 0..N-1 : C'[i] := 0 \| ; \\
& \quad \| i : 0..N-1 : e[i] := \text{false} \| ; \\
& \quad \| i : 0..N-1 : l[i] := 0 \| ; \\
& \quad * [i : 0..N-1 : C'[i] = 0 \wedge l[i] = 0 \longrightarrow X' := i; C'[i] := 1; l[i] := 1]; \\
& \quad * [j : 0..N-1 : (\wedge k : 0..N-1 : C'[k] \geq 1) \wedge l[j] = 1 \longrightarrow \\
& \quad \quad e[j] := (X' = j); C'[j] := 2; l[j] := 2] \\
& ] : e
\end{aligned}$$

Applying Rule 8 again then gives us:

$$\begin{aligned}
[\text{var } x; [\text{var } y; S_1]] & \leq [\text{var } x; [\text{var } y; S_2]] \\
& \equiv [\text{var } y; [\text{var } x; S_2]] \\
& \equiv [\text{var } y; [\text{var } x; S'_2[x := h_1/\text{skip}^1, \dots, x := h_m/\text{skip}^m]]] \\
& \equiv [\text{var } y; S'_2]
\end{aligned}$$

Transitivity of refinement then gives us the desired result:

$$[\text{var } x; S] \leq [\text{var } y; S'_2]$$

That is, by removing variables  $x$  and adding variables  $y$  using Rule 8, followed by a simple renaming of variables  $C'$  back to  $C$  and  $X'$  back to  $X$ , we have essentially converted the algorithm in **2.a**) into the algorithm in **2.b**).

### 3.b)

Proof of the relationship between the version of the leader-election algorithm that uses 2 loops and auxiliary variables to ensure termination (**2.b**), and the version of the leader-election algorithm that uses 1 loop and still ensures termination (**2.c**). This requires showing that the algorithm in **2.b** can be refined into the algorithm in **2.c** using Back's rules, in particular Rule 4.

#### Rule 4: Merging a sequence of loops:

$$*[i : A_i]; * [j : B_j] \leq *[i : A_i \parallel j : B_j] \quad (2)$$

To use this rule, we must show the following:

1. For every  $i, j$ ,  $B_j$  cannot enable or disable  $A_i$
2. For every  $i, j$ , either  $B_j$  cannot precede  $A_i$  or  $A_i$  commutes with  $B_j$
3.  $*[j : B_j]$  terminates when  $\vee gA_i$

#### Proof:

In this case, for  $i \in 0..N - 1$ :

$$A_i \hat{=} C[i] = 0 \wedge l[i] = 0 \longrightarrow X := i; C[i] := 1; l[i] := 1$$

Also, for  $j \in 0..N - 1$ :

$$B_j \hat{=} (\wedge k : 0..N - 1 : C[k] \geq 1) \wedge l[j] = 1 \longrightarrow e[j] := (X = j); C[j] := 2; l[j] := 2$$

1. For every  $i, j$ ,  $B_j$  cannot enable or disable  $A_i$

- To enable  $A_i$  would require that both  $C[i]$  and  $l[i] = 0$  be set to 0, that is, that the statements  $C[i] := 0$  and  $l[i] := 0$  be in the body of some action  $B_j$ . But, no  $B_j$  contains neither the statement  $C[i] := 0$  nor  $l[i] := 0$ . Therefore, no  $B_j$  can enable any  $A_i$ .
- To disable  $A_i$  would require that  $C[i]$  be changed from 0 to another value, that is, that  $C[i] = 0$  be changed to  $C[i] > 0$  by a statement in  $B_j$ , or that  $l[i]$  be changed from 0 to another value, that is, that  $l[i] = 0$  be changed to  $l[i] > 0$  by a statement in  $B_j$ . The statements  $C[j] := 2$  and  $l[j] := 2$  are in fact within the body of every  $B_j$  action except that in order for these actions to be triggered,  $B_j$ 's guard must be satisfied. Every  $B_j$ 's guard requires that for every  $k \in 0..N - 1$ ,  $C[k] \geq 1$  and that  $l[j] = 1$ . This means that every  $A_i$  must have been run and now be disabled since every  $A_i$ 's guard requires that  $C[i] = 0$  and  $l[i] = 0$  and every  $A_i$  action body contains the statement  $C[i] := 1$  and  $l[i] := 1$ , thus disabling itself when it runs. Therefore, the enabling condition for  $B_j$  effectively requires that all  $A_i$ s must have already been executed and be disabled. Therefore, no  $B_j$  can disable any  $A_i$  as they are already all disabled.

Therefore, for every  $i, j$ ,  $B_j$  cannot enable or disable  $A_i$ .

2. For every  $i, j$ , either  $B_j$  cannot precede  $A_i$  or  $A_i$  commutes with  $B_j$

Every  $B_j$ 's guard requires that all for all  $k \in 0..N - 1$ ,  $C[k] \geq 1$  and  $l[j] = 1$ . Therefore, every  $B_j$ 's enabling condition essentially means that no  $A_i$  can be enabled. Running a  $A_i$  action disables itself because it contains the statements  $C[i] := 1$  and  $l[i] := 1$ , negating it's enabling condition that  $C[i] = 0$  and  $l[i] = 0$ . Therefore, for a  $B_j$  action to fire requires that for all  $i \in 0..N - 1$ , action  $A_i$  has already happened. Therefore, no  $B_j$  can precede any  $A_i$ .

3.  $*[j : B_j]$  terminates when  $\vee gA_i$

This property insists that when even just one action  $A_i$  is or somehow becomes enabled, that if  $B_j$  actions are being undertaken, that  $*[j : B_j]$  terminate. Since every  $B_j$ 's guard requires that all for all  $k \in 0..N - 1$ ,  $C[k] \geq 1$  and  $l[j] = 1$ , then obviously no more  $B_j$  actions could run if even just one  $A_i$  action were enabled as this would mean that for some  $i \in 0..N - 1$ ,  $C[i] = 0$  nor  $l[i] = 0$ , thus disabling all  $B_j$  actions. Therefore, as long as even just one  $A_i$  were or became enabled somehow,  $*[j : B_j]$  would need to terminate.

That is, by merging the two loops using Rule 4, we have essentially converted the algorithm in **2.b)** into the algorithm in **2.c)**.