Calvin Passmore

A02107892

ECE 6790

Homework 2

Exercise 2.20:

Design a 1-bit synchronous adder *1BitAdder* by composing instances of And, Or, Not, and Xor gates. The component *1BitAdder* has three input variables, *x*, *y*, and *carry-in*, and two output variables *z* and *carry-out*, where *z* is the least significant bit, should equal the sum of the values of the three input variables. Then, design a 3-bit synchronous adder *3BitAdder* by composing three instances of the component *1BitAdder*. The component *3BitAdder* has input variables x_0, x_1, x_2, y_0, y_1, y_2, and *carry_in* and has output variables z_0, z_1, z_2, and *carry_out* should equal the sum of the 3-bit number encoded by the input variables x_0, x_1, x_2, and the 3-bit number encoded by the input variables y_0, y_1, y_2, and the input value of *carry_in*.

1Bit Adder

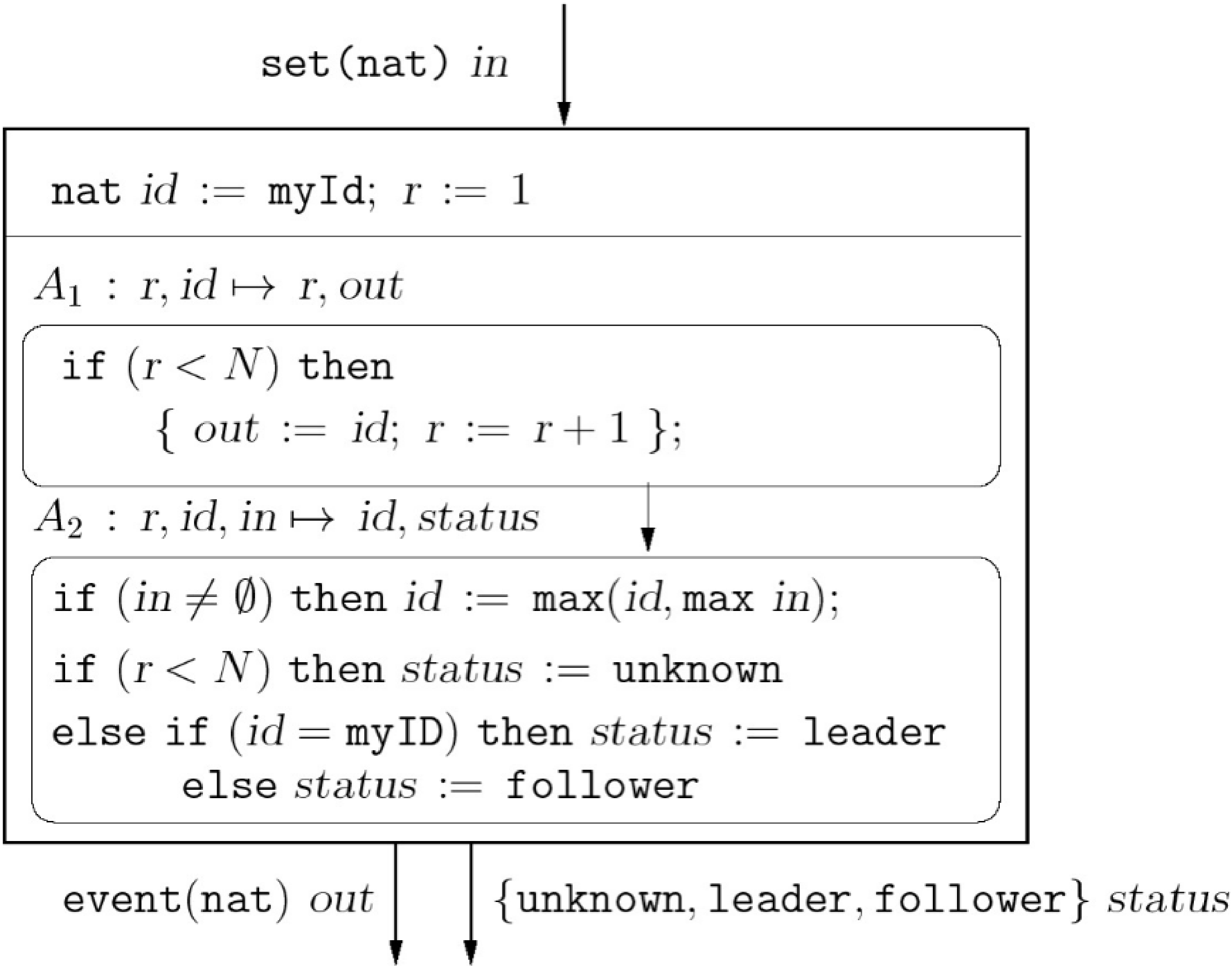
```
z = x XOR y XOR carry_in
carry_out = (x AND y) OR (x AND carry_in) OR (y AND carry_in)
```

3Bit Adder

```
1BitAdder0 = [x->x0][y->y0][z->z0][carry_in->carry_in ][carry_out-
>carry_in1]
1BitAdder1 = [x->x1][y->y1][z->z1][carry_in->carry_in1][carry_out-
>carry_in2]
1BitAdder1 = [x->x2][y->y2][z->z2][carry_in->carry_in2][carry_out-
>carry_out]
```

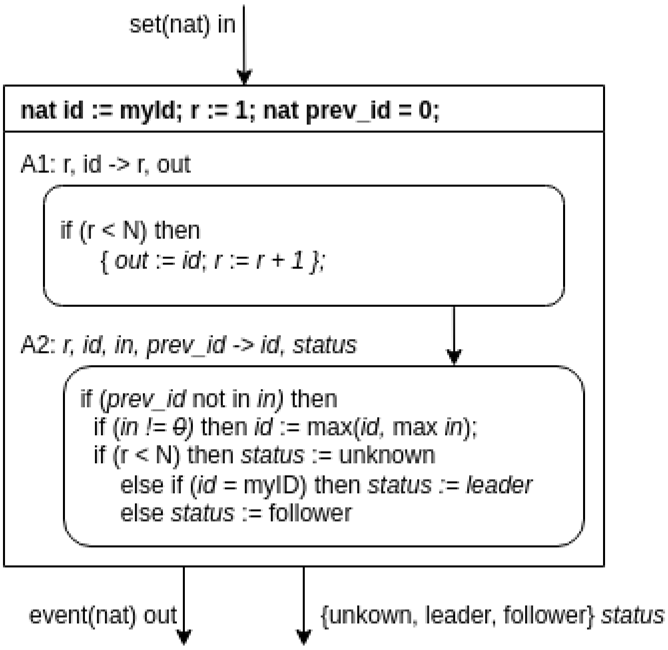
Exercise 2.22:

Consider the leader election algorithm in synchronous networks. Argue that if the value of *id* does not charge in a given round, then there is no need to send it in the following round (that is, the output *out* can be absent in the next round). This can reduce the number of messages sent. Modify the description of the component *SyncLENode* to implement this change.



If I the node receive the same value in two consecutive rounds, there is not point in sending the value again. It cannot be higher than itself, so sending it again doesn't change who is going to be elected leader.

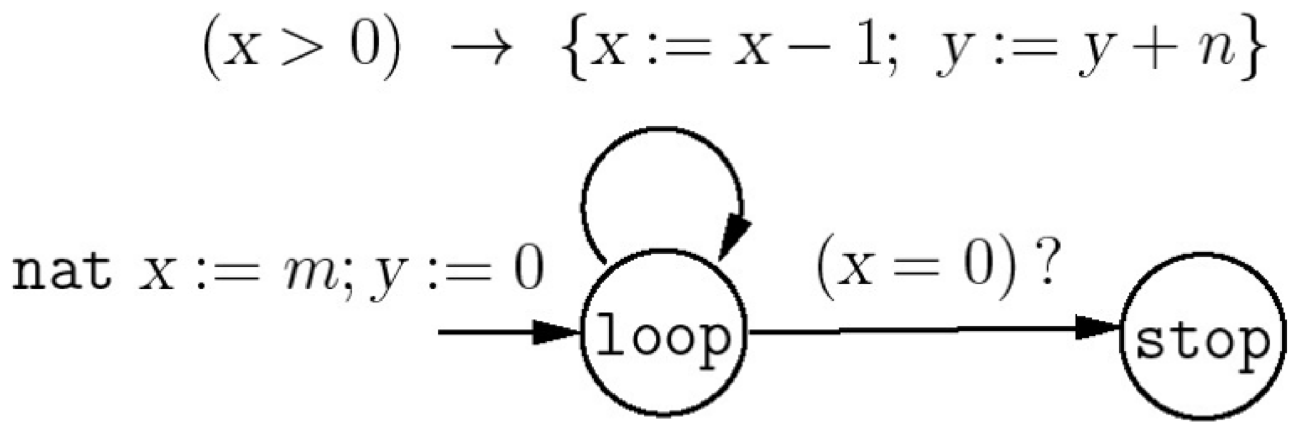
Modifications to *SyncLENode*:


set(nat) in
nat id := myId; r := 1; nat prev_id = 0;
A1: r, id -> r, out
if (r < N) then
{ out := id; r := r + 1 };
A2: r, id, in, prev_id -> id, status
if (prev_id not in in) then
if (in != ∅) then id := max(id, max in);
if (r < N) then status := unknown
else if (id = myID) then status := leader
else status := follower
event(nat) out
{unkown, leader, follower} status

Exercise 3.1:

Given two natural numbers *m* and *n*, consider the program Mult that multiplies the input numbers using two variables *x* and *y*, of type nat, as shown in the below figure. Describe the transition system *Mult*(m,n) that captures the behavior of this program on input numbers *m* and *n*, that is, describe the states, initial states, and transitions. Argue that when the value of the variable *x* is 0, the value of the variable *y* must equal the product of the input numbers *m* and *n*, that is, the following property is an invariant of this transition system:

```
(mode = stop) -> (y = m*n)
```



Initial States = {x = m; y = 0;}

States = {loop, stop}

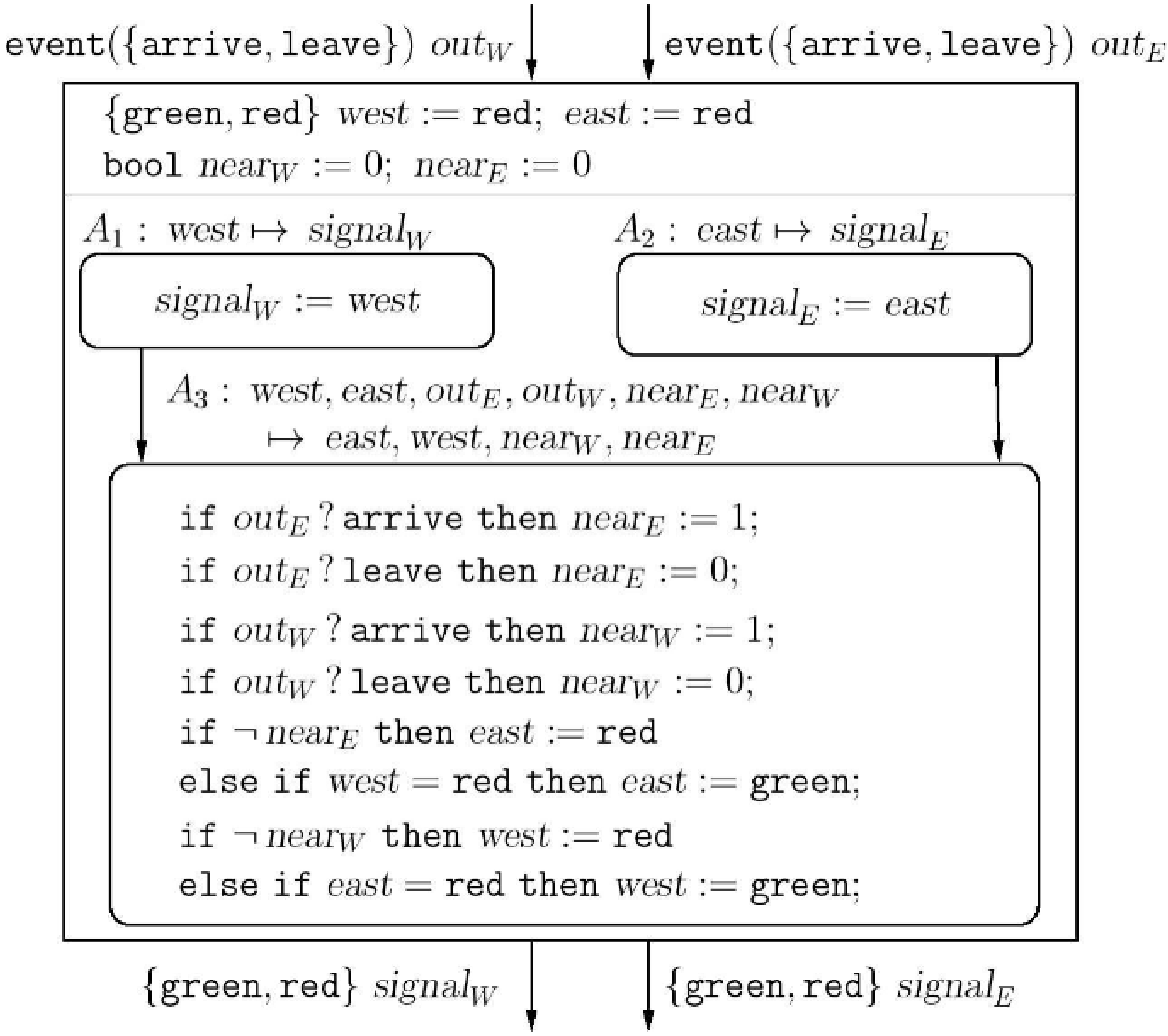
Transitions

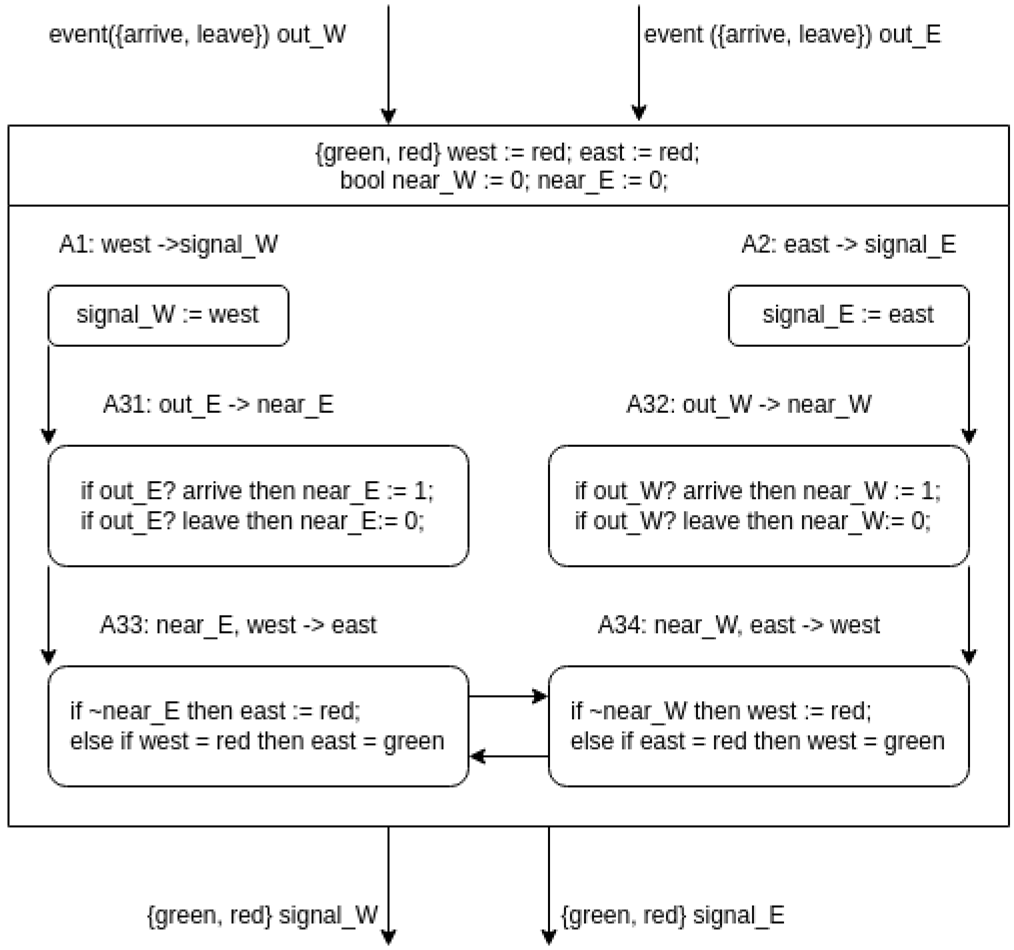
```
if x > 0 then
    x := x + 1;
    y := y + n;
    states := loop;
else
    states := stop
```

When x = 0, y must be the product of *m* and *n* because that is the definition of multiplication. Taking a number (*n*) and adding it to itself a number of times (*m*). x is set to the number of times and y is counting the running total.

Exercise 3.3:

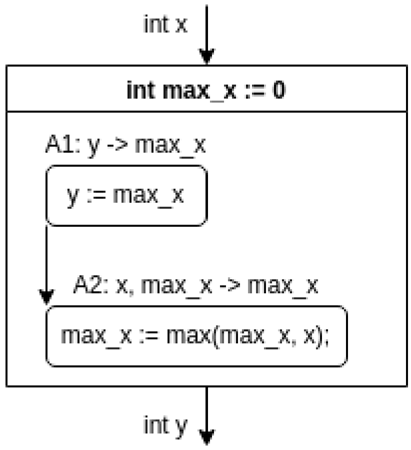
This reaction description for the controller *Controller2* consists of three tasks as shown below. *Split* the task A_3 into four tasks, each of which writes exactly on of the state variables *east*, *west*, $near_W$, and $near_E$. Each task should be described by its read-set, write-set, and update code, along with the necessary precedence constraints. The revised description should have the same set of reactions as the original description. Does this splitting impact output/input await dependencies? If not, what would be the potential benefits and/or drawbacks of the revised description compared to the original description?




event({arrive, leave}) out_W
event ({arrive, leave}) out_E
{green, red} west := red; east := red;
bool near_W := 0; near_E := 0;
A1: west ->signal_W
signal_W := west
A2: east -> signal_E
signal_E := east
A31: out_E -> near_E
if out_E? arrive then near_E := 1;
if out_E? leave then near_E:= 0;
A32: out_W -> near_W
if out_W? arrive then near_W := 1;
if out_W? leave then near_W:= 0;
A33: near_E, west -> east
if ~near_E then east := red;
else if west = red then east = green
A34: near_W, east -> west
if ~near_W then west := red;
else if east = red then west = green
{green, red} signal_W
{green, red} signal_E

Exercise 3.4:

Consider a component *C* with an output variable *x* of type *int*. Design a safety monitor to capture the requirement that the sequence of values output by the component *C* is strictly increasing (that is, the output in each round should be strictly greater than the output in the preceding round).



Exercise 3.6:

Consider a transition system *T* with two integer variables *x* and *y* and a Boolean variable *z*. All the variables are initially 0. The transitions of the system corresponding to executing the conditional statement.

```
if (z = 0) then {x := x + 1; z := 1} else {y := y + 1; z := 0}
```

Consider the property Φ given by *(x = y) || (x = y + 1)*. Is Φ and invariant of the transition system *T*? Is Φ an inductive invariant of the transition system *T*? Find a formula Ψ such that Ψ is stronger than Φ and is an inductive invariant of the transition system *T*. Justify your answers.

Φ is an invariant of the system.

```
x = 0; y = 0; z = 0;  <- satisfies x = y
x = 1; y = 0; z = 1;  <- satisfies x = y + 1
x = 1; y = 1; z = 0;  <- satisfies x = y
```

At this point, we are back to the starting configuration, with no change for divergence, therefore Φ is an invariant.

However, Φ is not an inductive invariant:

```
x = 1; y = 0; z = 0;  <- satisfies x = y
x = 2; y = 0; z = 1;  <- ERROR
```

To strengthen Φ add *(if x = y then z := 0 else z := 1)*, I will call this Φ_2. Φ_2 is an invariant, but isn't inductive following the same proofs as above. Let Ψ = Φ AND Φ_2.

Ψ = *(x = y) || (x = y + 1)* AND *(if x = y then z := 0 else z := 1)*.

```
x = 1; y = 0; z = 1;    <- satisfies x = y + 1
x = 1; y = 1; z = 0;    <- satisfies x = y
```

Exercise 3.7:

Recall that the transition system *Mult*(m,n) from exercise 3.1. First, show that the invariant property (mode = *stop*) -> (y = m * n) is not an inductive invariant. Then find a stronger property that is an inductive invariant. Justify your answers.

```
m = 2; n = 4;
x = 1; y = 0; mode = loop;
x = 0; y = 4; mode = stop;  <- ERROR
```

Ψ = *(mode = stop) -> (y = m * n)* AND *(if y = 0 then x = m and mode = loop)*

```
m = 2; n = 4;
x = 2; y = 0; mode = loop;
x = 1; y = 4; mode = loop;
x = 0; y = 8; mode = loop;
x = 0; y = 8; mode = stop;
```
