Exercises

Verification Technology 2008

February 13, 2009

1 Modeling

Exercise 1.1 Assume that we model a system which interacts with the environment via variables *req*, *conf*, and *warning*. We wish to verify the following property:

At each state where *req* is true, it is either the case that in some following state either

- the variable *conf* is true, or
- the variable *warning* becomes true and stays true for the entire remainder of the computation.

Your problem is

a) to describe how to express this property in temporal logic,

b) Assume that we have a PROMELA model of the system. What additional code do you add in order to use SPIN to validate that the above property holds for the system.

Exercise 1.2 (Exam 2007) Assume that you have constructed a large Promela model with many processes, where `x` is a global variable, and where one of the processes, `A`, contains a sequence of statements. I.e., `A` can be declared by something like

```
        byte x;
/* other declarations (not shown here) */

active proctype A() {

           stmt1 ;
           stmt2 ;
            ...
            ...
           stmtn
}

            ...
/* definitions of other processes (not shown here) */
```

We want to investigate whether other processes can possibly modify the variable `x` when process `A()` is somewhere between `stmt1` and `stmtn` (a possible reason could be that `stmt1` could be some statement that imitates setting of some kind of lock on `x`, which is released by `stmtn`, and that we would like to check whether this "lock discipline" actually works). Show how to add some Promela code to process `A` and to add a monitor process to check whether some other process can possibly modify `x` when process `A()` is somewhere between `stmt1` and `stmtn`.

2 Specifying Properties, e.g., in Temporal Logic

Exercise 2.1 (reexam 2005) Consider the following action system.

```
Action System  Program 1
  variables
      $x, y$ : integer
  initially
      $x = y = 0$
  actions
      $x := x + 1$
    $[\!]$
      $x > y \longrightarrow x := x - 1$
    $[\!]$
      $y < x \longrightarrow y := y + 1$
end
```

Assume that there are three weak fairness constraints: one for each of the three actions. Which of the following properties hold for the program:

a) $\Box\, x > y$

b) $\Box\, x \geq y$

c) $x \geq y \;\mathcal{W}\; x < y$

d) $\Diamond\, x \geq 10$

e) $\Box \Diamond\, x = y$

Exercise 2.2 (exam 2006) Two processes, `A` and `B`, access a shared resource. A part of a Promela model is given below.

```
/* declarations (not shown here) */

active proctype A() {
again:  /* some statements */

          ...
crit:   /* here is the access to the shared resource */

        /* some statements */

        goto again
}

active proctype B() {
again:  /* some statements */

        ...

crit:   /* here is the access to the shared resource */

        /* some statements */

        goto again
}
```

Formulate temporal logic formulas, which express the following properties:

a) The processes `A` and `B` alternate in accessing the file.

b) The processes `A` and `B` do not access the file at the same time.

You should **not** add extra variables to the Promela model, just write temporal logic formulas. Do not worry about whether or not the properties are satisfied by the model. Just express them.

Exercise 2.3 Give temporal logic formulas that express the following properties, for state formula p, q, and r.

a) Whenever p holds at position i, then it also holds at every position $j \geq i$ of a computation.

b) Whenever p holds at position i, then it also holds at position $i+1$ of a computation.

c) The first position in which q is true must coincide with or be preceded by a p-position.

d) The first position in which q is true must be preceded by a p-position in which q is not true.

Exercise 2.4 (Exam 2000) The following is an action system with two integer variables

x and y. It does not perform too many useful things.

```
**Action System** *Foo1*
  **variables**
      $x, y$ : integer
  **initially**
      $x = y = 0$
  **actions**
      $x \le y \longrightarrow x := x + 1$
    $[\!]$
      $x \ge y \longrightarrow y := y + 1$
    $[\!]$
      $x > y + 1 \longrightarrow y := y + 10$
**end**
```

For each of the three actions, there is a (weak) fairness constraint, defined by letting each action be a (weak) fairness set. Which of the following properties are satisfied by the program? Motivate your answers:

a) $\Box\ (x \ge y)$

b) $x = 0\ \ leads - to\ \ x = 10$

c) $x > y\, \mathcal{U}\ x < y$

d) $x \ge y\, \mathcal{U}\ x < y$

e) $x \ge y\ \mathcal{W}\ x < y$

f) $(x > y + 1)\ \mathcal{W}\ (y = x + 20)$

Exercise 2.5 (Exam 2002) Consider the following action system.

```
**Action System** *Program 1*
  **variables**
      $x, y$ : integer
  **initially**
      $x = y = 0$
  **actions**
      $x := x + 1$
    $[\!]$
      $x > y \longrightarrow x := x - 1$
    $[\!]$
      $y < x \longrightarrow y := y + 1$
**end**
```

Assume that there are three weak fairness constraints: one for each of the three actions. Which of the following properties hold for the program:

a) $\Diamond\ (y \ge 4)$

b) $\Box\ (x \geq y)$

c) $\Box\ ((y = 7 \implies \Diamond\ (x = 7 \lor y = 8))$

d) $\Box \Diamond\ (x = y)$

e) $\Box \Diamond\ (x > y)$

Give *short* motivations for your answers.

Exercise 2.6 (Exam 2002) Mr. Toggle has constructed a Promela model with two boolean variables, p and q. The purpose of the program is that

> when p is true then q should change its value in the next transition;
> when p if false, then q should remain unchanged;
> furthermore, q should change its value infinitely many times.

Your problems are

a) State the above correctness property in temporal logic.

b) Construct a never-claim which is added to the Promela model, so that SPIN can check that the program satisfies this correctness property.

Exercise 2.7 (Reexam 2004) Imagine that someone has constructed a Promela model with an integer variable `x`, and wants to check the following property:

> Whenever `x` assumes the value `7` after (but not necessarily immediately after) some point in time when `x` assumed the value `5`, then there must be some future point in time (i.e., following the point in time when `x` is `7`) where `x` assumes the value `9`.

a) Express the property in temporal logic.

b) Write a never claim which checks that the Promela model satisfies the property.

Exercise 2.8 (Exam 2007) Assume that Promela would have a datatype `bignum` which contains all nonnegative integers (i.e., no upper limit on the value of a variable of type `bignum`). Consider the following (extended) Promela model.

```
bignum  x = 2;
bit     b = 0;

active proctype A() {
        do
        :: !b -> x++
        od
}
```

```
active proctype B() {
        do
        :: (b && x > 0)  -> x--
        od
}

active proctype C() {
        b = 1
}
```

Assume that we specify process fairness for this Promela model. Which of the following properties hold for the model:

a) $\Diamond$ `(x == 0)`

b) $\Diamond$ `(x == 10)`

c) $\Box\,[$`(x == 10)` $\Longrightarrow$ `(x == 10)` $\mathcal{W}$ `(x > 10)`$]$

d) $\Box\,[$`(x == 10)` $\Longrightarrow$ $\Diamond$ `(x == 6)`

Exercise 2.9 (Exam 2007) Two processes, `A` and `B`, access a shared resource. A part of a Promela model is given below.

```
/* declarations (not shown here) */

active proctype A() {
again:  /* some statements */

        ...

crit:   /* here is the access to the shared resource */

        /* some statements */

        goto again
}

active proctype B() {
again:  /* some statements */

        ...

crit:   /* here is the access to the shared resource */

        /* some statements */
        goto again
}
```

Formulate temporal logic formulas, which express the following properties:

a) The process `A` never accesses the file.

b) The process `B` does not access the file before process `A` has accessed the file.

You should **not** add extra variables to the Promela model, just write temporal logic formulas.

c) Write a never-claim, which checks that

> between any two accesses of the process `A`, the process `B` accesses the file an odd number of times.

Do not worry about whether or not the properties are satisfied by the model. Just express them.

3 Manipulation of Logic Properties

Exercise 3.1 Which of the following temporal logic properties are valid? Here p and q are arbitrary state formulas.

a) $\Diamond p \wedge \Box q \quad \Leftrightarrow \quad \Diamond (p \wedge \Box q)$

b) $(\Box p \vee \Diamond q) \quad \Leftrightarrow \quad p \mathcal{W} (\Diamond q)$

c) $\Diamond \Box (p \implies \Box q) \quad \Leftrightarrow \quad (\Diamond \Box q \vee \Diamond \Box (\neg p))$

d) $\Diamond \Box p \wedge \Diamond \Box q \quad \Leftrightarrow \quad \Diamond (\Box p \wedge \Box q)$

e) $\Diamond p \wedge \Box q \quad \Leftrightarrow \quad \Box (\Diamond p \wedge q)$

f) $(p \mathcal{U} q) \mathcal{U} q \quad \Leftrightarrow \quad p \mathcal{U} q$

g) $p \mathcal{U} q \quad \Leftrightarrow \quad [((\neg p) \mathcal{U} q) \implies (p \mathcal{U} q)]$

h) $(p \mathcal{U} q) \wedge (q \mathcal{U} r) \quad \Leftrightarrow \quad p \mathcal{U} r$

i) $(p \mathcal{U} (\Box q)) \quad \Leftrightarrow \quad (p \mathcal{U} q \wedge \Box (q \implies \Box q)$

j) $(\Box p \vee \Diamond q) \quad \Leftrightarrow \quad p \mathcal{W} (\Diamond q)$

k) $\Diamond \Box (p \implies \Box q) \quad \Leftrightarrow \quad (\Diamond \Box q \vee \Diamond \Box (\neg p))$

l) $\Diamond p \wedge \Box q \quad \Leftrightarrow \quad \Box (\Diamond p \wedge q)$

m) $\Box (p \wedge q) \implies (\Box p \wedge \Box q)$

n) $\Box (p \implies q) \implies (\Box p \implies \Box q)$

o) $(\Box p \implies \Box q) \implies \Box (p \implies q)$

Exercise 3.2 (New derived temporal operators). We might introduce new temporal operators so that certain properties can be expressed in a more concise form than the standard temporal operators. Here is an example.

- Define the binary operator *precedes* (written $\mathcal{P}$) by

  $$p \, \mathcal{P} \, q \triangleq (\neg q) \, \mathcal{W} \, (p \wedge \neg q)$$

  Try to understand what this operators means intuitively. As a test of your understanding, solve the following problems.

  1. Give a semantic definition for when $(\beta, i) \models p \, \mathcal{P} \, q$ in terms of $(\beta, j) \models p$ and $(\beta, j) \models q$, as was done e.g., for $\mathcal{U}$.
  2. Show how to express $\mathcal{U}$ in terms of $\mathcal{P}$ and the boolean operators.

Exercise 3.3 Let p, q, and r be flexible boolean variables. Consider the two temporal formulas:

a) $(p \, \mathcal{U} \, q) \, \mathcal{U} \, r$

a) $p \, \mathcal{U} \, (q \, \mathcal{U} \, r)$

You have two problems:

1. Are these two formulas equivalent? Give a convincing argument for your answer
2. For each of the two formulas, give a Büchi automaton that accepts exactly the set of computations that satisfy the formula.

Exercise 3.4 Consider a transition system $\mathcal{T}$ Assume that $\mathcal{T}$ satisfies $\Box (p_1 \implies \Diamond q_1)$ and $\Box (p_2 \implies \Diamond q_2)$.

a) Does it follow that $\mathcal{T}$ satisfies $\Box (p_1 \wedge p_2 \implies \Diamond q_1 \wedge q_2)$?

b) Does it follow that $\mathcal{T}$ satisfies $\Box (p_1 \wedge p_2 \implies \Diamond q_1 \vee q_2)$?

c) Does it follow that $\mathcal{T}$ satisfies $\Box (p_1 \vee p_2 \implies \Diamond q_1 \vee q_2)$?

Motivate or give counterexamples.

Exercise 3.5 For each of the following temporal logic formulas, give an equivalent formula which uses as few occurrences of temporal operators as possible.

a) $\Box (\Box p \wedge \Diamond q)$

b) $\Box \Diamond \Box p$

c) $\Diamond \Box \Diamond p$

Exercise 3.6 (Exam 2002) (4 p) Suppose that p_1, p_2, q_1 and q_2 are state formulas (or boolean variables). Assume that a system satisfies: $p_1 \mathcal{W} q_1$ and $p_2 \mathcal{W} p_2$. Which of the following can be inferred from this information?

a) the system satisfies $(p_1 \wedge p_2) \mathcal{W} (q_1 \wedge q_2)$

b) the system satisfies $(p_1 \vee p_2) \mathcal{W} (q_1 \vee q_2)$

Exercise 3.7 (Exam 1999) For each of the following temporal logic formulas, give an equivalent formula which uses as few occurrences of temporal operators as possible.

a) $\Box (\Box p \wedge \Diamond q)$

b) $\Box \Diamond \Box p$

c) $\Diamond \Box \Diamond p$

Exercise 3.8 (Exam 2003) Each of the following properties in temporal logic can be expressed by a **simpler** formula, which expresses exactly the same property. For each of the properties, find such a formula, which is as simple as possible. (here "simple" means "small" or "having few symbols")

a) $\Box \Diamond (p \, \mathcal{U} \, q)$

b) $\Box (p \, \mathcal{U} \, (\Diamond q))$

c) $\Box ((p \, \mathcal{U} \, q) \, \mathcal{U} \, p)$

Exercise 3.9 (Exam 2004) consider the 3 following new binary temporal operators, $@X$ ("at next"), Wh ("while"), and $\mathcal{B}$ ("before"). Their meaning is as follows:

a) $\phi \; @X \; \psi$ means: at the next state where ψ holds, also ϕ holds (here, the "next state" can also be the current one).

b) $\phi \, Wh \psi$ means: ϕ holds at least as long as ψ does.

c) $\phi \; \mathcal{B} \; \psi$ means: If ψ holds sometimes, ϕ does so before (here, "sometimes" includes the current state and "before" means "strictly before").

A more precise definition is as follows. If β is a behavior $\beta(0) \;\; \beta(1) \;\; \beta(2) \;\; \cdots$, we use $(\beta, i) \models \phi$ to denote that the formula ϕ is true in the i^{th} state of β. The meaning of the new operators is then explained as follows.

a) $(\beta, i) \models \phi \; @X \; \psi$ iff
$\forall j \geq i \; : \; [(\beta, j) \models \psi \wedge (\forall k : i \leq k < j \; : \; (\beta, k) \models \neg\psi)] \implies (\beta, j) \models \phi$

b) $(\beta, i) \models \phi \, Wh \psi$ iff
$\forall j \geq i \;:\; [(\beta, j) \models \psi \wedge (\forall k : i \leq k < j \;:\; (\beta, k) \models \psi)] \implies (\beta, j) \models \phi$

c) $(\beta, i) \models \phi \, \mathcal{B} \, \psi$ iff
$\forall j \geq i \;:\; [(\beta, j) \models \psi \implies \exists k : i \leq k < j \;:\; (\beta, k) \models \phi]$

Your problem is to express the formulas $p \;@X\; q$, $p \, Whq$, and $p \, \mathcal{B} \, q$ using standard operators (i.e., using only $\circ$, $\Box$, $\Diamond$, $\mathcal{U}$, and $\mathcal{W}$).

Exercise 3.10 (Exam 2004) Choose **two** of the following temporal logic properties

a) $p \;@X\; q$

b) $p \, Whq$

c) $p \, \mathcal{B} \, q$

and construct never-claims which check that a given Promela model satisfies the property in question.

Exercise 3.11 (Exam 2003) Each of the following properties in temporal logic can be expressed by a **simpler** formula, which expresses exactly the same property. For each of the properties, find such a formula, which is as simple as possible. (here “simple” means “small” or “having few symbols”)

a) $\Box \Diamond (p \, \mathcal{U} \, q)$

b) $\Box (p \, \mathcal{U} \, (\Diamond q))$

c) $\Box ((p \, \mathcal{U} \, q) \, \mathcal{U} \, p)$

4 Promela

Exercise 4.1 (exam 2006) Model the following classical measuring problem in Promela, in such a way that SPIN finds a solution as a violation of an assertion, temporal logic formula, or similar.

> Alice wants to give exactly 5 liters of water to Bob. She has a large supply of water and two buckets: one taking 10 liters, and one taking 7 liters. Her only means of measuring volumes of water is to pour water into, out of, or between buckets, knowing their volume (10 and 7 liters). How can she end up with exactly 5 liters in one of the buckets?

You do **not** have to find the actual solution to the problem (since you do not have SPIN available).

Exercise 4.2 (exam 2006) Show how to define a stack data structure in Promela (a stack is a container of objects, which is Last-in-First-out, i.e., the last inserted element is the first to be removed). You should provide a recepy for which variables to declare to represent the stack. You should define two operations

- `push(x)` pushes the value of the variable `x` onto the stack, in one atomic operation. If the stack is full, the operation should be unexecutable.
- `pop(x)` pops the top value from the stack and assigns its value to `x`. If the stack is empty, the operation should be unexecutable.

You can, e.g., use `inline` definitions for these operations. You can make your own decision about the types of elements in the stack, how to define the capacity of the stack, etc.

Exercise 4.3 (exam 2006) Below is a Promela model of a simple system with two concurrent processes that manipulate a shared variable `N`.

```
byte    N = 0 ;

active [2] proctype P() {
        byte temp , i ;
        i = 1 ;
        do
        :: i < 10  -> temp = N ; temp++ ; N = temp ; i++
        :: else -> break
        od
}
```

What is the *smallest* possible value that the variable `N` can assume when both processes have terminated?

Exercise 4.4 (exam 2005) Promela supports point-to-point channels with one sending and one receiving process. Describe how in Promela one can model protocols that employ broadcast communication. We can assume that there are N processes, numbered 0, 1, . . . , $N - 1$. A process communicates by sending a message to *all* other processes. A broadcast operation should be *atomic*, in the sense that either it is sent to all other processes, or (maybe due to some overflow situation) to no other process at all. Furthermore, all processes should receive the messages that have been sent so far in the same (global) order.

Show your solution for a specific configuration with, say, 3 processes and some channels that you define in an appropriate manner. The broadcast operation should be defined in the form of a Macro, which for your specific configuration defines the macro `broadcast(i,m)` in terms of standard Promela constructs, where `i` is the number of the sending process and `m` is the message to be transmitted. We can assume that a message `m` is a byte.

It should be possible to use your macro in most reasonable contexts. For instance, one should be able to write something like

```
do :: broadcast(i,4)
   :: chan?j,x   ->   /* process message x */ ; break
od
```

in the code for process `i`, where `chan` is the name of some input channel of process `i` (you may use different names for channels, of course), to model a situation where process `i` retransmits message `4` until it receives an appropriate input message.

Exercise 4.5 In Promela, the `atomic` statement is useful for decreasing the number of transitions and the number of global states that must be analyzed. Often, a promela model can be made easier to analyze by putting some sequences of statements inside and `atomic` statement. However, one must be careful when inserting it, since the analysis may become incorrect.

Consider a Promela model that has the following structure.

```
int x = 0 ;
int y = 0 ;

proctype A()
{        ...
}

proctype B()
{        ...
}

init
{        run A(); run B()
}
```

The process `B` is defined in Promela. We do now know its code, only that it may contain one or several `assert` statements that check invariants containing `x` and `y`. We want to make analysis easier by inserting `atomic` statements into process `A`. Process `A` has local variables `i` and sometimes also `j`. We show two examples of process `A` before and after the change. The problem is to say which of these insertions are correct, regardless of how process `B` is defined. For those that are correct, give a short informal explanation why. For those that are incorrect, give a process `B` in which some `assert`ed invariant can be false before the change but not after inserting the `atomic` statement.

a) Changing

```
proctype A()
{       int i = 0,j = 0;
        i++ ;
        x = i ;
        j++
}
```

into

```
proctype A()
{       int i = 0,j = 0;
        atomic{ i++ ;
                x = i ;
                 j++
        }
}
```

b) Changing

```
proctype A()
{       int i = 0;
        x = i ;
        y = i ;
        i++
}
```

into

```
proctype A()
{       int i = 0;
        atomic{ x = i ;
                y = i ;
                i++
        }
}
```

Exercise 4.6 (exam 2004) Consider a process in a system which operates on a global datastructure, named `table`, which is an array of 100 integers. The process repeatedly performs an operation, in which it arbitrarily selects one integer in `table` which is even, and transmits this integer over a channel `ch`, provided that `ch` is not full. In an action-system like syntax, we can model this process as follows.

```
**Action System** *Process*
  **variables**
      table : $array[0..99]\,of\ integer$
      ch : $queue[0..4]\,of\ integer$
  **initially**
      **actions**
      $\langle [\!]\, i\ :\ 0 \le i \le 99\ ::\ even(\mathtt{table}[i]) \wedge \neg full(\mathtt{ch})\ \longrightarrow\ \mathtt{ch}\ :=\ \mathtt{ch} \bullet \mathtt{table}[i] \rangle$
**end**
```

Here, $\mathtt{ch} := \mathtt{ch} \bullet \mathtt{table}[i]$ means that $\mathtt{table}[i]$ is sent on channel `ch` ($\bullet$ denotes concatenation).

Your task is to model this process as a process in Promela. The process is assumed to be part of a bigger Promela model, where the relevant information is that in the bigger model `table` and `ch` are declared globally, e.g as:

```
int  table[100];
chan ch = [3] of { int };
```

and that `table` is initialized with integers somehow in system initialization, and does not change thereafter. Important constraints are that you should define the process so that when SPIN analyses the entire system, your process does not increase the number of possible global states of the system. (Of course, the global variables `table` and `ch` can assume many different possible values, but they do not belong to "your" process.) This can be achieved if you make the transmission of an integer an atomic operation, and make sure that between atomic operations, the process is in a unique local state. However, you must not restrict the possible behaviors of the process: it should be able to select *any* even integer for transmission at *any* point in time.

Exercise 4.7 (reexam 2004) consider the following simple Promela model of a (stupid) mutual exclusion algorithm.

```
int x,y;

active proctype userx
{
        do
        :: x < 9 -> x++
        :: (x ==  9 && y =< 9) -> x++
        :: x == 10 -> x = 0
        od
}

active proctype usery
{
        do
        :: y < 9 -> y++
        :: (y ==  9 && x =< 9) -> y++
        :: y == 10 -> y = 0
        od
}
```

Each process has a variable, which it increments until it gets the value 10. The value 10 represents that the process is in the critical section; i.e., process `userx` is in the critical section when `x` is `10`, and process `usery` is in the critical section when `y` is `10`.

When verifying that the system satisfies mutual exclusion (i.e., that the expression `x == 10 && y == 10` can never become true), SPIN will need to check many states (in this case roughly 100 states, but if we change 10 to 1.000 the number will increase). There are several ways to optimize the model and make the state-space smaller. In this case, your task is to utilize the *symmetry* of the problem, i.e., that the system and the correctness property are symmetric wrp. to `x` and `y`: for each state of the system there is another "symmetric" state where the roles of `x` and `y` are swapped. Explain how we can reduce the number of states that need to be considered in verification by exploiting the symmetry of the problem (e.g., we may not have to analyze a state whose corresponding symmetric state has been analyzed). Show how to modify the model to have less states, but without running into the danger that SPIN will miss potential violations of the mutual exclusion property (this model is simple, but your technique should also be applicable to models which are not so easy to understand). State roughly how much reduction of the number of reachable states you have achieved.

5 Büchi Automata and Never Claims

Exercise 5.1 Consider the formula $p\ \mathcal{U}\ (q\ \mathcal{U}\ r)$, where p and q are flexible boolean variables

a) Construct a Büchi automaton that accepts the set of infinite sequences of states that satisfy $p\ \mathcal{U}\ (q\ \mathcal{U}\ r)$.

b) Construct a Büchi automaton that accepts the set of infinite sequences of states that satisfy $\neg(p\ \mathcal{U}\ (q\ \mathcal{U}\ r))$.

c) Assume that we have a PROMELA model of a system containing the boolean variables p, q, and r. What additional code do you add in order to use SPIN to validate that the property $p\ \mathcal{U}\ (q\ \mathcal{U}\ r)$ holds for all computations of the model.

Exercise 5.2 Consider the following PROMELA model:

```
#define true    1
#define false   0
#define turn1   false
#define turn2   true

bool   y1, y2, t;

proctype P1()
{
l1:     y1 = true;
l2:     t = turn2;
l3:     (y2 == false ||  t == turn1) ;
l4:      /* critical section */
        atomic { y1 = false ; goto l1 }
}

proctype P2()
{
m1:     y2 = true;
m2:     t = turn1;
m3:     (y1 == false ||  t == turn2) ;
m4:      /* critical section */
        atomic { y2 = false ; goto m1 }
}

init
{       atomic  {  run P1() ; run P2() }
}
```

Show how to add never claims that check the following properties:

a) *at* (`l2`) leads to *at* (`l4`)

b) Bounded overtaking: Whenever *at* (`l2`) holds, then eventually *at* (`l4`) will hold, and until then process `P2` will enter to and exit from `m4` at most once.

You could of course run and check the results in SPIN.

Exercise 5.3 Since. Consider the new binary temporal operator S , pronounced "since". Intuitively, since is the "backwards" analogue of (strong) until. That is, $\phi_1 \mathcal{S} \phi_2$ means that the last occurrence of ϕ_2 was followed by a period of ϕ_1 up to the present from the state after that where ϕ_2 held. The formal semantics can be described as

- $(\sigma, i) \models \phi_1 \mathcal{S} \phi_2$ iff $\exists j \leq i \ : \ (\sigma, j) \models \phi_2$ and $\forall k : j < k \leq i \ (\sigma, k) \models \phi_1$

Your problem is the following:

a) Express $\Box\, (p \implies p\, \mathcal{S}\, q)$ as a formula containing p, q, and the other temporal operators that we have used ($\circ$, $\Box$, $\Diamond$, $\mathcal{U}$, $\mathcal{W}$).

b) Draw a Büchi automaton that accepts the language that satisfies $\Box\, (p \implies p\, \mathcal{S}\, q)$.

c) Make a `never`-claim in PROMELA that will check whether a program satisfies $\Box\, (p \implies p\, \mathcal{S}\, q)$.

Exercise 5.4 Assume that we model a system with the boolean variables p and q in PROMELA. We intend to verify that the system satisfies the temporal logic property

$$\Box\, (p \implies (p\, \mathcal{U}\, q))$$

Describe how to produce a never-claim in PROMELA, which checks whether each computation of the system satisfies $\Box\, (p \implies (p\, \mathcal{U}\, q))$

Exercise 5.5 Let ϕ be an arbitrary formula in temporal logic. We assume that all variables in ϕ are boolean. Let $\langle \Sigma, S, S^0, \rightarrow, F \rangle$ be a Büchi automaton which accepts exactly the infinite computations that satisfy the temporal logic formula ϕ. Describe how you can systematically construct a Büchi automaton which accepts exactly the temporal logic formula $\Diamond\, \phi$. Show how your construction works on a Büchi automaton that accepts $\phi \equiv p\, \mathcal{U}\, q$, where p and q are boolean (flexible) variables.

Exercise 5.6 This problem has two different parts:

a) Let p and q be boolean variables whose values may vary over time. Construct a Büchi Automaton which accepts exactly those infinite sequences of values of p and q which satisfy:

$$\Box \Diamond\, (p \wedge \circ\, q)$$

i.e., there should be infinitely many instants where p is true, such that q is true in the next instant.

b) Suppose you want to check whether a Promela program satisfies the above property. How can you add a never claim to check that?

Exercise 5.7 (Exam 2005) Consider the binary temporal operator $\mathcal{B}$, pronounced "before". That is, $\phi_1\, \mathcal{B}\, \phi_2$ means that any occurrence of a state that satisfies ϕ_2 must be (strictly) preceded by a state that satisfies ϕ_1. The formal semantics can be described as

$$(\sigma, i) \models \phi_1\, \mathcal{B}\, \phi_2 \quad \text{iff} \quad [\forall j \geq i : (\sigma, j) \models \phi_2 \text{ implies } \exists k : i \leq k < j : (\sigma, k) \models \phi_1]$$

Your problem is the following:

a) Construct a Büchi automaton which accepts the set of behaviors that satisfy $p\, \mathcal{B}\, (\Box\, q)$, where p and q are arbitrary boolean expressions.

b) Make a `never`-claim in Promela that will check whether a Promela model satisfies $p\, \mathcal{B}\, (\Box\, q)$

Exercise 5.8 (Reexam 2005) This problem has two different parts:

a) Let p and q be boolean variables whose values may vary over time. Construct a Büchi Automaton which accepts exactly those infinite sequences of values of p and q which satisfy:

$$\Box \Diamond\, (p \wedge \circ\, q)$$

i.e., there should be infinitely many instants where p is true, such that q is true in the next instant.

b) Suppose you want to check whether a Promela program satisfies the above property. How can you add a never claim to check that?

Exercise 5.9 (Reexam 2004) Imagine that someone has constructed a Promela model with an integer variable `x`, and wants to check the following property:

> Whenever `x` assumes the value `7` after (but not necessarily immediately after) some point in time when `x` assumed the value `5`, then there must be some future point in time (i.e., following the point in time when `x` is `7`) where `x` assumes the value `9`.

a) Express the property in temporal logic.

b) Write a never claim which checks that the Promela model satisfies the property.

Exercise 5.10 (Exam 2003) Let p be a boolean variable (of some transition system). The temporal property

> p is true at all even-numbered states and can have any value in odd-numbered states, i.e., in a sequence $s_0\ s_1\ s_2\ s_3\ s_4\ \ldots$ of states, p is true in states $s_0\ s_2\ s_4\ \ldots$ and has arbitrary values in the other states.

cannot be expressed in temporal logic. The problem is the following

a) Construct a Büchi automaton which accepts exactly all behaviors that satisfy the above property.

b) Construct a Büchi automaton which accepts exactly all behaviors that do **not** satisfy the above property.

c) Construct a Büchi automaton which accepts exactly all behaviors that have no suffix which satisfies the property. In other words, for any suffix $s_n\ s_{n+1}\ s_{n+2}\ s_{n+3}\ s_{n+4}\ \ldots$ it should **not** be the case that p is true in all of the states $s_n\ s_{n+2}\ s_{n+4}\ \ldots$.

Exercise 5.11 (Exam 2000) Prove that there is no *deterministic* Büchi Automaton, which recognizes exactly the set of infinite sequences that satisfy $\Diamond\ \Box\ p$, where p is a state formula.

Hint: Assume that such a deterministic automaton exists. Consider how it behaves when it sees a prefix of some word that satisfies $\Diamond\ \Box\ p$: eventually, e.g., after seeing a long sequence of p-states, it must get to an accepting state. After the automaton gets to an accepting state, assume that it sees a sequence of states satisfying $\neg p$: eventually it must get to a non-accepting state. Then we feed it states satisfying p, etc. Thus, the automaton can be made to oscillate between accepting and non-accepting states. Based on observations like this, you should be able to prove that a deterministic Büchi automaton can not accept exactly the sequences in $\Diamond\ \ \Box\ \ p$.

Exercise 5.12 (Exam 2004) construct two finite automata (with states, initial states, transitions, and accepting states), which are not equivalent when viewed as classical finite automata that accept finite strings, but are equivalent when viewed as Büchi automata that accept infinite strings.

Exercise 5.13 (Reexam 2004.) Imagine that we have constructed a Büchi automaton $\mathcal{A}$ which checks a particular temporal property ϕ (i.e., $\mathcal{A}$ accepts exactly the behaviors that satisfy $\neg\phi$). Show how to easily transform $\mathcal{A}$ into an automaton which checks $\Box\ \phi$. Apply your construction to the automaton which checks $\phi \equiv p \implies \Diamond\ q$ so that it checks $\Box\ (p \implies \Diamond\ q)$.

Exercise 5.14 Prove that any ω-regular safety property can be accepted by a Büchi automaton where all states are accepting. (This is an alternative definition of safety property. We know that the complement of a safety property is accepted by a Büchi automaton which cannot move from an accepting to a non-accepting state.)

Exercise 5.15 Prove that if all states of a Büchi automaton are accepting, then it accepts a safety property.

Exercise 5.16 (exam 2005) Construct two finite automata (with states, initial states, transitions, and accepting states), which are equivalent (i.e., accept the same regular sets) when viewed as classical finite automata that accept finite strings, but are not equivalent (i.e., accept different ω-regular sets) when viewed as Büchi automata that accept infinite strings.

Exercise 5.17 (exam 2007) Assume you have constructed a Promela model with boolean variables p and q. Construct either a Büchi automaton or a never claim, which accepts exactly the behaviors that violate the temporal logic formula

$$(\Box\ p)\ \mathcal{U}\ (\Box\ q)$$

6 Verification Algorithms

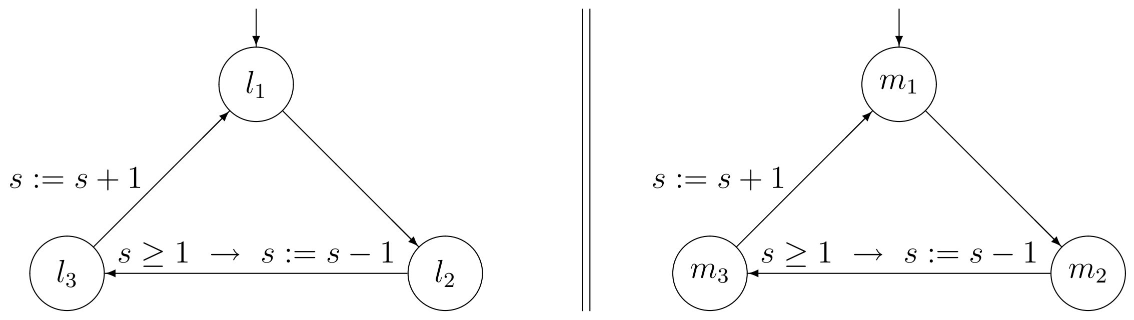
Exercise 6.1 (Exam 2007) Consider the following simple Promela model of a (stupid) mutual exclusion algorithm.

```
byte x = 0;
byte y = 0;

active proctype A() {
        do
        :: x <= 10 -> x++
        od
}

active proctype B() {
        do
        :: y < x  -> y++
        od
}
```

How many reachable states does this Promela model have?

Exercise 6.2 Consider the following attempt at solving the mutual exclusion problem

```
Transition System mutex1
   declare $s$ : integer
   initially $s = 1$
```



l_1
$s := s + 1$
l_3
$s \geq 1 \; \rightarrow \; s := s - 1$
l_2
m_1
$s := s + 1$
m_3
$s \geq 1 \; \rightarrow \; s := s - 1$
m_2

end

There are two weak fairness sets. The fairness set of the left process contains the actions that leave l_2 and l_3. The fairness set of the right process contains the actions that leave m_2 and m_3. The problem is to show how a model-checker, similar to SPIN, would analyze whether the system satisfies the following two properties

a) $\Box\ (\neg(at\ l_3 \wedge at\ m_3))$

b) $\Box\ (at\ l_2 \implies \Diamond\ at\ l_3)$

You can show how the model-checker would explore the state-space of the action system, and point out how it decides whether the properties are satisfied or not.

Exercise 6.3 Consider the following simple action system, which manipulates only one integer.

```
Action System  Simple
  variables
      $x$ : integer
  initially
      $x = 0$
  actions
      $x \leq 3 \longrightarrow x := x + 2$
   []
      $x \geq 2 \wedge even(x) \longrightarrow x := x - 2$
   []
      $even(x) \wedge x \leq 4 \longrightarrow x := x + 1$
end
```

There are three weak fairness sets, one for each action. Do the following

a) Draw the reachability graph for this action system. Label each state by the value of x.

b) Using the procedure for checking leadsto properties on reachability graphs (which is based on finding strongly connected components), analyze whether or not the action system satisfies the property

$$\Box\,(x = 0 \implies \Diamond\, x = 5)$$

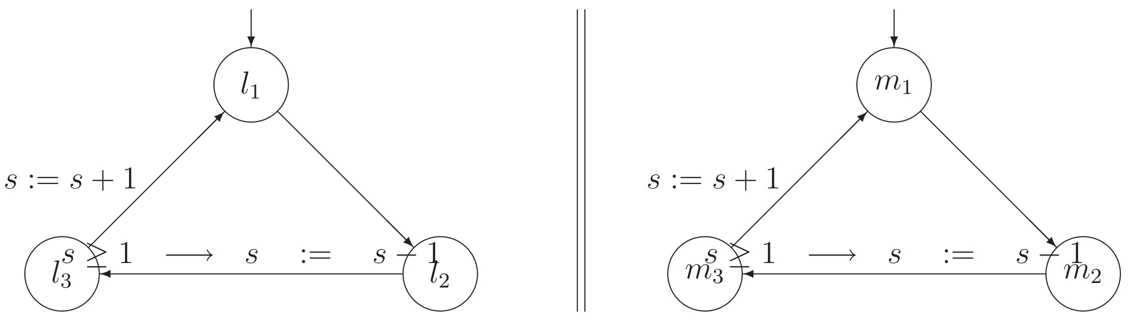
You should of course use the information about weak fairness sets given above.

Exercise 6.4 In the figure below is again a suggested program for acheiving mutual exclusion between two processes. In all action systems, locations l_1 and m_1 are intended to represent the section of a process which is not interested in the critical section, locations l_2 and m_2 are intended to represent the section where the process is interested in entering the critical section, and locations l_3 and m_3 represent the critical sections themselves. For this system, we assume that there is only one fairness constraint, represented by the set of non-idling transitions. This means only that a computation does not stop performing non-idling steps if some action in the figure below is enabled. This fairness constraint is e.g. the one assumed for PROMELA programs. There is no other fairness, e.g. that requires fairness for each process individually.

Action System *mutex1*

declare s : integer

initially $s = 1$


l_1
$s := s + 1$
$s \geq 1 \longrightarrow s := s - 1$
l_3
l_2
m_1
$s := s + 1$
$s \geq 1 \longrightarrow s := s - 1$
m_3
m_2

end

Figure 1: Mutex1

Assume that we want to check whether MUTEX1 satisfies the temporal logic formula *at* l_2 *leads* − *to at* l_3. We can then use the algorithm presented in the paper by Courcoubetis et al (*Memory Efficient Algorithms for the Verification of Temporal Properties*) where the temporal property is negated, then transformed into a Buchi automaton, which is composed with the mutual exclusion algorithm. The composition is searched depth-first, the accepting states are listed in postorder, and a second depth-first search is started from the accepting states.

Your task is to describe how the search tree in this algorithm looks by: drawing the search tree (of the composition of algorithm and Buchi automaton), showing the accepting states from the first pass in post-order, and adding on the continued second search. Describe the order in which nodes in the search tree are visited, and describe at which point the algorithm determines whether the formula holds or not.