ETH zürich

P. Müller and D. Basin

Formal Methods and Functional Programming

Session Sheet 14: Modeling and LTL

Installing and Running Spin

To run the Promela models, you will need to install the Spin model-checker as well as a C compiler. There are multiple ways to install Spin on your machine:

- **Windows:** You can download the archive from the following link and follow the readme: `https://polybox.ethz.ch/index.php/s/cQifMKXUW3G2iAI`
- **Ubuntu:** Run `sudo apt-get install spin` in a terminal to install the `spin` package.
- **Mac:** Use Homebrew (`https://brew.sh`) to install Spin by running `brew install spin`.
- **Executables:** Download pre-compiled executables from Spin's GitHub page: `https://github.com/nimble-code/Spin/tree/master/Bin`.
- **Compiling**: You can compile Spin from source from: `https://github.com/nimble-code/Spin`.

Short re-cap for running Spin:

- `spin filename.pml` will carry out a simulation of the model, yielding one random trace. This does *not* perform an exhaustive check the model.
- `spin -a filename.pml` will create a file `pan.c`, that must be compiled and run to exhaustively check a model. In case of failure, a corresponding trail file (`filename.pml.trail`) is typically generated, containing the information about the failing trace.
- `spin -t filename.pml` will replay the trace from the corresponding trail file.

Assignment 1 (Modeling in Promela)

Task 1.1. Consider the statement

```
y := 0;
while x > 0 do
    y := y + x;
    x := x - 2
end
```

and write a model in Promela to check if the statement, starting in a state σ with $\sigma(\mathtt{x}) = 3$ will reach a state σ' with $\sigma'(\mathtt{y}) = 4$.

Solution. We use the following Promela model:

```
#define x0 3

int x = x0, y

inline s() {
    y = 0
    do
    :: x > 0 -> y = y + x; x = x - 2
    :: else  -> break;
    od
}

init {
    printf("starting with x = %d and y = %d\n", x, y)
    s()
assert(y == 4)
    printf("finishing with x = %d and y = %d\n", x, y)
}
```

What changes do we need to make to the model if we want to use `proctype s()` instead of `inline s()`?

Task 1.2. Write a model in Promela to verify that executing the statement

$$\mathtt{x := 1} \ [] \ \mathtt{x := 2; \ x := x + 2}$$

will result in a state σ where either $\sigma(\mathtt{x}) = 1$ or $\sigma(\mathtt{x}) = 4$.

Solution. We use Promela's conditionals without conditions to model the non-determinism):

```
int x

init {
    if
    :: x = 1
    :: x = 2; x = x + 2
    fi

    assert(x == 1 || x == 4)
    printf("finishing with x = %d\n", x)
}
```

Task 1.3. Now, consider the statement

```
x := 1 par (x := 2; x := x + 2)
```

and write a model verifying that its execution results in a state σ with $\sigma(\mathrm{x}) \in \{1, 3, 4\}$.

Solution. A possible model is as follows:

```
int x

proctype left() {
x = 1
}

proctype right() {
x = 2
x = x + 2
}

init {
    run left()
    run right()

    // wait for processes to terminate
    _nr_pr == 1

    assert(x == 1 || x == 3 || x == 4)
    printf("finishing with x = %d\n", x)
}
```

Task 1.4. Consider the following program:

```
x := 5;
y := 1;
(while x > 1 and y < 5 do
   (x := x - y [] y := y + 1)
end
par
while x > 0 do
    y = y + 1;
    x = x - 1
end)
```

Assume that we start the program in some state. Can we reach a final state σ with $\sigma(\mathtt{x}) = -7$? What is the minimal value of the variable x after executing the program?

Solution. In order to check whether x can be -7, we use an assertion that states that this is not the case. As the model-checker detects a violation, we know that it can be reached. We use the second assertion to determine the least value of the variable x, using dichotomy to find the largest lower bound that is accepted. The least value possible for x turns out to be -9.

```
int x, y

proctype left() {
    do
    :: x > 1 && y < 5 -> x = x - y
    :: x > 1 && y < 5 -> y = y + 1
    :: else           -> break
    od
}

proctype right() {
    do
    :: x > 0 -> y = y + 1; x = x - 1
    :: else  -> break
    od
}

init {
    x = 5
    y = 1

    atomic {
        run left()
        run right()
    }
```

```
    // wait for processes to terminate
    _nr_pr == 1

    assert(x != -7)
    // assert(x >= -9)
    printf("finishing with x = %d and y = %d\n", x, y)
}
```

Note: The atomic block ensures that both threads, `left` and `right` are forked at the same time. Removing the atomic block allows additional traces where some steps of `left` are executed before `right` is forked; which does not affect the outcome of the execution, however.

Task 1.5. Consider the Promela model below and use spin to identify a deadlock.

```
int x

proctype left() {
    do
    :: x > 0 -> x = x - 1
    od
}

proctype right() {
    do
    :: x < 0 -> x = x + 1
    od
}

init {
x = 2
    run left()
    run right()
}
```

Solution. We use spin to check for invalid endstates.

Assignment 2 (Modeling Traffic Lights)

Consider a traffic light with a green, a yellow and a red light. We wish to check the safety property "red is always preceded by yellow". Which atomic propositions do you need? State the LTL property.

Solution. We use the atomic propositions y and r, which indicate whether the yellow light and the red light, respectively, are on. Since exactly one light is on at a time, this lets us derive the value of the green light as well (we choose not to check this property in the resulting model, which makes it, and the discussion below, simpler). So: $AP = \{y, r\}$, and hence, $\mathcal{P}(AP) = \{\emptyset, \{y\}, \{r\}, \{r, y\}\}$.

We will check the following LTL property: $\Box(\bigcirc r \Rightarrow y)$.

Note that this is not clearly an *exact* translation of the (informal) English description. For one thing, a trace in which the very first state has the red light on would seem to violate the informal description (in a way which likely doesn't matter in reality) but not the LTL formula. Note also that in some sense the LTL property (and, arguably, the English description) is stricter than the intuitive property one would like to check for a real traffic light, since it does not allow the traffic light to remain red for two successive states (e.g. a sequence of abstract states containing $\ldots, \{y\}, \{r\}, \{r\}, \ldots$ would violate the LTL property). A LTL formula addressing all these concerns would be $\neg r \wedge \Box(\bigcirc r \Rightarrow y \vee r)$, but it is not necessarily immediate to see that it ensures that any contiguous sequence of red lights is indeed preceded by a yellow light.

Assignment 3 (Linear Temporal Logic)

Task 3.1. Consider a transition system with two states s_1, s_2, where s_1 is the initial state, transitions back and forth from s_1 to s_2 and a loop from s_2 to itself. Let p be true in and only in state s_2. Discuss the difference between $\Box\Diamond p$ (holds) and $\Diamond\Box p$ (does not hold – counter example $s_1 s_2 s_1 s_2 s_1 s_2 \ldots$).

Solution. $\Box\Diamond p$ is a fairness property, that means that p holds infinitely often. In particular, this can hold even if the element of the trace for which p hold are not contiguous, or even separated by a large time span. This is why it holds for this system: when s_2 is left, the system must come back to s_2 eventually.

In contrast, $\Diamond\Box p$ also means that p holds infinitely often, but also that starting from some element of the trace, it always holds, so the elements for which p holds must all be contiguous from that point on. This does not hold for this system, because the system may always leave s_2.

In general, if $\Diamond\Box p$ holds then $\Box\Diamond p$ also holds, but as the example shows, the converse does not work.

Task 3.2. Now consider a transition system with three states s_1, s_2, s_3, where s_1 is the initial state. There are the following transitions: $s_1 \to s_2$, $s_1 \to s_3$, $s_2 \to s_3$, $s_2 \to s_2$ and $s_3 \to s_3$. Let p be true in and only in s_2. Discuss the LTL formula $\bigcirc p \Rightarrow \Box p$. The formula is false in the model. However, under the wrong interpretation of $\models$ one might think it is true ($\bigcirc p$ is false in the model; therefore $\bigcirc p \Rightarrow \Box p$ appears to be vacuously true). Notice that $\bigcirc \neg p \Rightarrow \Box \neg p$ happens to be true.

Solution. The formula $\bigcirc p \Rightarrow \Box p$ means that for every trace, if $\bigcirc p$ holds, then so does $\Box p$. But for the trace $s_1 s_2 s_3 s_3 s_3 \ldots s_3 \ldots$, $\bigcirc p$ holds and not $\Box p$. Confusion may arise because $\bigcirc p \Rightarrow \Box p$ can be intuitively taken to means that if $\bigcirc p$ holds, $\Box p$ must hold as well. However, the two interpretations of the formula are not equivalent, which is made clear if one unroll the definitions. In the first case, the two formula share the trace, while in the second they do not.

Task 3.3. In the same transition system, discuss the formula $\Diamond p \ \vee \ \Box \neg p$, which is true, but may be considered false (neither of the disjuncts is true).

Solution. The problem is similar to the one for Task 3.2, but in reverse. While none of the formula holds for every trace – respective counter-examples $s_1 s_3 s_3 s_3 \ldots s_3 \ldots$ and $s_1 s_2 s_3 s_3 s_3 \ldots s_3 \ldots$ – the disjunction of the two formulas does in fact hold because the two formulas do not *share* a single counter-example. $\Diamond p \vee \Box \neg p$ is actually true regardless of the transition system and of p: for a given trace, either the trace contains no element satisfying p, hence $\Box \neg p$ hold for that particular trace, or it does contain one, hence $\Diamond p$ hold for that particular trace.

The discrepancy between the correct interpretation of LTL formulas and the wrong intuition comes from the fact that in LTL only one trace is considered at a time. The operators $\vee, \Rightarrow$ etc. apply to the trace and not the model (i.e. a set of all traces). So it is possible, e.g., for a formula $\phi \vee \psi$ to be true in the model when ϕ and ψ are not.

Assignment 4 (Liveness and Safety Properties)

Let p be an atomic proposition.

Task 4.1. Prove that both $\Box \Diamond p$ and $\Diamond \Box p$ express liveness properties.

Solution. Let t be any finite sequence. Let t_c be an infinite sequence, in which each state satisfies p. Then $t.t_c$ satisfies both properties. For $\Diamond \Box p$, this is immediate as we can use the part in t_c from which $\Box p$ holds. For $\Box \Diamond p$, note that every element in $t.t_c$ happens before an element in t_c, after which an element satisfying p must arise because t_c satisfies $\Box \Diamond p$.

Task 4.2. Prove that $\Box p$ expresses a safety property.

Solution. Let t_c be an infinite sequence that does not satisfy $\Box p$. Then, by the definition of $\Box$, there is a state σ within t_c that does not satisfy p. Let t'_c be the finite prefix of t_c up to and including σ. Let t be any infinite sequence. Then $t'_c t$ does not satisfy $\Box p$.