ENS DE LYON

TD de Sémantique et Vérification

I– Modelling Concurrent Systems

Henning Basold
henning.basold@ens-lyon.fr

In this first set of exercises, we will discuss the modelling of concurrent systems as circuits, transition systems and program graphs. In particular, we will discuss the interleaving of parallel systems.

Recommendation: The exercises are all purely pen and paper exercises. However, it is quite fun to implement notions from the course and the exercises. At the very end, you may obtain this way your very own model checker. This week, you may implement transition systems, their execution and the interleaving operators. Note that your implementation will not be evaluated as part of the course.

Preliminaries: Interleaving Operators

In this preliminary section, we provide the definitions of the different interleaving operators from the book.

Definition 1 (Interleaving of Transition Systems)**.** Let $TS_i = (S_i, \mathrm{Act}_i, \to_i, I_i, \mathrm{AP}_i, L_i)$ be transition systems for $i = 1, 2$. The interleaved transition system is given by

$$TS_1 \mid\mid\mid TS_2 = (S_1 \times S_2, \mathrm{Act}_1 \cup \mathrm{Act}_2, \to, I_1 \times I_2, \mathrm{AP}_1 \cup \mathrm{AP}_2, L),$$

where $L\langle s_1, s_2\rangle = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\to$ is defined by the following two rules.

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1', s_2\rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1, s_2'\rangle}$$

Definition 2 (Interleaving of Program Graphs)**.** Let $PG_i = (\mathrm{Loc}_i, \mathrm{Act}_i, \mathrm{Effect}_i, \hookrightarrow_i, \mathrm{Loc}_{0,i}, g_{0,i})$ be program graphs for $i = 1, 2$. The interleaved program graph is given by

$$PG_1 \mid\mid\mid PG_2 = (\mathrm{Loc}_1 \times \mathrm{Loc}_2, \mathrm{Act}_1 \uplus \mathrm{Act}_2, \hookrightarrow, \mathrm{Loc}_{0,1} \times \mathrm{Loc}_{0,2}, g_{0_1} \wedge g_{0,2}),$$

where $\mathrm{Effect}(\alpha, \eta) = \mathrm{Effect}_i(\alpha, \eta)$ for $\alpha \in \mathrm{Act}_i$ and the transition relation $\hookrightarrow$ is defined by the following two rules.

$$\frac{\ell_1 \xhookrightarrow{g:\alpha}_1 \ell_1'}{\langle \ell_1, \ell_2\rangle \xhookrightarrow{g:\alpha} \langle \ell_1', \ell_2\rangle} \qquad \frac{\ell_2 \xhookrightarrow{g:\alpha}_2 \ell_2'}{\langle \ell_1, \ell_2\rangle \xhookrightarrow{g:\alpha} \langle \ell_1, \ell_2'\rangle}$$

Definition 3 (Handshaking)**.** Let $TS_i = (S_i, \mathrm{Act}_i, \to_i, I_i, \mathrm{AP}_i, L_i)$ be transition systems for $i = 1, 2$ and let H be a set of actions with $H \subseteq \mathrm{Act}_1 \cap \mathrm{Act}_2$ and $\tau \notin H$. The synchronised transition system is given by

$$TS_1 \parallel_H TS_2 = (S_1 \times S_2, \mathrm{Act}_1 \cup \mathrm{Act}_2, \to, I_1 \times I_2, \mathrm{AP}_1 \cup \mathrm{AP}_2, L),$$

where $L\langle s_1, s_2\rangle = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\to$ is defined by the following three rules.

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1', s_2\rangle}\ \alpha \notin H \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1, s_2'\rangle}\ \alpha \notin H \qquad \frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1', s_2'\rangle}\ \alpha \in H$$

If $H = \mathrm{Act}_1 \cap \mathrm{Act}_2$, then we abbreviate $TS_1 \parallel_H TS_2$ by $TS_1 \parallel TS_2$.

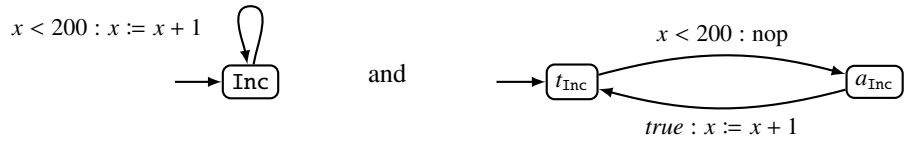
Note: We have that $\mid\mid\mid = \parallel_\emptyset$.

Program Graphs and Atomicity

The first part of the exercise is about different representations of programs as program graphs and the effect of separating tests and assignments. Suppose we are given the following program `Inc`.

Inc: **while** *true* **do if** $x < 200$ **then** $x := x + 1$

We may associate two different program graphs, the *atomic* $A_{\texttt{Inc}}$ and the *non-atomic* $N_{\texttt{Inc}}$, to this program:



The first program graph forces the atomic execution of the test and the assignment, while the second program graph separates these two. Since tests and the body of a while-loop are typically more complex, the second program graph is more realistic. These program graphs show by themselves the same behaviour. However, when combined with other processes that access the same variable, then the separation of test and assignment into the two states $t_{\texttt{Inc}}$ and $a_{\texttt{Inc}}$ matters, as we will see in the following exercise.

Exercise 1.

Let the two programs `Dec` and `Res` be given as follows.

Dec: **while** *true* **do if** $x > 0$ **then** $x := x - 1$

Res: **while** *true* **do if** $x = 200$ **then** $x := 0$

1. Give the atomic and non-atomic program graphs associated to the programs `Dec` and `Res`.
2. Show that $0 \leq x \leq 200$ is an invariant in the interleaving $A_{\texttt{Inc}} \,|||\, A_{\texttt{Dec}} \,|||\, A_{\texttt{Res}}$.
3. Show that there is an execution trace in the interleaving $N_{\texttt{Inc}} \,|||\, N_{\texttt{Dec}} \,|||\, N_{\texttt{Res}}$, in which x becomes negative.

Mutual Exclusion

Exercise 2.

Consider the following mutual exclusion algorithm that was proposed 1966 as a simplification of Dijkstra's mutual exclusion algorithm in case there are just two processes:

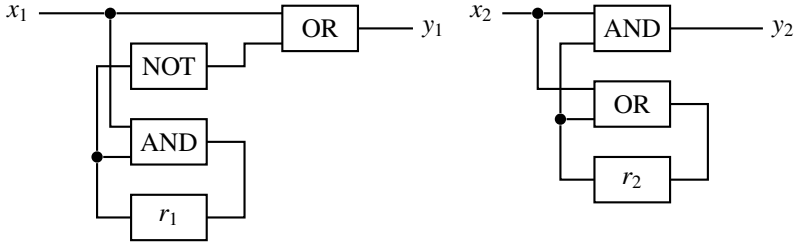
boolean array $b = [0; 1]$;
integer $k = 1$, i, j;
/* This is the program for computer i, which may be either 0 or 1, computer $j \neq i$ is the other one, 1 or 0 */
C0: $b(i) :=$ false;
C1: **if** $k \neq i$ **then**
C2: | **if** $\neg b(j)$ **then goto** C2;
| **else** $k := i$; **goto** C1;
else critical section;
$b(i) :=$ true;
remainder of program;
goto C0;

Here *C0*, *C1*, and *C2* are program labels, and the word "computer" should be interpreted as process.

1. Give the program graph representations for a single process. (A pictorial representation suffices.)
2. Give the reachable part of the transition system of $P_1 \parallel P_2$.
3. Check whether the algorithm indeed ensures mutual exclusion, that is, check whether that there is no reachable state in which both processes are in their critical section.

Sequential Hardware Circuits

In this part of the exercise, we deal with a particularly simple kind of computational system: sequential hardware circuits. Such a circuit has n inputs given by a set X of names, m outputs in Y and may store internally previous results in k registers in R. For simplicity, we assume that all three sets are disjoint. The circuit operates on Boolean values (0, 1), and computes in each (discrete) time step the output and new register values depending on the inputs and the previous register values. Such two circuits are displayed below.



Each circuit has one input, output and register. For the first circuit, we have $X_1 = \{x_1\}$, $Y_1 = \{y_1\}$ and $R_1 = \{r_1\}$, while for the second we have $X_2 = \{x_2\}$, etc. To describe circuits formally, let $\mathbb{B}$ be the set $\mathbb{B} = \{0, 1\}$ of Boolean values. We define *valuations* for a given set U to be the set of functions from U into $\mathbb{B}$:

$$\mathrm{Val}(U) = \{\sigma \mid \sigma \colon U \to \mathbb{B}\}.$$

In what follows, we will also need to restrict valuations. Thus, given $V \subseteq U$ and $\sigma \in \mathrm{Val}(U)$, we let $\sigma|_V \in \mathrm{Val}(V)$ be the valuation $\sigma \colon V \to \mathbb{B}$ with $\sigma|_V(x) = \sigma(x)$. A circuit is now given by a map

$$f \colon \mathrm{Val}(X \uplus R) \to \mathrm{Val}(Y \uplus R).$$

For example, the first circuit is given a the map $f_1 \colon \mathrm{Val}(\{x_1, r_1\}) \to \mathrm{Val}(\{y_1, r_1\})$ that is defined by the two cases $f_1(\sigma)(y_1) = \sigma(x_1) \vee \neg\sigma(r_1)$ and $f_1(\sigma)(r_1) = \sigma(x_1) \wedge \sigma(r_1)$.

Given an initial valuation $\rho_0 \in \mathrm{Val}(R)$, we can associate a transition system $(S, \mathrm{Act}, \longrightarrow, I, \mathrm{AP}, L)$ to such a circuit as follows. The states S are valuations of inputs and registers, that is, $S = \mathrm{Val}(X \uplus R)$; actions are irrelevant, hence $\mathrm{Act} = \{\tau\}$; the initial states set the registers to their initial values:

$$I = \{\sigma \mid \sigma|_R = \rho_0\};$$

transitions model exactly the computations given by f:

$$\sigma \longrightarrow \tau \text{ iff } f(\sigma)\big|_R = \tau|_R.$$

Finally, the atomic proposition are all variables $\mathrm{AP} = X \uplus Y \uplus R$ and the labelling is given by those variables that are true at a given state:

$$L(\sigma) = \{x \in X \;\big|\; \sigma(x) = 1\} \cup \{r \in R \;\big|\; \sigma(r) = 1\} \cup \{y \in Y \;\big|\; f(\sigma)(y) = 1\}.$$

For convenience, we simplify the graphical presentation of the states in the above described TS. Suppose $\sigma \in \mathrm{Val}(X_1 \uplus R_1)$ with $\sigma(x_1) = 1$ and $\sigma(r_1) = 0$, then we draw the state σ in the TS as follows.

$$\boxed{\begin{array}{c}\{x_1, y_1\} \\ x_1 = 1\ r_1 = 0\end{array}}$$

Exercise 3.

Consider the two sequential hardware circuits from above.

1. Give the map that describes the second circuit.
2. Assuming that the initial values of the registers are $r_1 = 0$ and $r_2 = 1$, give the associated transition systems of both hardware circuits in graphical notation.
3. Determine the reachable part of the interleaving of these transition systems.

Properties of Interleaving

Exercise 4.

Give an example of program graphs PG_1 and PG_2, such that $TS(PG_1) \;|||\; TS(PG_2)$ has evaluations as states that are impossible in $TS(PG_1 \;|||\; PG_2)$ *Hint: In the interleaving* $TS(PG_1) \;|||\; TS(PG_2)$ *the variables of* PG_1 *and* PG_2 *are renamed and thus not shared.*

Exercise 5.

Show that the handshaking operator $\|$ is associative. That is, show for arbitrary transition systems TS_1, TS_2, TS_3 that $(TS_1 \;\|\; TS_2) \;\|\; TS_3$ and $TS_1 \;\|\; (TS_2 \;\|\; TS_3)$ are essentially the same transition system.