Introduction to Model Checking 2015: Exercise 2.

Supervised by: Dr. Prof. Joost-Pieter Katoen

Hand in before: 29th April — Assisted by: Dr. N. Jansen & S. Chakraborty

---

Exercise 1 *(Point 4)*

The processes are numbered P_0 and P_1. For convenience, when representing P_i, we use P_j to denote the other process; that is, j equals $1 - j$. A solution to critical section problem must satisfy the following three requirements.

1. ***Mutual Exclusion**. If P_i is executing in its critical section, then no other processes can execute in their critical section.*
2. ***Progress**. If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.*
3. ***Bounded Waiting**. A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.*

1. *Consider Peterson's solution to critical section problem for two processes with two shared data items.*

```
        int turn;
        boolean flag[2];

    do {

        flag[i]=True;
        turn = j;
        while (flag[j] && turn == j);

            Critical section

        flag[i]= False

            Remainder Section
    }while(True);
```

   (a) *Draw the program graph.*

   (b) *Deduce from the program graph that, mutual exclusion, progress and bounded waiting holds.*

   (c) *What is the bound of the bounded waiting?*

2. *Consider the following simple solution with one shared variable,*

```
    int turn;
```

do {

```
turn = i;
while ( turn == j);
```

Critical section

```

```

Remainder Section

}*while(True);*

(a) *Draw the program graph.*

(b) *Verify using the program graph, whether mutual exclusion, progress and bounded waiting holds.*

3. *Consider the following little less simple solution with one shared variable,*

```
int turn;
```

do {

```
turn = i;
while ( turn == j);
```

Critical section

```
turn =j
```

Remainder Section

}*while(True);*

(a) *Draw the program graph.*

(b) *Verify using the program graph, whether mutual exclusion, progress and bounded waiting holds.*

Exercise 2 *(Point 3)*

1. *Show that, in general, the handshaking $\|_H$ operator* **is not** *associative, i.e.*

$$(\mathsf{T}_1 \|_H T_2) \|_{H'} \mathsf{T}_3 \neq \mathsf{T}_1 \|_H (\mathsf{T}_2 \|_{H'} \mathsf{T}_3)$$

2. *Show that the handshaking operator $\|$ that forces transition systems to synchronize over their common actions* **is** *associative. That is, show that*

$$(\mathsf{T}_1 \| \mathsf{T}_2) \| \mathsf{T}_3 \quad = \quad \mathsf{T}_1 \| (\mathsf{T}_2 \| \mathsf{T}_3)$$

where $\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3$ are arbitrary (finite) transition systems.

Hint: One approach to prove the above statement is to show that the transition system on the left hand side is isomorphic to the transition system on the right hand side of the equation. For this observe that the state space of both systems is $S_1 \times S_2 \times S_3$ (even though not necessarily all states are reachable). You can define a mapping that relates those states of the two transition systems that are equal component wise, i.e. $\langle\langle s_1, s_2\rangle, s_3\rangle \approx \langle s_1, \langle s_2, s_3\rangle\rangle$. *From here argue why this mapping is a bijection and why it preserves the transition relation.* When you argue about a transition with some action α you need to make a case distinction:

1.) $\alpha \in Act_1 \backslash (Act_2 \cup Act_3)$

2.) $\alpha \in (Act_1 \cap Act_2) \backslash Act_3$

3.) $\alpha \in Act_1 \cap Act_2 \cap Act_3$

You may dismiss all other cases because they are symmetric. Also keep in mind that a state can have several successors for one action.

Exercise 3 *(Point 3)*

In channel systems, values can be transferred from one process to another process. According to the lecture, the set of transitions of a program graph $PG = (\mathsf{Loc}, \mathsf{Act}, \mathsf{Effect}, \rightarrow, \mathsf{Loc}_0, g_0)$ *over* $(\mathsf{Var}, \mathsf{Chan})$ *is defined as*

$$\rightarrow \subseteq \mathsf{Loc} \times (Cond(Var) \times \mathsf{Act}) \times \mathsf{Loc} \quad \cup \quad \mathsf{Loc} \times \mathsf{Comm} \times \mathsf{Loc}$$

where $\mathsf{Comm} = \{c!v, c?x \mid c \in \mathsf{Chan}, v \in \mathsf{dom}(c), x \in \mathsf{Var}$ *with* $\mathsf{dom}(x) \supseteq \mathsf{dom}(c)\}$.

Here we consider two extensions to this definition. Give a formal definition of the transition system semantics of a channel system $CS = [PG_1 | \cdots | PG_n]$ *where*

1. *In asynchronous communication a channel shall always accept a sent value. If the channel is full it will simply drop the oldest element from its FIFO queue.*

2. *In synchronous message passing a channel may broadcast a value. That is, if several processes are willing to receive a value from a channel they will all receive it (instead of only one of them).*