Swansea University
Department of Computer Science

U Berger
November 29, 2022

CM385/CS-CM85 Modelling and Verification Techniques

Solutions and Feedback to the Coursework

Question 1 An LTS (S, α) over an alphabet A is called *trace-deterministic* if for all states $s, s_1, s_2 \in S$ and every label $a \in A$,

$$\text{if } s \xrightarrow{a} s_1 \text{ and } s \xrightarrow{a} s_2, \text{ then } s_1 =_{\mathrm{T}} s_2.$$

Let (S, α) and (T, β) be trace-deterministic LTSs.

Prove: For all states $s \in S$ and $t \in T$, if $s =_{\mathrm{T}} t$, then $s \sim t$.

Your proof should be a copy of the proof of part (b) of the Theorem on Page 8 of the lecture notes `verification-lts.pdf` where exactly one sentence is altered appropriately.

[30 marks]

Solution.

We modify the corresponding proof in the course notes which makes the stronger assumption of determinacy. The differences are *highlighted*.

We assume that the LTS is *trace-deterministic* and have to show that $s =_{\mathrm{T}} t$ implies $s \sim t$. We show this by coinduction. This means that we have to find a bisimulation R such that whenever $s =_{\mathrm{T}} t$, then sRt.

We simply define $sRt :\Leftrightarrow s =_{\mathrm{T}} t$. Hence, it suffices to show that $=_{\mathrm{T}}$ is a bisimulation. This means we have to show: If $s =_{\mathrm{T}} t$, then

(1) $\forall a \in A \forall s' \in S(s \xrightarrow{a} s' \Longrightarrow \exists t' \in T(t \xrightarrow{a} t' \wedge s' =_{\mathrm{T}} t'))$

(2) $\forall a \in A \forall t' \in T(t \xrightarrow{a} t' \Longrightarrow \exists s' \in S(s \xrightarrow{a} s' \wedge s' =_{\mathrm{T}} t'))$

Hence assume $s =_{\mathrm{T}} t$, that is $\mathrm{Traces}(s) = \mathrm{Traces}(t)$. In order to show (1) assume further $s \xrightarrow{a} s'$. We have to find $t' \in T$ such that $t \xrightarrow{a} t'$ and $s' =_{\mathrm{T}} t'$.

Since $s \xrightarrow{a} s'$, we have $\langle a \rangle \in \mathrm{Traces}(s)$. Hence $\langle a \rangle \in \mathrm{Traces}(t)$ because $\mathrm{Traces}(s) = \mathrm{Traces}(t)$. Therefore, there exists some $t' \in T$ such that $t \xrightarrow{a} t'$. We show that

this t' is as required. Hence it remains to show that $s' =_{\mathrm{T}} t'$, that is $\mathrm{Traces}(s') = \mathrm{Traces}(t')$.

We show first $\mathrm{Traces}(s') \subseteq \mathrm{Traces}(t')$: Assume $w \in \mathrm{Traces}(s')$. Then $\langle a \rangle \frown w \in \mathrm{Traces}(s)$. Since $\mathrm{Traces}(s) = \mathrm{Traces}(t)$, it follows $\langle a \rangle \frown w \in \mathrm{Traces}(t)$. Therefore, there exists $t'' \in T$ such that $t \xrightarrow{a} t''$ and $w \in \mathrm{Traces}(t'')$. *Because the LTS is trace-deterministic, we must have* $t' =_{\mathrm{T}} t''$. Hence $w \in \mathrm{Traces}(t')$.

The proof that $\mathrm{Traces}(t') \subseteq \mathrm{Traces}(s')$ is similar.

The proof of formula (2) is similar.

Question 2

Consider the following statements about trace equivalence ($=_{\mathrm{T}}$) and bisimilarity ($\sim$) of processes (where a is a visible event, different from the termination event, that is, $a \notin \{\tau, \checkmark\}$):

(1) $(a \to P) \,\Box\, (a \to Q) \quad =_{\mathrm{T}} \quad a \to (P \,\Box\, Q)$

(2) $(a \to P) \,\Box\, (a \to Q) \quad \sim \quad a \to (P \,\Box\, Q)$

Which of these statements are true for arbitrary processes P and Q?

In each case either prove trace equivalence respectively bisimilarity or give a concrete counterexample.

[30 marks]

Solution.

Regarding the firing rules of the operator $\Box$ (external choice) see the lecture notes `verification-csp.pdf` available on Canvas.

(1) $(a \to P) \,\Box\, (a \to Q) \quad =_{\mathrm{T}} \quad a \to (P \,\Box\, Q)$?

The statement is true for all processes P and Q.

First, one easily sees that in general

(i) $\mathrm{Traces}(U \,\Box\, V) = \mathrm{Traces}(U) \cup \mathrm{Traces}(V)$, and

(ii) $\mathrm{Traces}(a \to U) = \{\langle\rangle\} \cup (\langle a \rangle \frown \mathrm{Traces}(U))$

where $\langle a \rangle \frown X := \{\langle a \rangle \frown w \mid w \in X\}$. Furthermore, note that clearly

(iii) $\langle a \rangle \frown (X \cup Y) = (\langle a \rangle \frown X) \cup (\langle a \rangle \frown Y)$.

Therefore, by computing the left hand side, we get:

$$\begin{aligned}\mathrm{Traces}((a \to P) \,\Box\, (a \to Q)) &\overset{(i)}{=} \mathrm{Traces}(a \to P) \cup \mathrm{Traces}(a \to Q)\\ &\overset{(ii)}{=} \underline{(\{\langle\rangle\} \cup (\langle a\rangle \frown \mathrm{Traces}(P))) \cup (\{\langle\rangle\} \cup (\langle a\rangle \frown \mathrm{Traces}(Q)))}\end{aligned}$$

Computing the right hand side yields:

$$\begin{aligned}\mathrm{Traces}(a \to (P \,\Box\, Q)) &\overset{(ii)}{=} \{\langle\rangle\} \cup (\langle a\rangle \frown \mathrm{Traces}(P \,\Box\, Q))\\ &\overset{(i)}{=} \{\langle\rangle\} \cup (\langle a\rangle \frown (\mathrm{Traces}(P) \cup \mathrm{Traces}(Q)))\\ &\overset{(iii)}{=} \underline{\{\langle\rangle\} \cup (\langle a\rangle \frown \mathrm{Traces}(P)) \cup (\langle a\rangle \frown \mathrm{Traces}(Q))}\end{aligned}$$

We see that the two (underlined) results differ only in the order of the unions and the repeated occurrence of $\{\langle\rangle\}$, which doesn't make a difference. Therefore, the two results are equal.

(2) $(a \to P) \,\Box\, (a \to Q) \quad \sim \quad a \to (P \,\Box\, Q)$?

This is *not* true in general.

A counterexample is obtained by setting $P = b \to \mathrm{STOP}$ and $Q = \mathrm{STOP}$ (it is irrelevant whether the events a and b are different or equal).

Set $S_1 = (a \to P) \,\Box\, (a \to Q)$ and $S_2 = a \to (P \,\Box\, Q)$.

Then the attacker wins the bisimulation game starting with the configuration (S_1, S_2) by choosing the transition

$$S_1 \xrightarrow{a} Q.$$

The defender's only possible response is

$$S_2 \xrightarrow{a} (P \,\Box\, Q),$$

so the new configuration is $(Q, P \,\Box\, Q)$. Next, the attacker chooses

$$(P \,\Box\, Q) \xrightarrow{b} \mathrm{STOP}$$

which the defender cannot respond to since $Q = \mathrm{STOP}$.

Other counterexamples for (2) are possible.

Remark. In one student answer it was stated that the processes $(a \to P) \,\Box\, (a \to Q)$ and $a \to (P \,\Box\, Q)$ are bisimilar if and only if P and Q are bisimilar. This is actually true, and it is a useful exercise to prove this. On the basis of this observation, every pair of non-bisimilar processes P and Q would yield a counterexample to (2).

Question 3

Consider the following definition of a robot that can repeatedly report its positions, move to the left or right (within a given finite range), or do some work.

```
min = 0
max = 5

Range = {min..max}

datatype Direction = L | R

channel move : Direction
channel position : Range
channel work

inc(x) = if x < max then x+1 else x
dec(x) = if x > min then x-1 else x

Robot(x) = position.x -> Robot(x)   []
           move.L -> Robot(dec(x))  []
           move.R -> Robot(inc(x))  []
           work -> Robot(x)
```

Suppose doing work empties the robot's battery so that it needs at least two movements to recharge the battery (for simplicity, we count as movement every event `move.x` where `x` is `L` or `R`, even if the robot is at one end of its range and hence may not change its position).

We assume that, initially, the robot starts at position `min` and its battery is empty.

Define a process `Battery(n)`, with an integer parameter `n`, and a suitable synchronisation set `X` such that the process

```
Robot(min) [| X |] Battery(0)
```

models this behaviour.

[40 marks]

Solution.

```
Battery(n) = if n < 2
             then move?x -> Battery(n+1)
             else work -> Battery(0) [] move?x -> Battery(2)

System = Robot(min) [| {| move, work |} |] Battery(0)
```

An alternative way of defining the battery, given by some students, is:

```
charge(n) = if n < 2 then n+1 else n

Battery(n) = move?x -> Battery(charge(n))
             []
             (n == 2 & work -> Battery(0))
```

This solution makes it even clearer that the movements of the robot are not constraint.

For a solution to be correct, it is important that the definition of `Robot(n)` is not changed.

Feedback

Here is some general feedback. Individual feedback is given on Canvas.

Overall, I am impressed by the amount of work you invested in this coursework. This paid off by many very good solutions and a high average mark.

There was a big variation in the quality of solutions though. This indicates that there are a some key points of understanding which some students mastered very well while others are struggling. This specifically concerns the notions of trace equivalence, bisimilarity, and external choice, the meanings of which do not seem to be well understood by many of you.

A general mistake was that some solutions didn't answer the question at hand, but answered something else. To prevent similar mistakes in the exam, my advice is to carefully read what a question asks and then answer it to the point.

Question 1

There were some perfect solutions.

However, some students did not read or understand it properly and therefore missed the key point. The only difference of this question to the corresponding theorem in the course notes is that the assumption 'deterministic' is weakened to 'trace-deterministic'.

The proof is a copy of the proof in the notes. Apart from the slightly different statement of the problem (which was also missed by many) the only difference in the proof is that the phrase

> *Because the LTS is deterministic, we must have* $t' = t''$

is replaced by

> *Because the LTS is trace-deterministic, we must have* $t' =_{\mathrm{T}} t''$

Question 2

The quality of the solutions varied, though some were nearly perfect.

Both parts of this question ask first for a *decision* whether the statement is true or false, and then for a *justification* of the decision. A common mistake was to omit the decision, or to state the decision only after the justification.

In part (a) the main problem was that the proof of trace equivalence was either not correct or to vague. Also, many confused external choice with parallel composition.

In part (b) it was often the case that either the decision was incorrect, or the decision was correct but a concrete counterexample was missing.

Question 3

The solutions were generally very good. The main purpose of this question was to test your understanding of the controlled interaction of two processes through the generalised parallel operator. I am pleased that you understood this concept.

Two common mistakes were:

> The process `Robot(n)` was modified.
>
> The process `Battery(n)` constrained the robot's movement, so that it could not move if the battery was fully charged.
>
> The synchronisation set was wrong or not specified at all.