CSC385/CSCM85
Verification Coursework
17 November 2023

Authors:
Matthew Wilkinson: 2012007 CSC385
Linda Mafunu: 2216686 CSCM85
Eve Gittins: 2011863 CSCM85

Question 1

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 Pagge 8 of the lecture notes `verification-lts.pdf` where exactly one sentence is altered appropriately.

[30 marks]

Proof:

- we assume that the LTSs (S, α) and (T, β) are trace-deterministic. show that all states $s \in S$ and $t \in T$, if $s =_T t$, then $s \sim t$.
- using coinductive proof to find bisimulation R such that whenever $s =_T t$, then sRt

defining $sRt \equiv s =_T t$
we aim to show that $=_T$ is a bisimulation

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

2) $\forall a \in \forall t' \in T(t \xrightarrow{a} t' \rightarrow \exists s' \in S(s \xrightarrow{a} s' \wedge s' =_T t'))$

- Assume $s =_T t$ means $Traces(s) = Traces(t)$
- To prove (1), assume further $s \xrightarrow{a} s'$

we must find that $t' \in T$ such that $t \xrightarrow{a} t'$ and $s' =_T t'$

for $s \xrightarrow{a} s'$ we have $\langle a \rangle \in Traces(s)$ as $Traces(s) = Traces(t)$ in the trace-deterministic case, $\langle a \rangle \in Traces(t)$. such there exists $t' \in T$ such that $t \xrightarrow{a} t'$. To complete: we need to show that $s' =_T t'$, that meaning $Traces(s') = Traces(t')$

- $Traces(s') \subseteq Traces(t')$: Assume $w \in Traces(s')$
- then $\langle a \rangle \cdot w \in Traces(s)$.
- as $Traces(s) = Traces(t)$, $\langle a \rangle \cdot w \in Traces(t)$
- there exists $t'' \in T$ such that $t \xrightarrow{a} t''$ and $w \in Traces(t'')$
- As LTS is trace-deterministic, $t' = t''$, hence $w \in Traces(t')$

Proving $Traces(t') \subseteq Traces(s')$ is similar

Question 2

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

(1) $(a \to P) \,\Box\, (a \to Q) \quad =_T \quad a \to (P \,\Box\, Q)$

(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]

Example

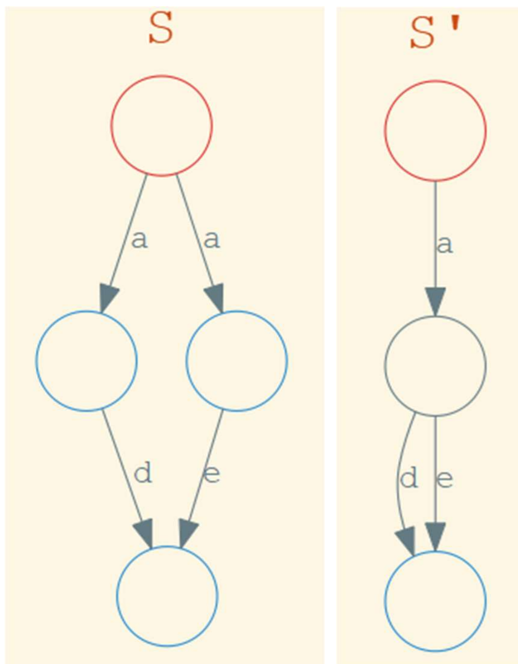
channel a, d, e

P = d -> STOP

Q = e -> STOP

S = (a -> P) [] (a -> Q)

S' = a -> (P [] Q)



(1) $(a \to P) \,\Box\, (a \to Q) \quad =_T \quad a \to (P \,\Box\, Q)$

LHS:

S = (a ->P) [] (a ->Q)

Trace(S) = {<>, a, ad, ae}

RHS:

S' = a -> (P [] Q)

Trace(S') = {<>, a, ad, ae}

S =T S' and S' =T S

For all arbitrary processes P and Q

LHS: S = (a ->P) [] (a ->Q)

Trace (a -> P) = {<>} ∪ {<a> ∧ W | W ∈ Trace(P)

Trace (a -> Q) = {<>} ∪ {<a> ∧ W | W ∈ Trace(Q)

S = (a ->P) [] (a ->Q) = Trace (a -> P) ∪ Trace (a -> Q)

RHS: S' = a -> (P [] Q)

R = Trace (P [] Q) = Traces(P) U Traces(Q)

S' = Trace (a -> R) = {<>} ∪ {<a> ∧ W | W ∈ Trace(R)

S and S' have equivalent traces

Trace(S) = {<>, a, a&Traces(P), a&Traces(Q)}

Trace(S') = {<>, a, a&Traces(P), a&Traces(Q)}

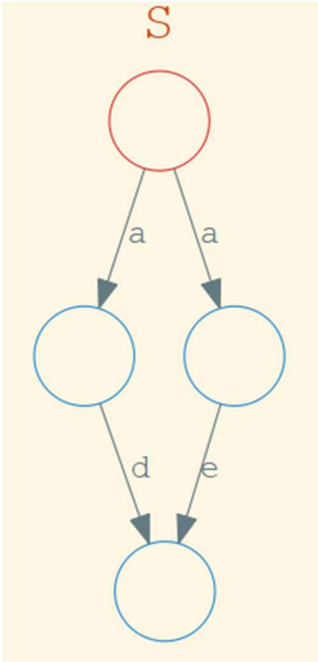
Therefore, statement (1) is true.

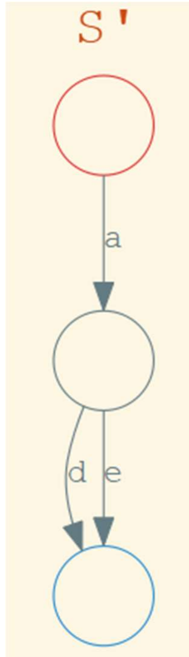
$$(2)\ \ (a \to P) \,\square\, (a \to Q) \quad \sim \quad a \to (P \,\square\, Q)$$

(2) is false.

Trace equivalence implies bisimilarity if both systems are deterministic. S and S' are trace equivalent, however S is not deterministic (i.e., there is a difference between the traces since Trace(S') branches off after <*a*> action compared to Trace(S) it branches off from the start) therefore statement (2) is false. This can be shown using the bisimulation game shown below.

Concrete counterexample using the bi-simulation game.





S = Defender

S’ = Attacker

1) S’ chooses a -> ((d [] e) -> STOP)
2) S chooses a -> (d -> STOP)
3) S’ chooses e-> STOP
4) S cannot choose e -> STOP

S’ has won.

Attacker S’ has the winning strategy as when it picks <a>, S must match <a> with either a-> (d->STOP) OR a -> (e -> STOP). Whichever S picks, S’ picks the opposite choice and so S cannot win.
As the defender cannot win, S and S’ are not bisimilar.

Question 3

-- initialise min and max

min = 0

max = 5

-- set the range

Range = {min..max}

-- set the directions robot will move

datatype Direction = L | R

-- declare the channels

channel move: Direction

channel position: Range

channel work

-- implement the functions for moving robot positions

inc(x) = if x < max then x+1 else x

dec(x) = if x > min then x-1 else x

-- implement the process for Robot(x)

Robot(x) = position.x -> Robot(x) []

move.L -> Robot(dec(x)) []

move. R -> Robot(inc(x)) []

work -> Robot(x)

--set of any moves left or right must be in sync

x = {| move |}

--set of move left or right must be in parallel with work

y = {| move, work|}

-- robot should not hit the wall

Control(x) = x >min & move. L -> Control(x) []

x < max & move. R -> Control(x+1)

-- Uses the control system so the robot cannot move. L if in position min and cannot Move.R if in position max. Robot(x) process should be in sync with Control(x) process

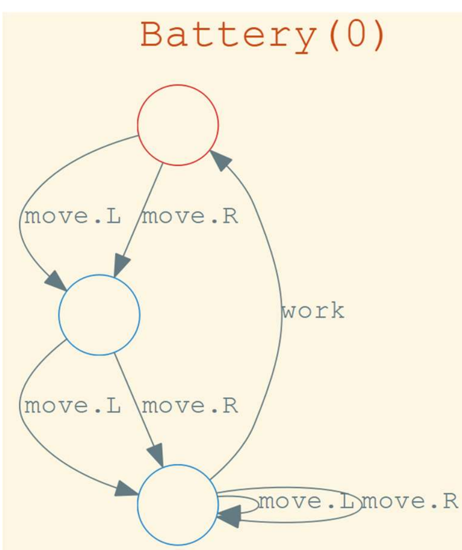
SystemC = Robot(min) [| x |] Control(min)

-- Battery does not force the robot to do work. If the battery is full the robot can move around freely and do work at any time.

-- Once work is done the battery is empty and the robot must move 2 positions to recharge the battery to do work again.

Battery(n) = (if n==2 then work -> Battery(n-2) else move?b -> Battery(n+1))

[] (n==2 & move?b -> Battery(n))



--Battery(n) process must be in sync with Robot(x) process with labels {move,work}.

-- Robot and battery without control system

SystemB = Robot(min) [| y |] Battery (0)

--Robot and battery with control system

SystemCB = SystemC [| y |] Battery(0)