Exercises:
Temporal logic
Test synthesis
Exercise

• Satisfaction of HML formulas:
  - \([[[true]]] = S, [[[false]]] = \emptyset\)
  - \([[[\neg \varphi]]] = S \setminus [[[\varphi]]]\)
  - \([[[\varphi_1 \land \varphi_2]]] = [[[\varphi_1]]] \cap [[[\varphi_2]]]\)
  - \([[[\varphi_1 \lor \varphi_2]]] = [[[\varphi_1]]] \cup [[[\varphi_2]]]\)
  - \([[[\langle \alpha \rangle \varphi ]]] = \{ s \in S \mid \exists s'. s -\alpha \rightarrow s' \land s' \in [[[\varphi]]]\}\)
  - \([[[[ \alpha ] \varphi ]]] = \{ s \in S \mid \forall s'. s -\alpha \rightarrow s' \Rightarrow s' \in [[[\varphi]]]\}\)

• Show that \(\langle \alpha \rangle \varphi_1 \lor \langle \alpha \rangle \varphi_2 = \langle \alpha \rangle (\varphi_1 \lor \varphi_2)\)
  - i.e., they are satisfied by the same subset of \(S\)
Exercise

Check the following properties on the LTS below.

1. $s \models \langle a \rangle \text{true}$
2. $s \models [b] \text{false}$
3. $s \models \langle a \rangle [b] \text{false}$
4. $s \models \langle a \rangle (\langle a \rangle \text{true} \land \langle b \rangle \text{true})$
5. $s \models [a] \langle a \rangle [a][b] \text{false}$
Exercise

Given the LTS below, compute the following sets:

1. \([[[a]\text{true}]]\]
2. \([[[a]\text{true} \land [b]\text{false}]]\]
3. \([[[a][b]\text{false}]]\)
Branching bisimulation

A branching bisimulation is a relation $R$ such that, if $(r, s) \in R$ and $r \xrightarrow{\mu} r'$ for some action, then either:

- $\mu = \tau$ and $(r', s) \in R$ or

- There is some $s'$ such that $s \xrightarrow{\tau} \cdots \xrightarrow{\tau} s' \xrightarrow{\mu} s''$ and $(r, s') \in R$, $(r', s'') \in R$.

- The same must hold for $s$ (if $s \xrightarrow{\mu} s'$, then either...)

- Note that two states that are strongly bisimilar are always branching bisimilar
Exercise

Are $s_0, t_0$ branching bisimilar?

$M_1 : s_0 \xrightarrow{A} s_1 \xrightarrow{B} s_2$

$M_2 : t_0 \xrightarrow{A} t_1 \xrightarrow{B} t_2$

$\tau$

$\tau$

$\tau$

$\tau$

$\tau$
Exercise

Are $s_0, t_0$ branching bisimilar?

$M_1 : s_0 \xrightarrow{A} s_1 \xrightarrow{C} s_2 \xrightarrow{\tau} s_3 \xrightarrow{B} s_4$

$M_2 : t_0 \xrightarrow{\tau} t_1 \xrightarrow{A} t_2 \xrightarrow{\tau} t_3$
Exercise

- Check $i_i \ ioco s_j$ for the following IOLTSs
Addendum: why $i_4$ ioco $s_4$?

- In $i_4$, after $?a$, two things may happen:
  - $\neg x$
  - Quiescence
- In $s_4$, after $?a$, two things may happen:
  - $\neg x$
  - Internal action, then quiescence
- We are talking about input-output conformance
  - The $\tau$ action is not visible
  - Thus, for $s_4$, after $?a$ we see $\neg x$ or quiescence
  - Therefore, $i_4$ ioco $s_4$
Lab session:
CADP and TESTOR
Overview of JardJeron05 (1/2)

Example from the first paper about the TGV tool

First let’s take a look at the specification
- All files are in ~/Desktop/TESTOR/demo
- Open jard_jeron_05_spec.lnt
- Generate and view its LTS
- Take a look at jard_jeron_05.io
Overview of JardJeron05 (2/2)

Left: LTS of jard_jeron_05_spec.lnt

jard_jeron_05.io:

input
A
B
C

(X, Y, and Z are outputs)
JardJeron05: Test Purpose (1/2)

Take a look at jard_jeron_05_purpose.lnt

1. What behaviour will be tested by this purpose?
2. What will happen if a Z output is observed?
JardJeron05: Test Purpose (2/2)

Take a look at jard_jeron_05_purpose.lnt

1. What behaviour will be tested by this purpose?
   An output action !Y followed by an output action !Z

2. What will happen if a !Z output is observed?
   The behaviour after !Z is ignored (TESTOR_REFUSE)
JardJeron05: Systems under test

- You have 3 files jard_jeron_05_sut<n>.aut
  - n = 1, 2, 3
  - Ignore the other SUTs

- They are in aut (automaton) format
  - Take a look at them (with a text editor, or via cat)
  - Can you guess how the aut format works?

- You can turn them into BCG thanks to bcg_io:
  bcg_io jard_jeron_05_sut1.aut .bcg
Intermezzo: the AUT format

- First line: description of the LTS
  - des (\langle initial-state \rangle, \langle number-of-transitions \rangle, \langle number-of-states \rangle)

- All other lines: labelled transitions
  - (\langle from-state \rangle, \langle label \rangle, \langle to-state \rangle)

- This format predates BCG and has been largely supplanted by it
  - Pros: intuitive, can be read/written via a text editor
  - Cons: inefficient for large LTSs
On-the-fly testing of JardJeron05 (1/3)

• First, perform these 3 commands once:
  – `lnr.open jard_jeron_05_purpose.lnt generator`
  – `rename tgv.rename tp.bcg`
  – `mkfifo sut.input`
  – `mkfifo sut.output`

• Then, for each `sut.bcg`, perform these 2 commands:
  `bcg_execute -io sut.io sut.bcg > sut.output < sut.input`
  `testor -interactive -io sut.io tp.bcg < sut.output 2> sut.input`

• Write down the result
On-the-fly testing of JardJeron05 (2/3)

• What did we do?
  - Generate the BCG of our test purpose (-rename needed for compatibility)
  - `bcg_execute ... &e`: run our SUT in the background
  - `testor -interactive`: compute and run the CTG for our test purpose
  - We connected the output of the SUT to the input of the CTG (and vice versa) via named pipes (`sut.output` and `sut.input`)

• You should get these results:
  - SUT1 and SUT3: Pass
  - SUT2: Inconclusive
On-the-fly testing of JardJeron05 (3/3)

• Graphical representation of our testing setup:

- `bcg_execute` (running the SUT)
- `sut.output`
- `sut.input`
- `testor` (running the CTG)

• More information about named pipes:
  - [https://www.linuxjournal.com/article/2156](https://www.linuxjournal.com/article/2156)
Final remarks: nondeterministic SUTs

• If your SUT is nondeterministic, different runs may produce different results
  - Typically, this is fine (you want to explore different behaviours)
  - But sometimes you may not want it (e.g., you may want to reproduce a failure)

• You can force bcg_execute to always perform the same execution, by adding \texttt{--seed <n>}
  - \texttt{n} is a number \texttt{>= 0}