Miguel Caçador Peixoto
pg50657

Homework 1 - CCS and equivalences
Cyber-Physical Computation 2022/2023

Escola de Engenharia
Universidade do Minho

1 CCS analysis

1. For each of the CCS processes below, draw its transition system.

   (a) $A = a.b.0$

$$\begin{tikzcd} {} \arrow[r] & A \arrow[r, "a"] & b.0 \arrow[r, "b"] & 0 \end{tikzcd}$$

   (b) $B = A + a.0$

$$\begin{tikzcd} {} \arrow[r] & B \arrow[r, "a"] \arrow[rr, bend left, "a"] & b.0 \arrow[r, "b"] & 0 \end{tikzcd}$$

   (c) $C = (B|c.d.A) \setminus \{d\}$

$$\begin{tikzcd}
 & & (b.0|c.d.A) \setminus \{d\} \arrow[dr, "b"] \arrow[drrr, bend left, "c"] & & & \\
{} \arrow[r] & C \arrow[ur, bend left, "a"] \arrow[rr, "a"] \arrow[dr, bend right, "c"] & & (0|c.d.A) \setminus \{d\} \arrow[r, "c"] & (0|d.A) \setminus \{d\} & (b.0|d.A) \setminus \{d\} \arrow[l, "b"] \\
 & & ((a.b.0 + a.0)|d.A) \setminus \{d\} \arrow[urr, bend right, "a"] \arrow[urrr, bend right, "a"] & & &
\end{tikzcd}$$

2. Recall A and B processes from Ex. 1.

- Prove that $A \lesssim B$ or explain why not.

  $A \lesssim B$ signifies that A is simulated by B. To prove this, we need to demonstrate that for every transition in process A, there exists a corresponding transition in process B with the same action, such that the resulting states also satisfy the simulation relation. We proceed by exploring the state space of both processes A and B concurrently until we reach a terminal state or encounter a counterexample, achieving the following simulation:

  $$R = \{\langle A, B\rangle, \langle b.0, b.0\rangle, \langle 0, 0\rangle\} \tag{1}$$

  Since we have shown that for every transition in process A, there exists a corresponding transition in process B with the same action and the resulting states satisfy the simulation relation, we can conclude that $A \lesssim B$.

- Prove that $B \lesssim A$ or explain why not

  As the same manner as before, $B \lesssim A$ is true due to the following simulation:

  $$R = \{\langle B, A\rangle, \langle b.0, b.0\rangle, \langle 0, b.0\rangle, \langle 0, 0\rangle\} \tag{2}$$

- Prove that $A \sim B$ or explain why not.

  To prove that the two processes are bissimilar both their simulations need to have the same pairs.

  $$R_{A,B} = \{\langle A, B\rangle, \langle b.0, b.0\rangle, \langle 0, 0\rangle\} \tag{3}$$

  $$R_{B,A} = \{\langle B, A\rangle, \langle b.0, b.0\rangle, \langle 0, b.0\rangle, \langle 0, 0\rangle\} \tag{4}$$

  Since the pair $\langle 0, b.0\rangle$ does not appear in $R_{A,B}$, it's concluded the processes are not bissimilar.

- **[Hard]** Prove that, for all CCS processes P and Q:

  $$P + Q \sim Q + P \tag{5}$$

  In this exercise, we aim to prove that $P + Q$ is bisimilar to $Q + P$.

  A bisimulation is a relation between two systems that have a similar behavior, in which each state in one system corresponds to a state in the other system. To prove that $P+Q$ is bisimilar to $Q + P$, we need to define a simulation R that satisfies the following conditions:

  - For any pair of related states $\langle s, t\rangle \in R$, if s can perform an action a that leads to a new state s', then there must exist a related state t' such that t can perform the same action a and reach t', and $\langle s', t'\rangle \in R$.
  - The initial states $P + Q$ and $Q + P$ are related, i.e., $\langle P + Q, Q + P\rangle \in R$.
  - For any pair of related states $\langle s, t\rangle \in R$, if s is a final state (i.e., it cannot perform any further actions), then t must also be a final state.

  And so, its possible to define the simulation R as follows:

  $$R = \{\langle P + Q, Q + P\rangle\} U \{\langle \alpha, \alpha\rangle \,|\alpha \in P\} U \{\langle \beta, \beta\rangle \,|\beta \in Q\}$$

  Which comprises all possible pairs of states in $P + Q$ and $Q + P$, as well as all pairs of states within P and within Q - $\langle \alpha, \alpha\rangle$ and $\langle \beta, \beta\rangle$ respectively.

  Let's now show that R satisfies the bisimulation conditions:

(a) Suppose $\langle s,t\rangle \in R$ for some states s in $P+Q$ and t in $Q+P$, and s can perform an action a that leads to a new state s'. Then we have two cases to consider:

i) If s evolves through P, then there must exist a state $\alpha \in P$ such that s can perform action a and evolve to α. Since $\langle s,t\rangle \in R$, we know that there exists a related state t in $Q+P$ such that t can perform the same action a and evolve to a state t'. This is valid because the '+' operator denotes a non-deterministic choice - the system can either evolve following P or Q. In this particular case, it can choose to evolve following P. Since $\alpha \in P$, we have $\langle \alpha,\alpha\rangle \in R$, and thus, we also have $\langle s',t'\rangle \in R$.

ii) If s evolves through Q, then we can similarly find a related state t in $Q+P$ that evolves to s' upon performing a.

(b) $\langle P+Q, Q+P\rangle \in R$ since P and Q are trivially bisimilar to themselves.

(c) If $\langle s,t\rangle \in R$ and s is a final state, then s cannot perform any further actions. But since $\langle s,t\rangle \in R$, we know that t is related to s, so t must also be a final state.

Therefore, $P+Q$ is bisimilar to $Q+P$.

2 CCS modelling

Consider the 5 components below.

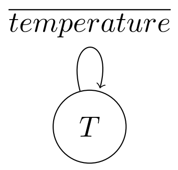
- T: A temperature sensor that periodically sends a temperature value;
- H: A humidity sensor that periodically sends a humidity value;
- C: A clock that sends a timestamp with the current time;
- O: An orchestrator that receives a temperature value, followed by a humidity value and by a timestamp, and in the end sends this data package;
- D: A display that receives data from the orchestrator and displays the content.

Consider each underlined word above to be an action of our CSS processes.

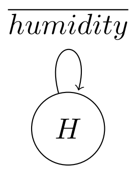
1. Specify each of these 5 components in CCS and draw their transition system.

   Specification of the 5 components in CSS:

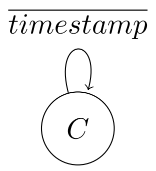
   - $T = (\overline{temperature}.T)$



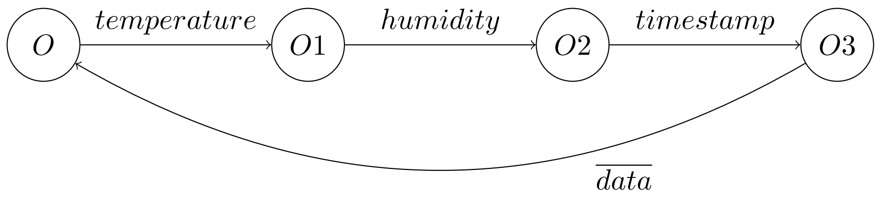
   - $H = (\overline{humidity}.H)$



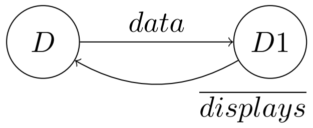
   - $C = (\overline{timestamp}.C)$



- $O = (temperature.O1)$
- $O1 = (humidity.O2)$
- $O2 = (timestamp.O3)$
- $O3 = (\overline{data}.O)$



- $D = (data.D1)$
- $D1 = (\overline{displays}.D)$



2. Specify a new component S of this system, which composes the 5 components above in parallel, imposing synchronisation of all actions except display.

$$S = (T|H|C|O|D) \setminus \{temperature, humidity, timestamp, data\}$$

3. Propose a variation of a similar system S2 in CCS with no orchestrator. In this variation:

   (a) the humidity sensor informs the temperature sensor, then
   (b) the temperature sensor informs the timestamp, then
   (c) the timestamp sends the whole data to the display; and finally
   (d) the display prompts the humidity sensor to restart the process.

   Specification of the new components and system S2:

   In order to keep the concurrence physolophy of the system, the new components where specified as follows:

   - $H' = (\overline{humidity}.restart.H')$
   - $T' = (humidity.\overline{temperature}.T')$
   - $C' = (temperature.\overline{timestamp}.C')$
   - $D' = timestamp.\overline{restart}.D'$

   And so,

$$S2 = (H'|T'|C'|D') \setminus \{temperature, humidity, timestamp, restart\}$$

   Which can be represented as follows:

$$\begin{tikzcd}
H' \arrow[r, "\overline{humidity}"] & H'0 \arrow[l, bend left, "\overline{restart}"]
\end{tikzcd}$$

$$\begin{tikzcd}
T' \arrow[r, "humidity"] & T'0 \arrow[l, bend left, "\overline{temperature}"]
\end{tikzcd}$$

$$\begin{tikzcd}
C' \arrow[r, "temperature"] & C'0 \arrow[l, bend left, "\overline{timestamp}"]
\end{tikzcd}$$

$$\begin{tikzcd}
D' \arrow[r, "timestamp"] & D'0 \arrow[l, bend left, "\overline{restart}"]
\end{tikzcd}$$

Note: In this problem it was assumed the display no longer displays the information due to that not being a requirement.

3 CCS modelling - mCRL2

Experiment with the tool mCRL2 (https://mcrl2.org). Use it to validate your S and S2 definitions above.

3.1 Validation S definition

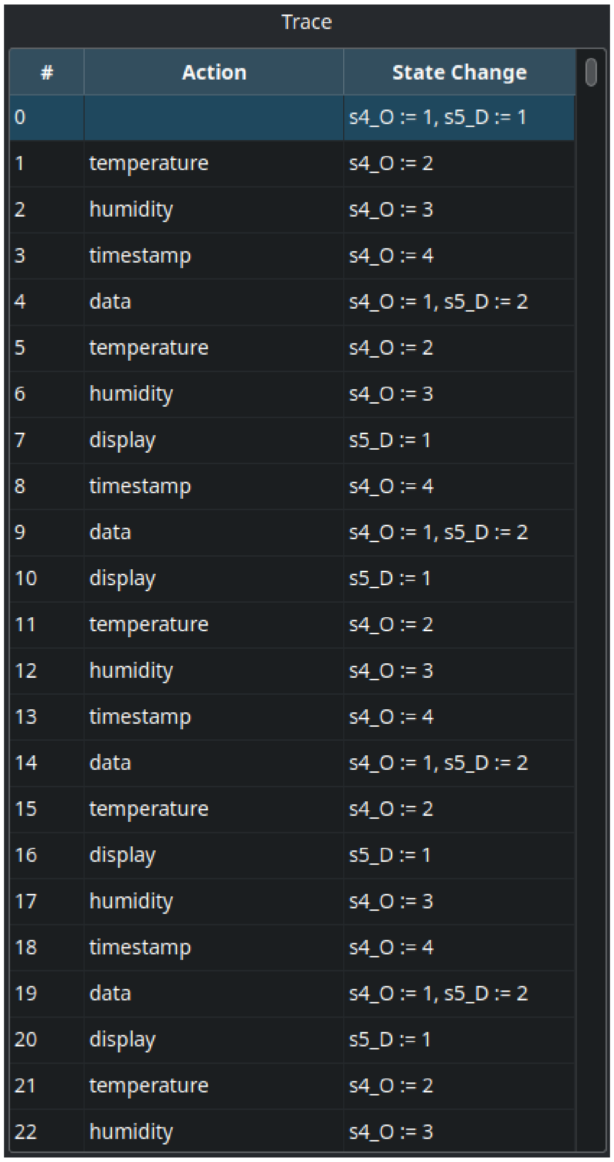
mCRL2 is a specification language for describing concurrent discrete event systems. It is accompanied with a toolset, that facilitates tools, techniques and methods for simulation, analysis and visualization of behaviour. So for validation the S definition specified in the previous section, some things need to be defined in mCRL2 IDE:

- Actions available in the system
- Processes, which are composed of action
- Initial state, defines the initial state and communication between processes. In here, we also need to specify the system allowed actions and describe the comunication between processes (e.g how actions synchronize)

And so, the following code was used to describe our system:

```
%------------------------------------------------------------
% Defining the actions of the system
% Note that the suffix '_emmit' means emission of a value (e.g., temperature_emmit for emitting a temperature value) and '_receive' the reception of a value (e.g., temperature_receive for receiving a temperature value).
%------------------------------------------------------------
act
  temperature, temperature_emmit, temperature_receive,
  humidity, humidity_emmit, humidity_receive,
  timestamp, timestamp_emmit, timestamp_receive,
  data, data_emmit, data_receive,
  display;
%------------------------------------------------------------
% Define the processes of our system
% T: The process that emits a temperature value and continues as T.
% H: The process that emits a humidity value and continues as H.
% C: The process that emits a timestamp value and continues as C.
% O: The process that receives temperature, humidity, and timestamp values and emits a data value. It then continues as O.
% D: The process that receives a data value, performs a display action, and continues as D.
%------------------------------------------------------------
proc
  T = temperature_emmit.T;
  H = humidity_emmit.H;
  C = timestamp_emmit.C;
  O = temperature_receive.humidity_receive.timestamp_receive.data_emmit.O;
  D = data_receive.display.D;
%------------------------------------------------------------
% Initial State:
% allow(): Defines allowed actions.
% comm(): Specifies how actions synchronize.
%------------------------------------------------------------
init
  allow(
    { temperature, humidity, timestamp, data, display },
    comm(
      {
        temperature_emmit|temperature_receive -> temperature,
        humidity_emmit|humidity_receive -> humidity,
        timestamp_emmit|timestamp_receive -> timestamp,
        data_emmit|data_receive -> data
      },
    T || H || C || O || D % Running all five processes concurrently
    )
);
```

By parsing and running a random play in mCRL2 we obtain the following trace:



Trace

#	Action	State Change
0		s4_O := 1, s5_D := 1
1	temperature	s4_O := 2
2	humidity	s4_O := 3
3	timestamp	s4_O := 4
4	data	s4_O := 1, s5_D := 2
5	temperature	s4_O := 2
6	humidity	s4_O := 3
7	display	s5_D := 1
8	timestamp	s4_O := 4
9	data	s4_O := 1, s5_D := 2
10	display	s5_D := 1
11	temperature	s4_O := 2
12	humidity	s4_O := 3
13	timestamp	s4_O := 4
14	data	s4_O := 1, s5_D := 2
15	temperature	s4_O := 2
16	display	s5_D := 1
17	humidity	s4_O := 3
18	timestamp	s4_O := 4
19	data	s4_O := 1, s5_D := 2
20	display	s5_D := 1
21	temperature	s4_O := 2
22	humidity	s4_O := 3

In here we can see that the actions 'temperature', 'humidity', 'timestamp', and 'data' always occur in a specific order due to synchronization. However, the action 'display' can take place after the 'data' action or between any other actions, provided that no two consecutive 'data' actions occur without a 'display' action in between. This behavior arises because synchronization is not imposed on the display action.

This behavior aligns with the specifications defined for process S.

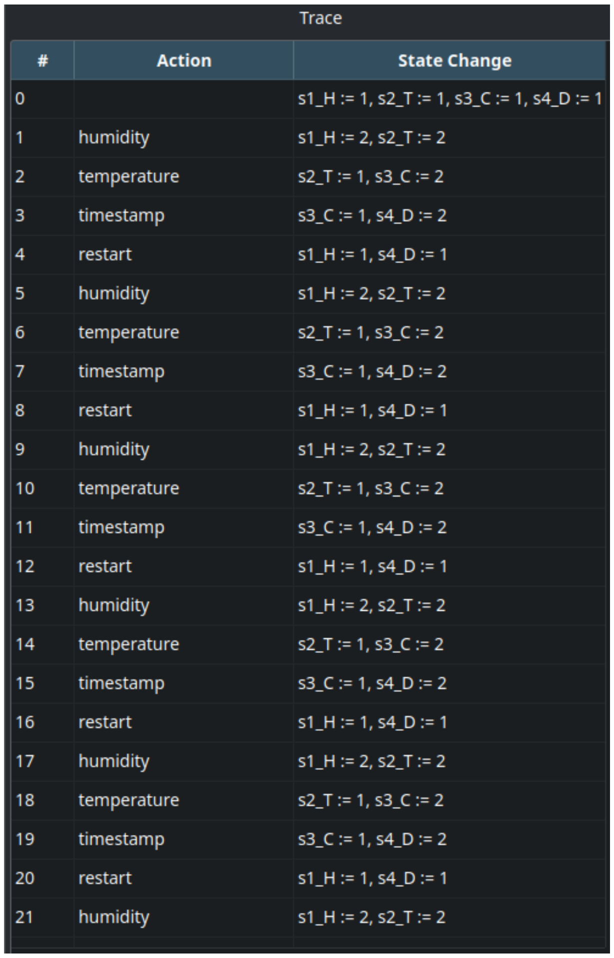
3.2 Validation S2 definition

As the same manner as before, the S2 definition was validated in mCRL2:

```
act
  temperature, temperature_emmit, temperature_receive,
  humidity, humidity_emmit, humidity_receive,
  timestamp, timestamp_emmit, timestamp_receive,
  restart, restart_emmit, restart_receive;
proc
  H = humidity_emmit.restart_receive.H;
  T = humidity_receive.temperature_emmit.T;
  C = temperature_receive.timestamp_emmit.C;
  D = timestamp_receive.restart_emmit.D;
```

```
init
  allow(
    { temperature, humidity, timestamp, restart },
    comm(
      {
        temperature_emmit|temperature_receive -> temperature,
        humidity_emmit|humidity_receive -> humidity,
        timestamp_emmit|timestamp_receive -> timestamp,
        restart_emmit|restart_receive -> restart
      },
    H || T || C || D % Running all five processes concurrently
    )
);
```

By parsing and running a random play in mCRL2 we obtain the following trace:



Trace

#	Action	State Change
0		s1_H := 1, s2_T := 1, s3_C := 1, s4_D := 1
1	humidity	s1_H := 2, s2_T := 2
2	temperature	s2_T := 1, s3_C := 2
3	timestamp	s3_C := 1, s4_D := 2
4	restart	s1_H := 1, s4_D := 1
5	humidity	s1_H := 2, s2_T := 2
6	temperature	s2_T := 1, s3_C := 2
7	timestamp	s3_C := 1, s4_D := 2
8	restart	s1_H := 1, s4_D := 1
9	humidity	s1_H := 2, s2_T := 2
10	temperature	s2_T := 1, s3_C := 2
11	timestamp	s3_C := 1, s4_D := 2
12	restart	s1_H := 1, s4_D := 1
13	humidity	s1_H := 2, s2_T := 2
14	temperature	s2_T := 1, s3_C := 2
15	timestamp	s3_C := 1, s4_D := 2
16	restart	s1_H := 1, s4_D := 1
17	humidity	s1_H := 2, s2_T := 2
18	temperature	s2_T := 1, s3_C := 2
19	timestamp	s3_C := 1, s4_D := 2
20	restart	s1_H := 1, s4_D := 1
21	humidity	s1_H := 2, s2_T := 2

In here we can see that the actions 'humidity', 'temperature', 'timestamp', and 'restart' always occur in this order, one after the other (as expected), due to the imposed synchronization and process architecture.

This behavior aligns with the specifications defined for process S2.