Comet – devoir à faire à la maison

Le devoir est à faire seul(e), et à rendre lors du cours du 23 novembre, ou, au plus tard, le 24 novembre dans le casier de D. Hirschkoff

Nota :

- *Vous pouvez tout à fait rédiger en français, l'énoncé est en anglais afin de faciliter la communication entre les enseignants.*
- *Vous êtes les bienvenus pour nous poser des questions et demander des précisions sur le DM ; tenez compte du fait que nous ne garantissons pas d'être réactifs la veille au soir...*

1 Asynchronous process calculi

Parts 1.1 and 1.2 are independent from eachother, although they share the theme of asynchrony.

1.1 Accs and asynchronous bisimilarity

We work in *asynchronous CCS*, written ACCS, which is defined by the following grammar:

$$P \quad ::= \quad a.P \;\big|\; \overline{a} \;\big|\; \tau.P \;\big|\; P_1|P_2 \;\big|\; P_1 + P_2$$

This is the *finite, public version* of the calculus. *Finite and public* refer to the fact that the calculus does not include neither replication nor restriction. *Asynchronous* refers to the fact that output is not a prefix: we can only write $\overline{a}$, which behaves like $\overline{a}.\mathbf{0}$ in CCS, but we cannot write $\overline{a}.P$. Therefore, the basic form of synchronisation in ACCS is: $a.P \mid \overline{a} \xrightarrow{\tau} P|\mathbf{0}$.

The LTS for ACCS has essentially the same rules as in CCS. It is defined by the following rules:

$$\frac{}{\overline{a} \xrightarrow{\overline{a}} \mathbf{0}} \qquad \frac{}{a.P \xrightarrow{a} P} \qquad \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \frac{P \xrightarrow{\overline{a}} P' \qquad Q \xrightarrow{a} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \qquad \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'}$$

(symmetrical versions of the rules for | and + are left implicit).

Based on this LTS, we define an auxiliary LTS, written $\xrightarrow{\mu}_1$, which is defined by the following two rules:

$$\frac{P \xrightarrow{\mu} P'}{P \xrightarrow{\mu}_1 P'} \qquad\qquad \frac{P|\overline{a} \xrightarrow{\tau} P'}{P \xrightarrow{a}_1 P'}$$

We define $\sim_1$ using $\xrightarrow{\mu}_1$, like we did in the course for $\sim$ for CCS. $\sim_1$ is called *asynchronous strong bisimilarity.* You can refer to a *1-bisimulation* if necessary in your answers.

Question 1.1 *Consider the equivalence between $a.(\overline{a}|P) + \tau.P$ and $\tau.P$, and prove that $\sim_1$ and $\sim$ do not coincide on ACCS.*

A relation $\mathcal{R}$ is a *2-bisimulation* whenever $P\mathcal{R}Q$ implies that

- if $P \xrightarrow{\overline{a}} P'$, there exists Q' s.t. $Q \xrightarrow{\overline{a}} Q'$ and $P'\mathcal{R}Q'$;
- if $P \xrightarrow{\tau} P'$, there exists Q' s.t. $Q \xrightarrow{\tau} Q'$ and $P'\mathcal{R}Q'$;

- if $P \xrightarrow{a} P'$, then
  - either there exists Q' s.t. $Q \xrightarrow{a} Q'$ and $P'\mathcal{R}Q'$,
  - or there exists Q' s.t. $Q \xrightarrow{\tau} Q'$ and $P'\mathcal{R}(Q'|\overline{a})$.

and symmetrically for the transitions of Q. (Note that 2-bisimulation is defined using $\xrightarrow{\mu}$, not $\xrightarrow{\mu}_1$.) Define $\sim_2$ as the greatest 2-bisimulation

Question 1.2 *Prove that $\sim_1$ and $\sim_2$ coincide on* Accs.

1.2 Asynchronous π

The Asynchronous π-calculus, written $A\pi$, is defined like Accs: it is like π, but only asynchronous outputs, of the form $\overline{a}b$, are allowed (not $\overline{a}b.P$). The "usual π" we saw in the course is the synchronous π. As we saw in the course, it comes in two versions, monadic (where one name is transmitted in each communication), and diadic (where two names are transmitted).

We consider the following transition in (synchronous) π:

$$a(x).P \mid \overline{a}b.Q \quad \xrightarrow{\tau} \quad P[b/x] \mid Q \ ,$$

and we want to "program" this synchronous interaction in $A\pi$ (like we did in the course, when we programmed diadic interaction in monadic π).

Question 1.3 *In this question, we work in diadic $A\pi$, where a pair of names is exchanged in each communication:*

$$a(x,y).P \mid \overline{a}\langle b,c\rangle \quad \xrightarrow{\tau} \quad P[b/x,c/y] \mid \mathbf{0}$$

Define an encoding of synchronous monadic π into diadic $A\pi$. Like in the course, the encoding should be "reasonable", in the sense that some form of atomicity should be respected.

Question 1.4 *A way to obtain an encoding of synchronous monadic π in monadic $A\pi$ is to compose the encoding you just defined with the encoding of diadic π into monadic π we saw in the course. But this is too easy/lazy.*

Define a simpler, more direct encoding of synchronous monadic π into monadic $A\pi$.

2 Language inclusion

Let S be a set and $\mathbf{Rel}_S$ be the lattice of relations over S. Recall from the course the function $r, s, t\colon \mathbf{Rel}_S \to \mathbf{Rel}_S$ defined for all relations $R \subseteq S \times S$ as

$$r(S) = \{(x,x) \text{ s.t. } x \in S\}, \quad s(R) = \{(x,y) \text{ s.t. } (y,x) \in R\}, \quad t(R) = \{(x,z) \text{ s.t. } (x,y) \in R \text{ and } (y,z) \in R\}.$$

A *pre-order* is a pair $(S, \sqsubseteq)$ where S is a set and $\sqsubseteq$ is a relation on S such that $\sqsubseteq$ is reflexive ($r(\sqsubseteq) \subseteq \sqsubseteq$), transitive ($t(\sqsubseteq) \subseteq \sqsubseteq$). A *partial order* is a pre-order $(S, \sqsubseteq)$ where $\sqsubseteq$ is also antisymmetric ($\sqsubseteq \cap s(\sqsubseteq) \subseteq r(\sqsubseteq)$).

An example of partial order is the set $2 = \{0,1\}$ together with the ordering $0 \sqsubseteq 0$, $0 \sqsubseteq 1$ and $1 \sqsubseteq 1$.

2.1 Deterministic automata

Let (S, o, t) be a deterministic automaton over the alphabet A. Recall from the course the map $[\![\cdot]\!]\colon S \to 2^{A^*}$ defined for all $x \in S$, $a \in A$ and $w \in A^*$ as

$$[\![x]\!](\varepsilon) = o(x) \quad \text{and} \quad [\![x]\!](aw) = [\![t(x)(a)]\!](w).$$

We call $[\![x]\!]$ the *language accepted* by the state x. For two states $x, y \in S$, we say that the language of x is included in the language of y, written $x \precsim y$, if and only if $[\![x]\!] \subseteq [\![y]\!]$. We call the relation $\precsim \subseteq S \times S$ *language inclusion.* Let $b' \colon \mathbf{Rel}_S \to \mathbf{Rel}_S$ be the map defined for all relations R as

$$b'(R) = \{(x, y) \text{ s.t. } o(x) \sqsubseteq o(y) \text{ and } (t(x)(a), t(y)(a)) \in R\}.$$

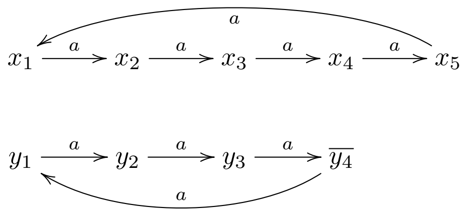
Question 2.1 *Prove the following facts on b'.*

1. *It is monotone (that is $b'(R_2) \subseteq b'(R_1)$ for all relations $R_2 \subseteq R_1$);*
2. *Its greatest fixed-point is language inclusion, in symbols $\nu b' = \precsim$.*

Given the previous characterisation of $\precsim$ as a greatest fixed-point, we can safely use coinduction to prove language inclusion. A postfixed point of b', namely a relation R such that $R \subseteq b'(R)$, is called a *b'-simulation.* The coinduction proof principle informs us that, in order to check whether $x \precsim y$ for two states $x, y \in S$, it is enough to show a b'-simulation R such that $(x, y) \in R$.

$$\frac{\exists R,\ \{(x, y)\} \subseteq R \subseteq b'(R)}{\{(x, y)\} \subseteq \precsim} \tag{1}$$

Question 2.2 *Show a b'-simulation proving $x_1 \precsim y_1$ in the following deterministic automaton.*



$$\begin{tikzcd}
x_1 \arrow[r, "a"] & x_2 \arrow[r, "a"] & x_3 \arrow[r, "a"] & x_4 \arrow[r, "a"] & x_5 \arrow[llll, bend right, "a"']
\end{tikzcd}$$

$$\begin{tikzcd}
y_1 \arrow[r, "a"] & y_2 \arrow[r, "a"] & y_3 \arrow[r, "a"] & \overline{y_4} \arrow[lll, bend left, "a"]
\end{tikzcd}$$

Like for language equivalence, the coinduction proof principle can be enhanced by mean of up-to techniques. In order to prove their soundness it is usually convenient to first prove compatibility of some basic techniques and then deduce the compatibility of the compound techniques.

Question 2.3 *For each of the following monotone maps $f \colon \mathbf{Rel}_S \to \mathbf{Rel}_S$ prove that it is compatible with b' (that is $fb'(R) \subseteq b'f(R)$ for all relations R) or give a counterexample.*

1. *r;*
2. *s;*
3. *t;*
4. *the equivalence closure, that is $e = (Id \cup r \cup s \cup t)^\omega$;*
5. *the reflexive and transitive closure $(\cdot)^\star = (Id \cup r \cup t)^\omega$;*

Question 2.4 *For each of the following statements prove it or give a counterexample.*

1. *Let $f, b \colon \mathbf{Rel}_S \to \mathbf{Rel}_S$ be two arbitrary monotone maps. If f is compatible with b, then $f(\nu b) \subseteq \nu b$.*
2. *Language inclusion is a pre-order.*
3. *Language inclusion is a partial order.*

Consider now the problem of automatically checking language inclusion of two states x and y of a deterministic automaton. One can proceed as in the case of language equivalence: by using b' in place of b, one attempts to build a b'-simulation relating the initial states x and y. We call this algorithm `Naive'`. As for the Hopcroft and Karp's algorithm, one can cut the search space by exploiting some up-to techniques (take the largest that you proved to be sound in Question 2.3). By analogy, we call the resulting algorithm `HK'`.

Question 2.5 *Give the pseudo-code, in the style seen at lesson, of the algorithms* `Naive'` *and* `HK'`. *Prove that they are sound and complete (that is, they return true iff the inclusion holds).* Hint: Use the results in the previous questions and adapt the proof seen at lesson.

Question 2.6 *Can* `HK'` *be more efficient than* `Naive'` *in terms of number of iterations? Give an example or sketch a proof.*

2.2 Non-deterministic automata

Recall from the course that a *join-semilattice* is a triple $(S, +, 0)$ such that $+ \colon S \times S \to S$ is associative $((x + y) + z = x + (y + z))$, commutative $(x + y = y + x)$ and idempotent $(x + x = x)$ and $0 \in S$ is its unit $(x + 0 = x)$. An *homomomorphism of join-semilattices* $f \colon (S_1, +_1, 0_1) \to (S_2, +_2, 0_2)$ is a function $f \colon S_1 \to S_2$ such that for all $x, y \in S_1$, $f(x +_1 y) = f(x) +_2 f(y)$ and $f(0_1) = 0_2$.

Examples seen at lessons are the join-semilattices $(\mathcal{P}(S), \cup, \emptyset)$ of sets of states and $(2^{A^\star}, \cup, \emptyset)$ of languages. Given a non-deterministic automaton (S, o, t) and the corresponding determised one $(\mathcal{P}(S), o^\sharp, t^\sharp)$, the morphism $[\![\cdot]\!] \colon (\mathcal{P}(S), \cup, \emptyset) \to (2^{A^\star}, \cup, \emptyset)$ is an example of homomorphism of join-semilattices.

Question 2.7

1. *Prove that every join-semilattice $(S, +, 0)$ induces a partial order $\sqsubseteq$ defined as $x \sqsubseteq y$ iff $x + y = y$.*
2. *Prove that every homomorphism of join-semilattices induces a monotone map between the corresponding partial orders.*

Question 2.8 *Give a good algorithm to check language inclusion of non-deterministic automata. Prove that it is sound and complete.*