Exercises on CCS, CCSK and RCCS

Gabriele Cecilia

Exercises 1

- Write a high-level explanation of the situation represented by the process $((a.P_1.\overline{b}.P_2|b.Q_1.Q_2)\backslash b) + ((a.Q_1.\overline{c}.Q_2|c.P_1.P_2)\backslash c)$ (with b and c not occurring free in P_1 and Q_1 respectively) taking inspiration from the following questions:
  - Can $b.Q_1.Q_2$ receive a message on b from any other process than $a.P_1.\overline{b}.P_2$?
  - Can P_1 and Q_1 execute at the same time?
  - Can P_2 and Q_1 execute at the same time?
  - Can Q_1 execute twice?

→ The given process can either behave as $((a.P_1.\overline{b}.P_2|b.Q_1.Q_2)\backslash b)$ or $((a.Q_1.\overline{c}.Q_2|c.P_1.P_2)\backslash c)$, without executing both; the structure of the two subprocesses is similar. In case the former is chosen, the process must first output a message through the channel a and then execute P_1 (as b does not occur free in P_1): we have no other choice because b can't be used to output a message yet, as the channel b is restricted (as a consequence, P_1 must execute before Q_1). After these two steps, the current process is $((\overline{b}.P_2|b.Q_1.Q_2)\backslash b)$. Next, we need to either receive or send a message through b, but the usage of b is restricted: this means that no communication via b with the outside is allowed, and the only possibility is to perform an internal communication between the two sides of the parallel composition along b. After this interaction, the process $((P_2|Q_1.Q_2)\backslash b)$ can behave as P_2 or Q_1 in any order, or at the same time (always, given that b does not occur free in them); it can also execute Q_2, only after Q_1 has executed, independently from the time P_2 gets executed.

In case the second process $((a.Q_1.\overline{c}.Q_2|c.P_1.P_2)\backslash c)$ is chosen, the process behaviour is the same, up to a renaming of process and channel names.

To directly reply to all of the questions: $b.Q_1.Q_2$ can't *output* a message through b towards any other process than $a.P_1.\overline{b}.P_2$, due to the restriction; Q_1 cannot execute twice, since the sum forces us to choose only one of two (non-recursive) processes in which Q_1 does not occur more than once.

- Usually, we simplify the notation by assuming some convention. We do not write "trailing 0", so that $a.0$ is the same as a, and we assume that the operators have decreasing binding power, in the following order: $\backslash\alpha, \alpha., |, +$.

  Explain the meaning of this (slightly modified) quote, and write down the parenthesised version of a couple of terms without parentheses. For instance, is $a + b|c$ the same as $(a + b)|c$, or the same as $a + (b|c)$? Is $A + a.a|b + c$ the same as $A + ((a.a)|(b + c))$, or is it something else entirely?

→ Given an expression with (at least) two operators with different binding power, the operator with stronger power is meant to be executed first (it has precedence).

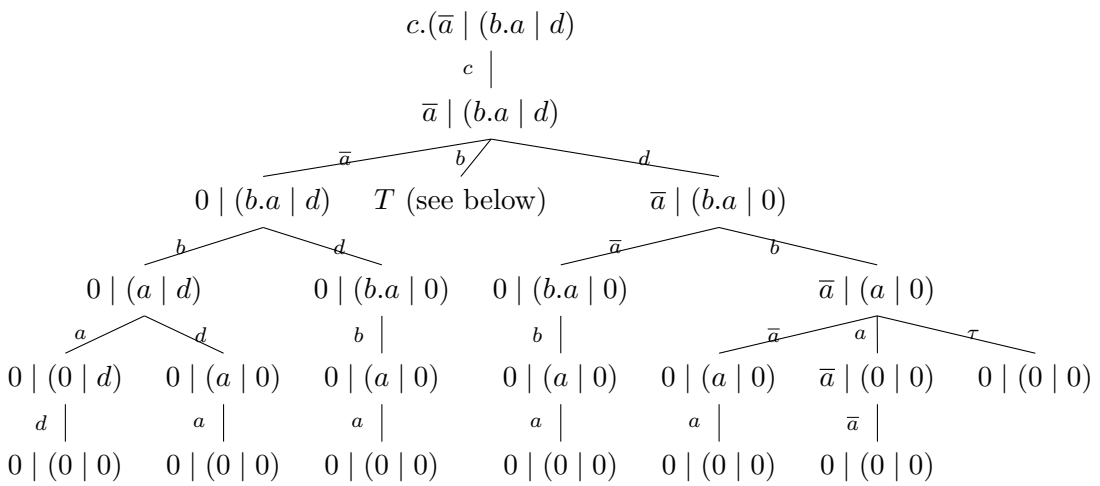
$a + b \mid c$ is the same as $a + (b \mid c)$, as $\mid$ has a greater binding power than $+$.

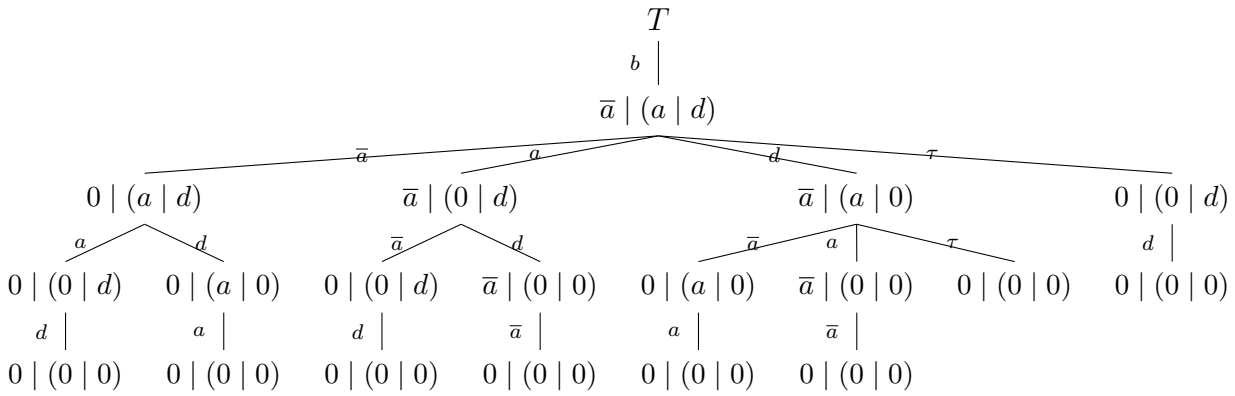
$A + a.a|b + c$ is the same as $(A + ((a.a) \mid b)) + c$, assuming that $+$ is left-associative.

$A \mid B\backslash a + b.C$ is the same as $(A \mid (B\backslash a)) + (b.C)$.

- In forward-only CCS, list all the different reductions that $c.(\overline{a} \mid (b.a \mid d))$ can perform to reach $0|(0|0)$.

→





- Assuming that $a \neq b$, write the derivation of the following transitions.

→

$$\dfrac{\dfrac{\dfrac{}{a.P \xrightarrow{a} P}\ \text{ACT.}}{a.P + b.0 \xrightarrow{a} P}\ +_L}{(a.P + b.0) \mid \overline{a}.Q \xrightarrow{a} P \mid \overline{a}.Q}\ \mid_L$$

$$\dfrac{\dfrac{}{\overline{a}.Q \xrightarrow{\overline{a}} Q}\ \text{ACT.}}{(a.P + b.0) \mid \overline{a}.Q \xrightarrow{\overline{a}} (a.P + b.0) \mid Q}\ \mid_R$$

$$\dfrac{\dfrac{\dfrac{}{b.0 \xrightarrow{b} 0}\,\text{ACT.}}{a.P + b.0 \xrightarrow{b} 0}\,+_R}{(a.P + b.0) \mid a.Q \xrightarrow{b} 0 \mid a.Q}\,|_L \qquad a \neq b\ \dfrac{\dfrac{\dfrac{}{b.0 \xrightarrow{b} 0}\,\text{ACT.}}{a.P + b.0 \xrightarrow{b} 0}\,+_R}{(a.P + b.0) \mid a.Q \xrightarrow{b} 0 \mid a.Q}\,|_L}{((a.P + b.0) \mid a.Q)\backslash a \xrightarrow{b} (0 \mid a.Q)\backslash a}\,\text{RES.}$$

$$\dfrac{a \neq b\ \dfrac{\dfrac{\dfrac{}{a.P \xrightarrow{a} P}\,\text{ACT.}}{a.P \mid Q \xrightarrow{a} P \mid Q}\,|_L}{(a.P \mid Q)\backslash b \xrightarrow{a} (P \mid Q)\backslash b}\,\text{RES.} \qquad \dfrac{}{\overline{a}.Q'\backslash c \xrightarrow{\overline{a}} Q'\backslash c}\,\text{ACT.}}{(a.P \mid Q)\backslash b \mid (\overline{a}.Q'\backslash c) \xrightarrow{\tau} (P \mid Q)\backslash b \mid (Q'\backslash c)}\,\text{SYN.}$$

- Consider the vending machine $V \stackrel{\text{def}}{=} 2\text{p.big.collect}.V + 1\text{p.little.collect}.V$.
  - Modify V so that after inserting 1p you can either buy a little chocolate or insert 1p more and buy a big one.

  → If we want to modify only the behaviour of V after inserting 1p, without modifying its behaviour relative to the 2p insertion:

  $V \stackrel{\text{def}}{=} 2\text{p.big.collect}.V + 1\text{p.}(\text{little.collect}.V + 1\text{p.big.collect}.V)$

  Otherwise, if we read the exercise more strictly and we want V to *only* behave as in the exercise assignment:

  $V \stackrel{\text{def}}{=} 1\text{p.}(\text{little.collect}.V + 1\text{p.big.collect}.V)$

  - Further modify V so that after inserting 2p you can buy either one big chocolate or two little ones.

  → Again, if we read the exercise more loosely, taking the previous vending machine and adding on the requested feature, we have the following machine. Here, I am not sure whether we want the user to collect the two little chocolates with a single "collect" action or with two; I am showing only one option.

  $V \stackrel{\text{def}}{=} 2\text{p.}(\text{big.collect}.V + \text{little.collect.collect}.V) + 1\text{p.}(\text{little.collect}.V + 1\text{p.big.collect}.V)$

  Otherwise, reading it in an exclusive way, we have the following machine:

  $V \stackrel{\text{def}}{=} 2\text{p.}(\text{big.collect}.V + \text{little.collect.collect}.V)$

  - Consider the following vending machine:

  $$V_{\text{bad}} \stackrel{\text{def}}{=} 2\text{p.}(\text{big.collect}.V_{\text{bad}} + \text{little.collect.little.collect}.V_{\text{bad}})$$

  Most people would consider this vending machine as bad: can you explain why?

  → One reason might be the fact that the 1p port, if still present on the vending machine, would be useless. Another reason might be the fact that, in order to obtain two little

chocolates, you have to press the "little" button a second time after taking the first chocolate: if the user inserts 2p and asks the machine to output two little chocolates, it expects the machine to output them both, without the need to confirm his decision in between. Moreover, if the user doesn't know how the machine works and doesn't try to press the "little" button again, the machine gets stuck.

Exercises 2

- For each of the processes below, decide if they are standard, and if they are not, write their set of keys.
  - $X_1 := a[k].b \mid \overline{b}$: not standard. $\text{key}(X_1) = \{k\}$.
  - $X_2 := a \mid b.c$: standard.
  - $X_3 := (b[k].\overline{c} + a) \mid (c + \overline{b})$: not standard. $\text{key}(X_3) = \{k\}$.
  - $X_4 := (a[k].c[k']) \mid (\overline{a}[k].d)$: not standard. $\text{key}(X_4) = \{k, k'\}$.

- Write the term you would obtain in CCSK for each of the transition listed in the last exercise of the previous exercise sheet.

→

$$\dfrac{\text{STD}(b.0) \quad \dfrac{\dfrac{}{a.P \xrightarrow{a[k]} a[k].P}\,\text{ACT.}}{a.P + b.0 \xrightarrow{a[k]} a[k].P + b.0}\,+_L \qquad k \notin \text{KEY}(\overline{a}.Q)}{(a.P + b.0) \mid \overline{a}.Q \xrightarrow{a[k]} (a[k].P + b.0) \mid \overline{a}.Q}\,|_L$$

$$\dfrac{k \notin \text{KEY}(a.P + b.0) \quad \dfrac{\dfrac{}{\overline{a}.Q \xrightarrow{\overline{a}[k]} \overline{a}[k].Q}\,\text{ACT.}}{(a.P + b.0) \mid \overline{a}.Q \xrightarrow{\overline{a}[k]} (a.P + b.0) \mid \overline{a}[k].Q}\,|_R$$

$$\dfrac{\text{STD}(a.P) \quad \dfrac{\dfrac{}{b.0 \xrightarrow{b[k]} b[k].0}\,\text{ACT.}}{a.P + b.0 \xrightarrow{b[k]} a.P + b[k].0}\,+_R \qquad k \notin \text{KEY}(a.Q)}{(a.P + b.0) \mid a.Q \xrightarrow{b[k]} (a.P + b[k].0) \mid a.Q}\,|_L$$

$$\dfrac{a \neq b \quad \dfrac{\text{STD}(a.P) \quad \dfrac{\dfrac{\dfrac{}{b.0 \xrightarrow{b[k]} b[k].0}\,\text{ACT.}}{a.P + b.0 \xrightarrow{b[k]} a.P + b[k].0}\,+_R \qquad k \notin \text{KEY}(a.Q)}{(a.P + b.0) \mid a.Q \xrightarrow{b[k]} (a.P + b[k].0) \mid a.Q}\,|_L}{((a.P + b.0) \mid a.Q)\backslash a \xrightarrow{b[k]} ((a.P + b[k].0) \mid a.Q)\backslash a}\,\text{RES.}$$

$$\dfrac{k \notin \text{KEY}(Q)\ \dfrac{\dfrac{}{a.P \xrightarrow{a[k]} a[k].P}\ \text{ACT.}}{a.P \mid Q \xrightarrow{a[k]} a[k].P \mid Q}\ |_L \quad a \neq b}{(a.P \mid Q)\backslash b \xrightarrow{a[k]} (a[k].P \mid Q)\backslash b}\ \text{RES.} \qquad \dfrac{}{\overline{a}.Q'\backslash c \xrightarrow{\overline{a}[k]} \overline{a}[k].Q'\backslash c}\ \text{ACT.}$$

$$\dfrac{(a.P \mid Q)\backslash b \xrightarrow{a[k]} (a[k].P \mid Q)\backslash b \qquad \overline{a}.Q'\backslash c \xrightarrow{\overline{a}[k]} \overline{a}[k].Q'\backslash c}{(a.P \mid Q)\backslash b \mid (\overline{a}.Q'\backslash c) \xrightarrow{\tau[k]} (a[k].P \mid Q)\backslash b \mid (\overline{a}[k].Q'\backslash c)}\ \text{SYN.}$$

- Write the reverse LTS for CCSK.

→

$$\text{STD}(X)\ \frac{}{\alpha[k].X \overset{\alpha[k]}{\rightsquigarrow} \alpha.X}\ \text{ACT.} \qquad k \neq k'\ \frac{X' \overset{\beta[k]}{\rightsquigarrow} X}{\alpha[k'].X' \overset{\beta[k]}{\rightsquigarrow} \alpha[k'].X}\ \text{PRE.}$$

$$a \notin \{\alpha, \overline{\alpha}\}\ \frac{X' \overset{\alpha[k]}{\rightsquigarrow} X}{X'\backslash a \overset{\alpha[k]}{\rightsquigarrow} X\backslash a}\ \text{RES.}$$

$$k \notin \text{KEY}(Y)\ \frac{X' \overset{\alpha[k]}{\rightsquigarrow} X}{X' \mid Y \overset{\alpha[k]}{\rightsquigarrow} X \mid Y}\ |_L \qquad k \notin \text{KEY}(X)\ \frac{Y' \overset{\alpha[k]}{\rightsquigarrow} Y}{X \mid Y' \overset{\alpha[k]}{\rightsquigarrow} X \mid Y}\ |_R$$

$$\frac{X' \overset{\lambda[k]}{\rightsquigarrow} X \qquad Y' \overset{\overline{\lambda}[k]}{\rightsquigarrow} Y}{X' \mid Y' \overset{\tau[k]}{\rightsquigarrow} X \mid Y}\ \text{SYN.}$$

$$\text{STD}(Y)\ \frac{X' \overset{\alpha[k]}{\rightsquigarrow} X}{X' + Y \overset{\alpha[k]}{\rightsquigarrow} X + Y}\ +_L \qquad \text{STD}(X)\ \frac{Y' \overset{\alpha[k]}{\rightsquigarrow} Y}{X + Y' \overset{\alpha[k]}{\rightsquigarrow} X + Y}\ +_R$$

Figure 1: Reverse LTS semantics for CCSK (without rule for relabeling).

- Write all of the forward and backward transitions that the following processes can perform:
  - $a[k].b|\overline{b}$
  - → *Forward:* $a[k].b \mid \overline{b} \xrightarrow{b[k']} a[k].b[k'] \mid \overline{b}$, for any $k' \neq k$ (CCSK is infinitely branching moving forward)

    $a[k].b \mid \overline{b} \xrightarrow{\overline{b}[k']} a[k].b \mid \overline{b}[k']$, for any $k' \neq k$

    $a[k].b \mid \overline{b} \xrightarrow{\tau[k']} a[k].b[k'] \mid \overline{b}[k']$, for any $k' \neq k$

    Backward: $a[k].b \mid \overline{b} \overset{a[k]}{\rightsquigarrow} a.b \mid \overline{b}$
  - $(b[k].\overline{c} + a) \mid (c + \overline{b})$
  - → *Forward:* $(b[k].\overline{c} + a) \mid (c + \overline{b}) \xrightarrow{\overline{c}[k']} (b[k].\overline{c}[k'] + a) \mid (c + \overline{b})$, for any $k' \neq k$ (from now on, we are going to omit this condition in forward transitions)

$(b[k].\overline{c} + a) \mid (c + \overline{b}) \xrightarrow{c[k']} (b[k].\overline{c} + a) \mid (c[k'] + \overline{b})$

$(b[k].\overline{c} + a) \mid (c + \overline{b}) \xrightarrow{\overline{b}[k']} (b[k].\overline{c} + a) \mid (c + \overline{b}[k'])$

$(b[k].\overline{c} + a) \mid (c + \overline{b}) \xrightarrow{\tau[k']} (b[k].\overline{c}[k'] + a) \mid (c[k'] + \overline{b})$

Backward: $(b[k].\overline{c} + a) \mid (c + \overline{b}) \overset{b[k]}{\rightsquigarrow} (b.\overline{c} + a) \mid (c + \overline{b})$

– $(a[k].c[k']) \mid (\overline{a}[k].d)$

→ *Forward:* $(a[k].c[k']) \mid (\overline{a}[k].d) \xrightarrow{d[k'']} (a[k].c[k']) \mid (\overline{a}[k].d[k''])$
Backward: $(a[k].c[k']) \mid (\overline{a}[k].d) \overset{c[k']}{\rightsquigarrow} (a[k].c) \mid (\overline{a}[k].d)$ (and next, a $\tau[k]$ backward transition is possible. An $\overline{a}[k]$ backward transition is not possible, since k occurs as a key on the left-hand side of the parallel composition.)

- Looking back at the vending machine example from the previous exercise sheet, explain intuitively why $1p.1p.big.collect.V + 1p.little.collect.V$ could be an ok way of answering exercise 4 in a reversible system, and why this process would not fulfill the requirement of exercise 4 in a non-reversible set-up.

  (Reminder of exercise 4: Modify V so that after inserting 1p you can either buy a little chocolate or insert 1p more and buy a big one.)

→ In a non-reversible system, with this vending machine the user has to choose in advance whether he wants a little chocolate or a big one: it does not fulfill the requirement of exercise 4 because the decision cannot be delayed (after inserting 1p, the choice is already done). In a reversible system, if the user performs only forward transitions, the result is the same, as the role of the sum operator is the same as in the non-reversible system: after executing one side of a standard sum process, the other side cannot be executed, because the former side is not standard anymore. However, the user could make a choice, insert 1p, then change his mind, turn back with a backward transition, and then make the opposite decision.

I'm not sure about the answer, which doesn't exactly meet the requirement of exercise 4, but this argument allows the user to make his decision even after inserting the first coin.

Exercises 3

CCS

- Prove that the empty relation is a bisimulation.

→ There is nothing to prove. A binary relation $\mathcal{R}$ is a bisimulation if for all P, Q such that $P \mathcal{R} Q$, then certain properties about P and Q can be proved. If we take $\mathcal{R}$ as the empty relation, we don't have processes P and Q from the start, so the conclusion is trivially true.

- Prove that the identity relation $\mathcal{I}$ ($P \mathcal{I} P$ for all P) is a bisimulation.

→ Let's take a couple (P, P) for some P. If we have a transition $P \xrightarrow{\alpha} P'$, for example starting from the first P occurring in the couple (P, P), then we have identically a transition $P \xrightarrow{\alpha} P'$ starting from the second P in the couple, and $P' \mathcal{I} P'$ holds by definition of $\mathcal{I}$. The symmetric assertion is exactly the same.

- Prove that $a|a$ and $a.a$ are bisimilar.

→ Let's first reason about the transition graph of the two processes. $a|a$ can only make transitions through a into $0|a$ or $a|0$, which both can only make a transition through a into $0|0$. $a.a$ can only transition through a into a, which transitions through a into 0.

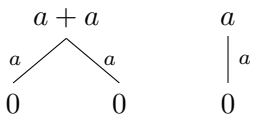
$\mathcal{R} = \{(a|a,\, a.a), (0|a,\, a), (a|0,\, a), (0|0,\, 0)\}$ is the desired bisimulation: given each couple $(P, Q) \in \mathcal{R}$, P and Q only make transitions through a (or don't make transitions) and they end up in a couple of processes belonging to $\mathcal{R}$.

- Prove that $b|a$ and $a.b$ are not bisimilar.

→ There exists a transition $b|a \xrightarrow{b} 0|a$, but there isn't any transition $a.b \xrightarrow{b} P$ for any P.

- Prove that $a + a$ and a are bisimilar.

→ Given the following derivation trees, $\mathcal{R} = \{(a + a, a), (0, 0)\}$ is the desired bisimulation.

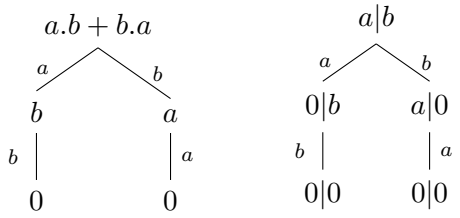


- Prove that $a + b$ and $a|b$ are not bisimilar.

→ If we take the transition $a|b \xrightarrow{a} 0|b$, the process $a + b$ can only simulate it with the transition $a + b \xrightarrow{a} 0$. However, in order for a bisimulation $\mathcal{R}$ to exist, $0|b\,\mathcal{R}\,0$ must hold, but the two processes are not bisimilar. The left process in the couple $(0|b, 0)$ can perform a transition through b, while the right can't.

- Prove that $a.b + b.a$ and $a|b$ are bisimilar.

→ Given the following derivation trees, $\mathcal{R} = \{(a.b + b.a, a|b), (b, 0|b), (a, a|0), (0, 0|0)\}$ is the desired bisimulation.



- Prove that for any P and Q, $P|Q$ and $Q|P$ are bisimilar.

→ Consider $\mathcal{R} = \{(P|Q, Q|P) : P, Q \in \text{Proc}\}$. If we prove that $\mathcal{R}$ is a bisimulation, we obtain that $P|Q$ and $Q|P$ are bisimilar for any P and Q.

Given any couple $(P|Q, Q|P) \in \mathcal{R}$, consider the transitions which the left process can make: they have the shape $P|Q \xrightarrow{\alpha} P'|Q$, $P|Q \xrightarrow{\alpha} P|Q'$ or $P|Q \xrightarrow{\tau} P'|Q'$ (premises are clear and omitted). The right process can respectively make the transition $Q|P \xrightarrow{\alpha} Q|P'$, $Q|P \xrightarrow{\alpha} Q'|P$ or $Q|P \xrightarrow{\tau} Q'|P'$; in each case, the couples $(P'|Q, Q|P')$, $(P|Q', Q'|P)$ and $(P'|Q', Q'|P')$ belong to $\mathcal{R}$. If we consider transitions starting from the right process, we conclude analogously.

- Prove that for any P, $P + 0$ and P are bisimilar.

→ *Disclaimer:* by mistake, I proved that P and $P + 0$ are bisimilar. The proof can be adjusted easily to prove that $P + 0$ and P are bisimilar - or just exploit the fact that bisimilarity is a symmetric relation.

Consider $\mathcal{R} = \mathcal{I} \cup \{(P, P + 0) : P, Q \in \text{Proc}\}$. If we prove that $\mathcal{R}$ is a bisimulation, we obtain that $P|Q$ and $Q|P$ are bisimilar for any P and Q.

Given any couple in $\mathcal{R}$, it has either the form (P, P) or $(P, P + 0)$ for some P. In the first case, the conclusion follows from the fact that $\mathcal{I}$ is a bisimulation itself and it is included in $\mathcal{R}$.

In the second case, first consider any transition $P \xrightarrow{\alpha} P'$ made by the left process: the right process can make the transition $P + 0 \xrightarrow{\alpha} P'$, thanks to the $+_L$ rule, and the couple (P', P') belongs to $\mathcal{R}$. Next, consider transitions starting from the right process: since 0 does not make any transition, they only have the form $P + 0 \xrightarrow{\alpha} P'$, with $P \xrightarrow{\alpha} P'$ as a premise. We already have the needed transition starting from P, and again (P', P') belongs to $\mathcal{R}$.

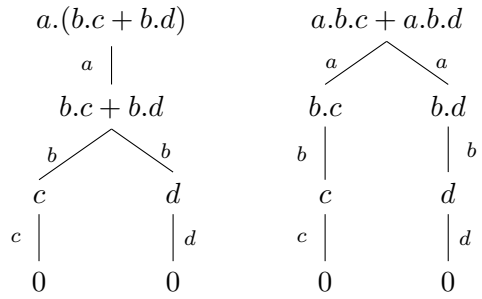
- For each of the following process decide whether they are bisimilar and if not, justify why not:

  - $b.a + b$ and $b.(a + b)$

  → Not bisimilar. The second process can perform two b transitions in a row, while the first process can't.

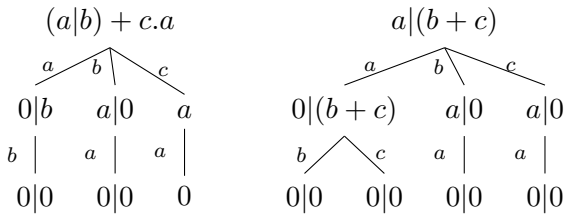
  - $a.(b.c + b.d)$ and $a.b.c + a.b.d$

  → Not bisimilar: the first process simulates the second one, but not vice versa. Given $a.(b.c + b.d) \xrightarrow{a} b.c + b.d$, for the second process we have to choose one of the two sides of the sum (for example, $a.b.c + a.b.d \xrightarrow{a} b.c$). At this point, $b.c + b.d$ can transition through b and next d, but $b.c$ can't.



  - $(a|b) + c.a$ and $a|(b + c)$

  → Not bisimilar. As in the diagrams below, the right process has one transition sequence (a and then c) which the left process can't simulate.



- Last but not least, check whenever $2p.big.collect + 1p.1p.big.collect + 1p.little.collect$ and $2p.big.collect + 1p.(1p.big.collect + little.collect)$ are bisimilar.

→ Not bisimilar: as in a former example, $a.B + a.C$ and $a.(B + C)$ are generally not bisimilar. The second one simulates the first one, but not vice versa.

CCSK

- Prove that $a[k] + b$ and $a[k].b$ are not bisimilar.

→ The process $a[k] + b$ cannot make any forward transition, while the other process can make the transition $a[k].b \xrightarrow{b[k']} a[k].b[k']$.

- Prove that $a[k].(b + c)$ and $a[k].b + c$ are not bisimilar.

→ The former can make a transition $a[k].(b + c) \xrightarrow{c[k']} a[k].(b + c[k'])$, while the latter can't make any transition through $c[k']$.

- Prove that $a[k]|b[k']$ and $a[k].b[k']$ are not bisimilar.

→ The former can make a backward transition through $a[k]$, while the latter can't (it first has to go backwards through $b[k']$).

- Decide whenever $(a.a)|b$ and $(a|a)|b$ are bisimilar.

→ Not bisimilar: the latter can go forward through $a[k]$ and $a[k']$ and go backward through $a[k]$; the former, after going forward through $a[k]$ and $a[k']$, can only go backwards through $a[k']$.

- Decide whenever $a.(b + b)$ and $(a.b) + (a.b)$ are bisimilar.

→ Bisimilar: take $\mathcal{R} = \{(a.(b + b), (a.b) + (a.b)),\ (a[k].(b + b), (a[k].b) + (a.b)),$
$(a[k].(b[k'] + b), (a[k].b[k']) + (a.b)),\ (a[k].(b + b[k']), (a[k].b[k']) + (a.b)),$
$(a[k].(b + b), (a.b) + (a[k].b)),\ (a[k].(b[k'] + b), (a.b) + (a[k].b[k'])),$
$(a[k].(b + b[k']), (a.b) + (a[k].b[k']))\}$.

- Check whenever $2p.big.collect + 1p.1p.big.collect + 1p.little.collect$ and $2p.big.collect + 1p.(1p.big.collect + little.collect)$ (this time, seen as reversible processes) are bisimilar.

→ Not bisimilar: as in forward-only CCS, $a.B + a.C$ and $a.(B + C)$ are generally not bisimilar, for the same reasons as before.

Exercises Bonus

- Write the following terms with as few parenthesis as possible, without changing the meaning or structure of the terms:
  - $(\lambda x.(\lambda y.(\lambda z.((x@z)@(y@z)))))$: $\lambda x.\lambda y.\lambda z.x@z@(y@z)$
  - $(((a@b)@(c@d))@((e@f)@(g@h)))$: $a@b@(c@d)@(e@f@(g@h))$

– $(\lambda x.((\lambda y.(y@x))@(\lambda v.v)@z)@u)@(\lambda w.w)$: $(\lambda x.(\lambda y.y@x)@(\lambda v.v)@z@u)@\lambda w.w$

- Restore all the dropped parentheses in the following terms, without changing the meaning or structure of the terms:

  – $x@x@x@x$: $(((x@x)@x)@x)$

  – $\lambda x.x@\lambda y.y$: $(\lambda x.(x@(\lambda y.y)))$

  – $\lambda x.(x@\lambda y.y@x@x)@x$: $((\lambda x.(x@(\lambda y.((y@x)@x))))@x)$

- Let true and false be $tru = \lambda xy.x$ and $fls = \lambda xy.y$. Write all the possible β-reductions from $fls@fls@(fls@tru)$ to Id.

→ I consider the given term as $(\lambda xy.y)@(\lambda xy.y)@((\lambda xy.y)@(\lambda xy.x))$, and not as $\lambda xy.y@\lambda xy.y@(\lambda xy.y@\lambda xy.x)$: I might be wrong about it. The given term is the same as $[(\lambda xy.y)@(\lambda xy.y)]@[(\lambda xy.y)@(\lambda xy.x)]$, where square brackets are the same as regular brackets (I'm using them just to visualize things more clearly). In order to display the chosen redex at each stage, I have colored the application symbol @ relative to such redex.

Reduction sequences:

– $[(\lambda xy.y)@(\lambda xy.y)]@[(\lambda xy.y)@(\lambda xy.x)] \xrightarrow{\beta} [\lambda y.y]@[(\lambda xy.y)@(\lambda xy.x)] \xrightarrow{\beta} \ldots$
$\ldots \xrightarrow{\beta} [(\lambda xy.y)@(\lambda xy.x)] \xrightarrow{\beta} \lambda y.y$

– $[(\lambda xy.y)@(\lambda xy.y)]@[(\lambda xy.y)@(\lambda xy.x)] \xrightarrow{\beta} [\lambda y.y]@[(\lambda xy.y)@(\lambda xy.x)] \xrightarrow{\beta} \ldots$
$\ldots \xrightarrow{\beta} [\lambda y.y]@[\lambda y.y] \xrightarrow{\beta} \lambda y.y$

– $[(\lambda xy.y)@(\lambda xy.y)]@[(\lambda xy.y)@(\lambda xy.x)] \xrightarrow{\beta} [(\lambda xy.y)@(\lambda xy.y)]@[\lambda y.y] \xrightarrow{\beta} \ldots$
$\ldots \xrightarrow{\beta} [\lambda y.y]@[\lambda y.y] \xrightarrow{\beta} \lambda y.y$