1 Unique decomposition of processes

A process $P$ is $\sim$-prime if $P \sim 0$ and whenever $P \sim P_1|P_2$, then either $P_1 \sim 0$ or $P_2 \sim 0$.

**Question 1.1.** Give an example of a $\sim$-prime process in CCS.

The following theorem, by Moller and Milner, is admitted:

**Theorem.** Any CCS process can be expressed as a parallel composition of prime processes, up to bisimilarity. Moreover, this decomposition is unique.

**Question 1.2.** Prove that the above theorem does not hold for trace equivalence.

2 A finer version of weak bisimilarity

We suppose given a LTS $P \xrightarrow{\mu} P'$. A relation $R$ is a regular bisimulation if $P \mathrel{R} Q$ implies

1. Whenever $P \xrightarrow{\mu} P'$,
   
   - either there are $Q_1, Q_2, Q'$ such that $Q \Rightarrow Q_1, Q_1 \xrightarrow{\mu} Q_2$ and $Q_2 \Rightarrow Q'$, with $P \mathrel{R} Q_1, P' \mathrel{R} Q_2$ and $P' \mathrel{R} Q'$,
   - or $P' \mathrel{R} Q$ and $\mu = \tau$ (note that $\mu = \tau$ is also possible in the case above).

2. Symmetrically when $Q \xrightarrow{\mu} Q'$.

Regular bisimilarity, written $\simeq$, is the largest regular bisimulation. Graphically, the first clause corresponds to the following alternative:

```
       P \xrightarrow{\mu} Q
      / \  \ /
Q_1  Q_2  Q'
```

**Question 2.1.** Prove that $\simeq \subseteq \approx$ (where $\approx$ is weak bisimilarity, as seen in the course).

**Question 2.2.** Define two CCS processes to show that $\approx \neq \simeq$. 

Define \( \simeq' \) as a variant of \( \simeq \), where we replace the first diagram above with the following:

\[
\begin{array}{c}
P \\
\downarrow^\mu \\
\downarrow \\
P'
\end{array}
\begin{array}{c}
\downarrow \\
Q_1 \\
\downarrow \\
Q'
\end{array}
\begin{array}{c}
\downarrow \\
\downarrow \\
\downarrow \\
\downarrow
\end{array}
\]

(we change the definition of regular bisimulation accordingly, and leave the other case unchanged).

**Question 2.3.** Prove that \( \simeq' \) and \( \simeq \) coincide.

Define \( \approx \) as the greatest relation such that the following diagram (and its symmetrical counterpart) is satisfied (for all \( \mu \), note the “hat”):

\[
\begin{array}{c}
P \\
\downarrow^\mu \\
\downarrow \\
P'
\end{array}
\begin{array}{c}
\downarrow \\
Q_1 \\
\downarrow \\
Q'
\end{array}
\begin{array}{c}
\downarrow \hat{\mu} \\
\downarrow \\
\downarrow \\
\downarrow
\end{array}
\]

**Question 2.4.** Prove that \( \approx \) is transitive.

**Question 2.5.** Prove that \( \approx \subseteq \simeq \).

### 3 Finite deterministic automata

We have seen in the course a naive algorithm for checking language equivalence of a pair of states in a finite automaton. Here is the corresponding pseudo-code:

```pseudocode
/* Assume a DFA \((X, o, t)\) over the alphabet \(A\), and two states \(x, y\) in \(X\) */
(1) \(R\) is empty; \(todo\) is empty;
(2) insert \((x, y)\) in \(todo\);
(3) while \(todo\) is not empty do
    (3.1) extract \((x', y')\) from \(todo\);
    (3.2) if \((x', y') \in R\) then continue;
    (3.3) if \(o(x') \neq o(y')\) then return \(false\);
    (3.4) for all \(a \in A\),
        insert \((t_a(x'), t_a(y'))\) in \(todo\);
(3.5) insert \((x', y')\) in \(R\);
(4) return \(true\);
/* return \(true\) if \(L(x) = L(y)\), \(false\) otherwise */
```

Recall that language equivalence of states can be characterised as the greatest bisimulation on the set \(X\) of states. The algorithm is said correct if it answers true only when the input states are equivalent. It is said to be complete if it answers false only if the input states are not language equivalent.

**Question 3.1.** Give an invariant for the while loop that ensures the correctness of the algorithm (some property about the program variables and inputs which always holds just before line (3.1)). Show that this invariant holds when entering the loop for the first time, that it is preserved along iterations, and that it entails correctness.
Question 3.2. Give another loop invariant to prove completeness.

Question 3.3. Give a variant ensuring termination of the algorithm (a measure that strictly decreases along iterations).

We would like to obtain a similar algorithm for language inclusion in finite deterministic automata.

Question 3.4. Characterise language inclusion as the greatest fixpoint of some monotone function.

Question 3.5. What modifications have to be applied to the previous pseudo-code and invariants to obtain an algorithm for language inclusion?

Now we would like to improve this algorithm, using up-to techniques. Recall that Hopcroft and Karp’s algorithm for language equivalence actually uses an up to equivalence technique, which we proved correct in the course using compatibility.

Question 3.6. Propose the best compatible up-to technique you can think of for checking language inclusion (do not invent weird things, stick to what has been seen in the course). Prove its compatibility.

Question 3.7. How to modify the previous pseudo-code, invariants, and proofs to obtain an improved algorithm?

Question 3.8. Give a scenario where the algorithm improved with this up-to technique answers true earlier than the naive one. Is it possible when the sets of states reachable from the starting states x and y are disjoint?

There is a nice data-structure in Hopcroft and Karp’s algorithm to take care of equivalence classes in an efficient way. Unfortunately, we (the professors) are not aware of any good data-structure for exploiting the up-to technique we expect you to define in the previous questions.

Question 3.9 (Open, possibly impossible, optional). Find a data-structure to make the previous algorithm almost linear in the number of states.

4 Coalgebra

Recall that the final coalgebra for the functor $F = 2 \times Id^A$ (such that $FX = 2 \times X^A$) is the coalgebra of formal languages on the alphabet $A$, with derivatives describing the dynamics: $\langle P(A^*), \langle \epsilon, \delta \rangle \rangle$.

Question 4.1. Describe the final coalgebra for the functors $B \times Id^A$ and $B \times Id$, where $B$ is an arbitrary set (justify your answers).

The above functors preserve pullbacks. Now consider the functor $FX = P(A \times X)$ corresponding to LTSs with labels in a fixed set $A$. This functor maps a morphism $f$ between $X$ and $Y$ to a morphism $Ff$ defined by

$Ff : FX \rightarrow FY$

$P \mapsto \{(a, f(x)) \mid \langle a, x \rangle \in P\}$

Question 4.2. Show that $F$ weakly preserve pullbacks, i.e., that it sends pullback squares to weak pullback squares. Give a counter-example to pullback preservation in general. (Hint: decompose $F$ into two smaller functors.)