Communication and Concurrency
Lecture 7

Colin Stirling (cps)

School of Informatics

10th October 2013

Process equivalence: motivation

- "The sequence of actions $a_1 \ldots a_n$ must be carried out cyclically starting with a_1" (the scheduler of Lecture 4)

Process equivalence: motivation

- "The sequence of actions $a_1 \dots a_n$ must be carried out cyclically starting with a_1" (the scheduler of Lecture 4)
- This property cannot be formalised in CTL^-

Process equivalence: motivation

- "The sequence of actions $a_1 \dots a_n$ must be carried out cyclically starting with a_1" (the scheduler of Lecture 4)
- This property cannot be formalised in CTL$^-$
- More natural way of specifying this:
  When all actions but $a_1, \dots, a_n$ are restricted, the system should "behave like" the process P, defined by

$$P \stackrel{\text{def}}{=} \mathtt{a_1}.\mathtt{a_2}.\dots.\mathtt{a_n}.P$$

Process equivalence: motivation

- "The sequence of actions $a_1 \dots a_n$ must be carried out cyclically starting with a_1" (the scheduler of Lecture 4)
- This property cannot be formalised in CTL$^-$
- More natural way of specifying this:
  When all actions but $a_1, \dots, a_n$ are restricted, the system should "behave like" the process P, defined by

$$P \stackrel{\mathrm{def}}{=} \mathtt{a}_1.\mathtt{a}_2.\dots.\mathtt{a}_\mathtt{n}.P$$

- Generally: many systems are informally specified by "behave like" statements.
  Example: when using `telnet` our machine should "behave like" the remote machine (abstracting from delays).

Process equivalence: motivation

- "The sequence of actions $a_1 \dots a_n$ must be carried out cyclically starting with a_1" (the scheduler of Lecture 4)
- This property cannot be formalised in CTL$^-$
- More natural way of specifying this:
  When all actions but $a_1, \dots, a_n$ are restricted, the system should "behave like" the process P, defined by

$$P \stackrel{\text{def}}{=} \mathtt{a_1}.\mathtt{a_2}.\dots.\mathtt{a_n}.P$$

- Generally: many systems are informally specified by "behave like" statements.
  Example: when using `telnet` our machine should "behave like" the remote machine (abstracting from delays).
- But how to formalise "behavioural equivalence"?

Wish list

1. Behavioural equivalence should be an equivalence relation, reflexive, symmetric and transitive.

Wish list

1. Behavioural equivalence should be an equivalence relation, reflexive, symmetric and transitive.
2. Processes that may terminate (deadlock) should not be equivalent to processes that may not terminate (deadlock).

Wish list

1. Behavioural equivalence should be an equivalence relation, reflexive, symmetric and transitive.
2. Processes that may terminate (deadlock) should not be equivalent to processes that may not terminate (deadlock).
3. Congruence: if a component Q of P is replaced by an equivalent component Q' yielding P', then P and P' should also be equivalent.

Wish list

1. Behavioural equivalence should be an equivalence relation, reflexive, symmetric and transitive.
2. Processes that may terminate (deadlock) should not be equivalent to processes that may not terminate (deadlock).
3. Congruence: if a component Q of P is replaced by an equivalent component Q' yielding P', then P and P' should also be equivalent.
4. Two processes should be equivalent iff they satisfy exactly the same properties (such as expressible in modal or temporal logic)

Wish list

1. Behavioural equivalence should be an equivalence relation, reflexive, symmetric and transitive.
2. Processes that may terminate (deadlock) should not be equivalent to processes that may not terminate (deadlock).
3. Congruence: if a component Q of P is replaced by an equivalent component Q' yielding P', then P and P' should also be equivalent.
4. Two processes should be equivalent iff they satisfy exactly the same properties (such as expressible in modal or temporal logic)
5. It should abstract from silent actions.

Wish list

1. Behavioural equivalence should be an equivalence relation, reflexive, symmetric and transitive.
2. Processes that may terminate (deadlock) should not be equivalent to processes that may not terminate (deadlock).
3. Congruence: if a component Q of P is replaced by an equivalent component Q' yielding P', then P and P' should also be equivalent.
4. Two processes should be equivalent iff they satisfy exactly the same properties (such as expressible in modal or temporal logic)
5. It should abstract from silent actions.

We deal first with conditions $1 - 4$

A first candidate: trace equivalence

- A trace of a process E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F.

A first candidate: trace equivalence

- A trace of a process E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F.
- E and F are trace equivalent if they have the same traces.

A first candidate: trace equivalence

- A trace of a process E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F.
- E and F are trace equivalent if they have the same traces.
- This notion satisfies 1 and 3, but not 2.

A first candidate: trace equivalence

- A trace of a process E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F.
- E and F are trace equivalent if they have the same traces.
- This notion satisfies 1 and 3, but not 2.
- Counterexample. $\mathtt{Cl}$, $\mathtt{Cl}'$ trace equivalent

$$
\begin{aligned}
\mathtt{Cl} &\stackrel{\text{def}}{=} \mathtt{tick}.\mathtt{Cl} \\
\mathtt{Cl}' &\stackrel{\text{def}}{=} \mathtt{tick}.\mathtt{Cl}' + \mathtt{tick}.0
\end{aligned}
$$

A second candidate: completed trace equivalence

- A completed trace of E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F that cannot execute any action

A second candidate: completed trace equivalence

- A completed trace of E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F that cannot execute any action
- E and F are completed trace equivalent if they have the same traces and the same completed traces

A second candidate: completed trace equivalence

- A completed trace of E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F that cannot execute any action
- E and F are completed trace equivalent if they have the same traces and the same completed traces
- This notion satisfies 1 and 2, but not 3.

$$
\begin{aligned}
\mathtt{Ven}_1 &\stackrel{\text{def}}{=} \mathtt{1p}.\mathtt{1p}.(\mathtt{tea}.\mathtt{Ven}_1 + \mathtt{coffee}.\mathtt{Ven}_1) \\
\mathtt{Ven}_2 &\stackrel{\text{def}}{=} \mathtt{1p}.(\mathtt{1p}.\mathtt{tea}.\mathtt{Ven}_2 + \mathtt{1p}.\mathtt{coffee}.\mathtt{Ven}_2) \\
\mathtt{Use} &\stackrel{\text{def}}{=} \overline{\mathtt{1p}}.\overline{\mathtt{1p}}.\overline{\mathtt{tea}}.\overline{\mathtt{ok}}.0
\end{aligned}
$$

A second candidate: completed trace equivalence

- A completed trace of E is a sequence w of actions such that $E \xrightarrow{w} F$ for some process F that cannot execute any action
- E and F are completed trace equivalent if they have the same traces and the same completed traces
- This notion satisfies 1 and 2, but not 3.

$$
\begin{aligned}
\mathtt{Ven}_1 &\stackrel{\text{def}}{=} \mathtt{1p}.\mathtt{1p}.(\mathtt{tea}.\mathtt{Ven}_1 + \mathtt{coffee}.\mathtt{Ven}_1) \\
\mathtt{Ven}_2 &\stackrel{\text{def}}{=} \mathtt{1p}.(\mathtt{1p}.\mathtt{tea}.\mathtt{Ven}_2 + \mathtt{1p}.\mathtt{coffee}.\mathtt{Ven}_2) \\
\mathtt{Use} &\stackrel{\text{def}}{=} \overline{\mathtt{1p}}.\overline{\mathtt{1p}}.\overline{\mathtt{tea}}.\overline{\mathtt{ok}}.0
\end{aligned}
$$

- $\mathtt{Ven}_1$ and $\mathtt{Ven}_2$ are completed-trace equivalent, but $(\mathtt{Ven}_1 \mid \mathtt{Use})\backslash K$ and $(\mathtt{Ven}_2 \mid \mathtt{Use})\backslash K$, where $K = \{\mathtt{1p}, \mathtt{tea}, \mathtt{coffee}\}$, are not.