Supplementary Notes on Concurrent Processes

15-312: Foundations of Programming Languages
Frank Pfenning

Lecture 23
November 14, 2002

We have seen in the last lecture that by investigating the reactive behavior of systems, we obtain a very different view of computation. Instead of termination and the values of expressions, it is the interactions with the outside world that are of interest. As an example, we showed an important notion of program equivalence, namely strong bisimulation and contrasted it with observational equivalence of computation with respect to values.

The processes we have considered so far were non-deterministic, but sequential. In this lecture we generalize this to allow for concurrency and also name restriction to obtain a form of abstraction.

In order to model concurrency we allow *process composition*, $P_1 \mid P_2$. Intuitively, this means that processes P_1 and P_2 execute concurrently. Such concurrent processes can interact in a synchronous fashion when one process wants to perform an input action and another process wants to perform a matching output action. As a very simple example, consider two processes A and B plugged together in the following way. A performs input action a and then wants to perform output action $\overline{\mathrm{b}}$, returning to state A. Process B performs an input action b followed by an output action $\overline{\mathrm{c}}$, returning to state B upon completion.

$$\begin{array}{rcl} A & \stackrel{\text{def}}{=} & \mathrm{a}.\overline{\mathrm{b}}.A \\ B & \stackrel{\text{def}}{=} & \mathrm{b}.\overline{\mathrm{c}}.B \end{array}$$

We assume we start with A and B operating concurrently, that is, in state

$$A \mid B$$

Now we can have the following sequence of transitions:

$$A \mid B \xrightarrow{\mathrm{a}} \overline{\mathrm{b}}.A \mid \mathrm{b}.\overline{\mathrm{c}}.B \longrightarrow A \mid \overline{\mathrm{c}}.B \xrightarrow{\overline{\mathrm{c}}} A \mid B$$

We have explicitly unfolded B after the first step to make the interaction between $\overline{\mathrm{b}}$ and b clear. Note that this synchronization is not an external event, so the transition arrow is unadorned. We call this an *internal action* or *silent action* are write τ.

The second generalization from the sequential processes is to permit name hiding (abstraction). In the example above, we plugged processes A and B together, intuitively connecting the output $\overline{\mathrm{b}}$ from A with the input b from B. However, it is still possible to put another process in parallel with A and B that could interact with both of them using b. In order to prohibit such behavior, we can locally bind the name b. We write $\mathsf{new}\, a.P$ for a process with a locally bound name a. Names bound with $\mathsf{new}\, a.P$ are subject to α-conversion (renaming of bound variables) as usual. In the example above, we would write

$$\mathsf{new}\, b.A \mid B.$$

However, we have created a new problem: the name b is bound in this expression, but the scope of b does not include the definitions of A and B. In order to avoid this scope violation we parameterize the process definitions by all names that they use, and apply uses of the process identifier with the appropriate local names. We can think of this as a special form of parameter passing or renaming.

$$\begin{array}{rcl} A(a,b) & \stackrel{\mathrm{def}}{=} & \mathrm{a}.\overline{\mathrm{b}}.A\langle a,b\rangle \\ B(b,c) & \stackrel{\mathrm{def}}{=} & \mathrm{b}.\overline{\mathrm{c}}.B\langle b,c\rangle \end{array}$$

The process expression can now hygienically refer to locally bound names.

$$\mathsf{new}\, b.A\langle a,b\rangle \mid B\langle b,c\rangle$$

This leads to the following language of *concurrent process expressions*.

$$\begin{array}{rrcl} \text{Process Exps} & P & ::= & A\langle a_1,\ldots,a_n\rangle \mid N \mid (P_1 \mid P_2) \mid \mathsf{new}\, a.P \\ \text{Sums} & N & ::= & \alpha.P \mid N_1 + N_2 \mid 0 \\ \text{Action Prefix} & \alpha & ::= & \mathrm{a} \mid \overline{\mathrm{a}} \mid \tau \end{array}$$

In order to describe the possible transitions we use a *structural congruence*, written $P \equiv Q$ that allows us rearrange the pieces of a process expression in a meaning-preserving way. It is given by the following laws, which can be applied anywhere in a process expression. We write FN for the free names in process expression.

$$
\begin{array}{l}
P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P \mid 0 \equiv P \\
M + N \equiv N + M \qquad M + (N + N') \equiv (M + N) + N' \qquad M + 0 \equiv M \\
\mathsf{new}\, a.(P \mid Q) \equiv P \mid (\mathsf{new}\, a.Q) \qquad \text{provided } a \notin \mathrm{FN}(P) \\
\mathsf{new}\, a.P \equiv P \qquad \text{provided } a \notin \mathrm{FN}(P) \\
\mathsf{new}\, a.\mathsf{new}\, b.P \equiv \mathsf{new}\, b.\mathsf{new}\, a.P \\
A\langle b_1, \ldots, b_n \rangle \equiv \{b_1/a_1, \ldots, b_n/a_n\} P_A \qquad \text{provided } A(a_1, \ldots, a_n) \stackrel{\text{def}}{=} P_A
\end{array}
$$

Renaming of variables by new is implicit here. With this definition we can transform any process expression into a standard form

$$\mathsf{new}\, a_1. \ldots \mathsf{new}\, a_n.(M_1 \mid \cdots \mid M_k)$$

where we write 0 if $k = 0$.

In order to define the operational semantics we take advantage of structural congruence to put the expressions that have to interact into proximity. In this semantics, all transitions are silent.

$$
\frac{}{\tau.P + M \longrightarrow P}\ \text{Tau} \qquad \frac{}{(\mathrm{a}.P + M) \mid (\bar{\mathrm{a}}.Q + N) \longrightarrow P \mid Q}\ \text{React}
$$

$$
\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}\ \text{Par} \qquad \frac{P \longrightarrow P'}{\mathsf{new}\, a.P \longrightarrow \mathsf{new}\, a.P'}\ \text{New}
$$

$$
\frac{P \equiv Q \quad Q \longrightarrow Q' \quad Q' \equiv P'}{P \longrightarrow P'}\ \text{Struct}
$$

If we want to examine the interaction of a system with its environment we consider the environment as another *testing process* that is run concurrently with the system whose behavior we wish to examine. As example for the above rules, consider the following process expression.

$$P = (\mathsf{new}\, a.((\mathrm{a}.Q_1 + \mathrm{b}.Q_2) \mid \bar{\mathrm{a}}.0)) \mid (\bar{\mathrm{b}}.R_1 + \bar{\mathrm{a}}.R_2)$$

Note that the output action before R_2 is a different name than a used as the input action to Q_1, the latter being locally quantified. This means there are only two possible transitions.

$$
\begin{array}{l}
P \longrightarrow (\mathsf{new}\, a.(Q_1 \mid 0)) \mid (\bar{\mathrm{b}}.R_1 + \bar{\mathrm{a}}.R_2) \\
P \longrightarrow (\mathsf{new}\, a.(Q_2 \mid \bar{\mathrm{a}}.0)) \mid R_1
\end{array}
$$

We next present an alternative semantics in which we do not need to resort to structural equivalence, except reassociating the terms in a sum. In this semantics an action is made explicit in a transition, but matching input/output actions become silent. We use λ to stand for either a or $\overline{\mathrm{a}}$ and $\overline{\lambda}$ for $\overline{\mathrm{a}}$ or a, respectively.

$$\frac{}{M + \alpha.P + N \xrightarrow{\alpha} P}\ \mathrm{Sum}_t \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \mathrm{React}_t$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{L\text{-}Par}_t \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \mathrm{R\text{-}Par}_t$$

$$\frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin \{\mathrm{a}, \overline{\mathrm{a}}\})}{\mathsf{new}\, a.P \xrightarrow{\alpha} \mathsf{new}\, a.P'}\ \mathrm{Res}_t$$

$$\frac{\{b_1/a_1, \ldots, b_n/a_n\}P_A \xrightarrow{\alpha} P' \quad (A(a_1, \ldots, a_n) \overset{\mathrm{def}}{=} P_A)}{A\langle b_1, \ldots, b_n\rangle \xrightarrow{\alpha} P'}\ \mathrm{Ident}_t$$

As another example of this form of concurrent processes, consider two two-way transducers of identical structure.

$$A(a, a', b, b') \quad \overset{\mathrm{def}}{=} \quad \mathrm{a}.\overline{\mathrm{b}}.A\langle a, a', b, b'\rangle + \mathrm{b}'.\overline{\mathrm{a}'}.A\langle a, a', b, b'\rangle$$

We now compose to instances of this process concurrently, hiding the internal connection between.

$$\mathsf{new}\, b.\mathsf{new}\, b'.(A\langle a, a', b, b'\rangle \mid A\langle b, b', c, c'\rangle)$$

At first one might suspect this is bisimilar with $A\langle a, a', c, c'\rangle$, which shortcircuits the internal synchronization along b and b'. While we have not formally defined bisimilarity in this new setting, this new composition is in fact buggy: it can deadlock when put in parallel with $\overline{\mathrm{a}}.P$, $\mathrm{c}.P'$, $\overline{\mathrm{c}'}.Q$, $\mathrm{a}'.Q'$

$$\begin{array}{l}
\overline{\mathrm{a}}.P \mid \mathrm{c}.P' \mid \overline{\mathrm{c}'}.Q \mid \mathrm{a}'.Q' \mid \mathsf{new}\, b.\mathsf{new}\, b'.(A\langle a, a', b, b'\rangle \mid A\langle b, b', c, c'\rangle) \\
\longrightarrow P \mid \mathrm{c}.P' \mid \overline{\mathrm{c}'}.Q \mid \mathrm{a}'.Q' \mid \mathsf{new}\, b.\mathsf{new}\, b'.(\overline{\mathrm{b}}.A\langle a, a', b, b'\rangle \mid A\langle b, b', c, c'\rangle) \\
\longrightarrow P \mid \mathrm{c}.P' \mid Q \mid \mathrm{a}'.Q' \mid \mathsf{new}\, b.\mathsf{new}\, b'.(\overline{\mathrm{b}}.A\langle a, a', b, b'\rangle \mid \overline{\mathrm{b}'}.A\langle b, b', c, c'\rangle)
\end{array}$$

At this point all interactions are blocked and we have a deadlock. This can not happen with the process $A\langle a, a', c, c'\rangle$. It can evolve in different ways but not deadlock in the manner above; here is an example.

$$
\begin{array}{l}
\overline{\mathrm{a}}.P \mid \mathrm{c}.P' \mid \overline{\mathrm{c}'}.Q \mid \mathrm{a}'.Q' \mid A\langle a, a', c, c'\rangle \\
\longrightarrow P \mid \mathrm{c}.P' \mid \overline{\mathrm{c}'}.Q \mid \mathrm{a}'.Q' \mid \overline{\mathrm{c}}.A\langle a, a', c, c'\rangle \\
\longrightarrow P \mid P' \mid \overline{\mathrm{c}'}.Q \mid \mathrm{a}'.Q' \mid A\langle a, a', c, c'\rangle \\
\longrightarrow P \mid P' \mid Q \mid \mathrm{a}'.Q' \mid \overline{\mathrm{a}'}.A\langle a, a', c, c'\rangle \\
\longrightarrow P \mid P' \mid Q \mid Q' \mid A\langle a, a', c, c'\rangle
\end{array}
$$

The reader should make sure to understand these transition and redesign the composed two-way buffer so that this deadlock situation cannot occur.

The two forms of semantics we have given are equivalent in the following way:

(i) If $P \longrightarrow P'$ then $P \xrightarrow{\tau} P''$ and $P'' \equiv P'$ for some P''.

(ii) If $P \xrightarrow{\tau} P'$ then $P \longrightarrow P'$.

These theorems are proving after appropriate generalization, by inductions over the given derivations. We do not give the proof here.