Formalizing Pi-Calculus in Coq

Richard Zhang

December 19, 2016

1 Introduction

This semester I read about the pi-calculus and notions of equivalence of pi-calculus processes. I then formalized a simplified version of the pi-calculus and some of these equivalence principles in Coq.

2 The Pi-Calculus

2.1 Syntax

The pi-calculus is a process calculus meant to represent aspects of concurrrent systems. It is defined as follows:

$$
\begin{aligned}
P \ ::= \ & x(y).P && \text{(receive on channel x and bind to y in P)} && (1)\\
& \overline{x}(y).P && \text{(send y on channel x and run P)} && (2)\\
& 0 && \text{(process that does nothing)} && (3)\\
& P|Q && \text{(parallel composition of processes)} && (4)\\
& \nu x.P && \text{(create a channel x and then run P)} && (5)\\
& !P && \text{(repeatedly run copies of P)} && (6)
\end{aligned}
$$

Other constructs commonly seen in applications of the pi-calculus include name matching and nondeterministic choice.

The subset of the pi-calculus used in the Coq formalization does not include the restriction operator. This is because the original formalization that included the restriction operator and used bare names didn't take proper account of variable capture, which made the theorems about bisimulation difficult to prove.

The current version of the formalization uses a locally nameless representation. This required removing the restriction operator because defining the dynamics of restrict using a locally nameless representation is difficult - certain steps of the labelled transition semantics with restriction can step a locally closed term to one that is not.

The more serious problem is that in general some rules in the labelled transition semantics may force the reordering of restrictions (since the open rule removes the restriction and the close rule puts it back in place). This requires us (if we're using a locally nameless representation using DeBrujin indices) to define the open rule in a way so that a later close rule is always valid. Intuitively this is a serious problem since we don't know ahead of time what the new index of the variable should be.

2.2 Dynamics

The internal behavior of a pi-calculus process is captured via the reduction semantics, which appears in Figure 1.

At first glance the reduction semantics seems to be missing rules (for example, a rule that lets compositions step in the second argument). This is taken into account by the last rule, which invokes structural congruence. Structural congruence is the smallest congruence that respects the rules defined in Figure 2.

Figure 1: Reduction Semantics

$$\overline{x(z).P|\overline{x}(y).Q \to [z/y]P|Q}$$

$$\frac{P \to P'}{P|Q \to P'|Q}$$

$$\frac{P \to P' \qquad x \notin fv(P)}{\nu x.P \to \nu x.P'}$$

$$\frac{P \to Q \qquad P \equiv P' \qquad Q \equiv Q'}{P' \to Q'}$$

Figure 2: Structural Congruence

$$(P_1|P_2)|P_3 \equiv P_1|(P_2|P_3)$$
$$P_1|P_2 \equiv P_2|P_1$$
$$P|0 \equiv P$$

$$\nu z.\nu w.P \equiv \nu w.\nu z.P$$
$$\nu z.0 \equiv 0$$
$$\nu z.(P_1|P_2) \equiv P_1|\nu z.P_2 \text{ if } z \notin fv(P_1)$$

$$!P \equiv P|!P$$

Figure 3: Labelled Transition Semantics

$$\textsc{Input}\quad \frac{}{x(y).P \xrightarrow{xz} [z/y]P}$$

$$\textsc{Output}\quad \frac{}{\overline{x}(y).P \xrightarrow{\overline{x}y} P}$$

$$\textsc{Compose-L}\quad \frac{P \xrightarrow{\alpha} P' \qquad bv(\alpha) \cap fv(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\textsc{Comm-L}\quad \frac{P \xrightarrow{xy} Q \qquad P' \xrightarrow{\overline{x}y} Q'}{P|P' \xrightarrow{\tau} Q|Q'}$$

$$\textsc{Res}\quad \frac{P \xrightarrow{\alpha} P' \qquad x \notin fv(P)}{\nu x.P \xrightarrow{\alpha} \nu x.P'}$$

$$\textsc{Open}\quad \frac{P \xrightarrow{\overline{x}z} P' \qquad z \neq x}{\nu z.P \xrightarrow{\overline{x}(z)} P'}$$

$$\textsc{Close-L}\quad \frac{P \xrightarrow{\overline{x}(z)} P' \qquad Q \xrightarrow{xz} Q'}{P|Q \xrightarrow{\tau} \nu z.(P'|Q')}$$

$$\textsc{Rep-Act}\quad \frac{P \xrightarrow{\alpha} P' \qquad x \notin fv(P)}{!P \xrightarrow{\alpha} P'|!P}$$

$$\textsc{Rep-Comm}\quad \frac{P \xrightarrow{xz} P' \qquad P \xrightarrow{\overline{x}y} P''}{!P \xrightarrow{\tau} [z/y]P'|P''|!P}$$

$$\textsc{Rep-Close}\quad \frac{P \xrightarrow{\overline{x}(z)} P' \qquad P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} \nu z.(P'|P'')|!P}$$

The reduction semantics is a good account of internal transitions, but we would like to also have a semantics that defines how a process interacts with the environment. This is defined by the labelled transition semantics, in Figure 3. (Note that the rules PAR-L, COMM-L and CLOSE-L have a symmetric version which is not written here, for brevity).

Another reason the labelled transition semantics is convenient is that we are not required to reason up to structural congruence.

Intuitively, τ-transitions in the labelled transition semantics represent internal transitions. Since reductions in the reduction semantics are also meant to represent internal transitions, we want to show that they are equivalent. This equivalence is captured in the following theorem:

Theorem 1 $P \to Q \Leftrightarrow \exists Q', P \xrightarrow{\tau} Q', Q \equiv Q'$.

That is, steps in the reduction semantics are equivalent to tau-transitions in the labelled semantics.

The proof of this theorem relies on four lemmas. First, in the forward direction, we need:

Lemma 1 $P \to P' \iff P \equiv \nu\vec{w}.(\overline{x}(y).R_1|x(z).R_2|S)$ *and* $P' \equiv \nu\vec{w}.(R_1|[z/y]R_2|S)$.

Intuitively, this says that the derivation of the step relation can delay using structural congruence until the final rule. The proof of this theorem is *cong_end* in the Coq formalization.This immediately gives us the theorem.

In the backward direction, we use three lemmas:

Lemma 2 $P \xrightarrow{\overline{x}y} P' \iff P \equiv \nu\vec{w}.(\overline{x}(y).R|S)$ *and* $P' \equiv \nu\vec{w}.(R|S)$ *where* $x, y \notin \vec{w}$.

Lemma 3 $P \xrightarrow{\overline{x}(y)} P' \iff P \equiv \nu y, \vec{w}.(\overline{x}(y).R|S)$ *and* $P' \equiv \nu\vec{w}.(R|S)$ *where* $x \notin y, \vec{w}$.

Lemma 4 $P \xrightarrow{xy} P' \iff P \equiv \nu\vec{w}.(x(y).R|S)$ *and* $P' \equiv \nu\vec{w}.(R|S)$ *where* $x \notin \vec{w}$.

When we have these three lemmas, the proof follows via an induction on the derivation of the LTS. The proofs of the lemmas (ignoring the bound output lemma, since it doesn't arise without restriction) are *lts_input_helper* and *lts_output_helper* in the Coq formalization.

2.3 Bisimulation

To define what it means for two processes to be equivalent, we use the notion of bisimulation. We define it as follows:

A binary relation $\equiv$ is an (early) bisimulation (denoted $\sim_e$) if it is symmetric and for any P and Q, when $P \equiv Q$, $P \xrightarrow{\alpha} P'$, then there exists Q' so that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.

Two processes are (early) bisimilar if they are related by some early bisimulation.

The most obvious way to show that two processes are early bisimilar is to exhibit an early bisimulation that relates them. However, in some cases it can be difficult to explicitly construct this relation. The rest of this section will demonstrate various techniques for proving bisimilarity.

First we introduce the idea of progression.

A relation R *progresses to* a relation S (denoted $\hookrightarrow$) if, whenever $(P, Q) \in R$,

- $P \xrightarrow{\alpha} P'$ implies there exists Q' so that $Q \xrightarrow{\alpha} Q'$ and $(P, Q) \in S$.
- $Q \xrightarrow{\alpha} Q'$ implies there exists P' so that $P \xrightarrow{\alpha} P'$ and $(P, Q) \in S$.

Note that if we have a symmetric relation that progresses to itself, then it is a bisimulation. Note also that if $R \hookrightarrow S$, R is symmetric, and S is a bisimulation, $R \cup S$ is a bisimulation.

A function on relations F is *strongly safe* if $R \subset S$ and $R \hookrightarrow S$ implies $F(R) \subset F(S)$ and $F(R) \hookrightarrow F(S)$.

One key property of strongly safe functions is captures in the following theorem.

Theorem 2 *If F is strongly safe and* $R \hookrightarrow F(R)$*, then R and F(R) are both included in* $\sim_e$.

The proof of this theorem relies on the definition of the following series of relations: let $R_0 = R$, $R_{n+1} = R_n \cup F(R_n)$. We want to show:

Lemma 5 $R_n \hookrightarrow R_{n+1}$.

The proof of this lemma is by induction on n and is *apply_n_progresses* in the Coq formalization. Using this lemma, we can argue that $R_\omega = \bigcup R_n$ is a bisimulation (since if $(P, Q) \in R_\omega$, then $(P, Q) \in R_n$ for some n, and then $R_n \hookrightarrow R_{n+1} \subset R_\omega$).

Since R is included in $\bigcup R_n$, R is included in $\sim_e$.

This gives us another technique for showing $P \sim_e Q$: we need to find a relation R that relates P and Q and a function F so that $R \hookrightarrow F(R)$.

We would like to use this technique to show that the following function is strongly safe:

$$F_{ni}(R) = \{(C[P], C[Q]) | \text{C is a non-input context}, (P, Q) \in R\}.$$

Note that we require the context to be non-input since bisimulation is not respected by substitution.

We also run into another issue: when we step, we may need to consider contexts that have more than one hole. This gives us the idea of considering the transitive closure of $F_{ni}(R)$.

This idea requires us prove the following lemma:

Lemma 6 *Suppose F is such that $R \subset S$ and $R \hookrightarrow S$ implies $F(R) \subset F^*(S)$ and $F(R) \hookrightarrow F^*(S)$. Then F^* is strongly safe.*

We define $F^n(R)$ as:

- $F^0(R) = F(R)$
- $F^{n+1}(R) = F(R)F^n(R)$

Note that $\bigcup F^n(R) = F^*(R)$, where $F^*(R)$ denotes the transitive closure.

This lemma reduces to showing that if $R \subset S$ and $R \hookrightarrow S$, $F^n(R) \subset F^*(S)$ and $F^n(R) \hookrightarrow F^*(S)$. The proof of this lemma is by induction on n and is *iterate_progresses* in the Coq formalization.

Using this theorem, we can show that:

Theorem 3 *F^*_{ni} is strongly safe.*

Using Lemma 6, to prove this we need to show that, if $R \subset S$ and $R \hookrightarrow S$, then $F_{ni}(R) \subset F^*_{ni}(S)$ and $F_{ni}(R) \hookrightarrow F^*_{ni}(S)$.

The inclusion follows directly from the definition of F_{ni}, since we can use the context $[]$.

For the progression, we need to show that if $(P, Q) \in R$, $C[P] \xrightarrow{\alpha} P'$, then there exists Q' so that $C[Q] \xrightarrow{\alpha} Q'$ and $(P', Q') \in F^*_{ni}(S)$. This proof proceeds by induction on C and is *ni_context_progresses* in the Coq formalization.

This gives us the principle of reasoning up to context: R is a bisimulation up to non-input context if $R \hookrightarrow F_{ni}(R)$.

2.4 Other Notions of Equivalence

Nothing in this section is formalized in Coq. This is because the revised input step in the labelled semantics with late binding can take a locally closed term into one that is not. I think it's possible to fix this when reasoning about bisimulation but I haven't completely thought it through.

To define the other notions of equivalence, it's helpful to recast the labelled transition semantics to bind the name later. This can be found in Figure 4. The difference is in the rules INPUT and COMM-L: here we wait until communication happens to substitute, while in the early semantics we substituted immediately when we took an input transition.

Note that the early transition semantics and the late transition semantics are equivalent, in the sense that they have the same τ-transitions.

A binary relation $\equiv$ is a late bisimulation if for any P and Q, when $P \equiv Q$:

Figure 4: Late Labelled Semantics

$$\textsc{Input}\quad \frac{}{x(y).P \xrightarrow{xy} P}$$

$$\textsc{Output}\quad \frac{}{\overline{x}(y).P \xrightarrow{\overline{x}y} P}$$

$$\textsc{Compose-L}\quad \frac{P \xrightarrow{\alpha} P' \qquad bv(\alpha) \cap fv(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\textsc{Comm-L}\quad \frac{P \xrightarrow{xz} Q \qquad P' \xrightarrow{\overline{x}y} Q'}{P|P' \xrightarrow{\tau} [z/y]Q|Q'}$$

$$\textsc{Res}\quad \frac{P \xrightarrow{\alpha} P' \qquad x \notin fv(P)}{\nu x.P \xrightarrow{\alpha} \nu x.P'}$$

$$\textsc{Open}\quad \frac{P \xrightarrow{\overline{x}z} P' \qquad z \neq x}{\nu z.P \xrightarrow{\overline{x}(z)} P'}$$

$$\textsc{Close-L}\quad \frac{P \xrightarrow{\overline{x}(z)} P' \qquad Q \xrightarrow{xz} Q'}{P|Q \xrightarrow{\tau} \nu z.(P'|Q')}$$

$$\textsc{Rep-Act}\quad \frac{P \xrightarrow{\alpha} P' \qquad x \notin fv(P)}{!P \xrightarrow{\alpha} P'|!P}$$

$$\textsc{Rep-Comm}\quad \frac{P \xrightarrow{xz} P' \qquad P \xrightarrow{\overline{x}y} P''}{!P \xrightarrow{\tau} [z/y]P'|P''|!P}$$

$$\textsc{Rep-Close}\quad \frac{P \xrightarrow{\overline{x}(z)} P' \qquad P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} \nu z.(P'|P'')|!P}$$

1. If $P \xrightarrow{\tau} P'$, then there exists Q' so that $Q \xrightarrow{\tau} Q'$ and $P' \equiv Q'$.

2. If $P \xrightarrow{\overline{a}(b)} P'$, then there exists Q' so that $Q \xrightarrow{\overline{a}(b)} Q'$ and $P' \equiv Q'$.

3. If $P \xrightarrow{a(b)} P'$, then there exists Q' so that $Q \xrightarrow{a(b)} Q'$ and for all w, $[w/b]P' \equiv [w/b]Q'$.

4. Symmetric requirements (interchanging P and Q in the above).

Two processes are early bisimilar if they are related by some early bisimulation. We denote this $\sim_e$. Similarly, two processes are late bisimilar if they are related by some late bisimulation. We denote this $\sim_l$.

Note that neither early bisimilarity nor late bisimilarity are fully congruent, as they aren't respected by name substitution. This motivates the definition of open bisimulation.

A binary relation $\equiv$ is an open bisimulation if for all P and Q where $P \equiv Q$ and name substitutions σ, whenever $P\sigma \xrightarrow{\alpha} P'$, there exists $Q\sigma \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$ (and the symmetric requirement, interchanging P and Q). Two processes are open bisimilar if they are related by some open bisimulation, we denote this $\sim_o$.

The main reason open bisimulation is significant is that it is a congruence under all the operations of the pi-calculus.

3 Citations

Benjamin C. Pierce. Foundational Calculi for Programming Languages. In Allen B. Tucker, editor, Handbook of Computer Science and Engineering, chapter 139. CRC Press, 1996.

Robin Milner, Joachim Parrow, David Walker. A calculus of mobile processes. Information and Computation, Volume 100, Issue 1, September 1992, Pages 1-77.

D. Sangiorgi. A theory of bisimulation for the Pi-calculus. Best (Ed.), Proc. CONCUR93, Lecture Notes in Computer Science, vol. 715, Springer, Berlin (1993).

Sangiorgi, Davide, and David Walker. The pi-calculus: a theory of mobile processes. Cambridge: Cambridge U Press, 2003. Print.