The π-Calculus as a Rewrite Theory

CS 524
Lecture 19
J. Meseguer

As with the λ-Calculus, the notation is deceptively simple. That is, it hides considerable complexities in the handling of names and substitution and, furthermore, in two other "dimensions" of the π-calculus:

1. the notion of process equivalence/equality, and
2. the kinds of computation steps allowed

The problem with the notion of process equivalence is how to make it implementable, since at the moment the notions of equivalence proposed by Milner are not so [see pages 7-8 ~~and~~ in Lecture 18]. That is, we would like to have an automatic way of dealing with process equivalences by confluent and terminating equations modulo axioms.

The problem with computation steps is that, in the π-Calculus not all such steps are allowed. The only such steps allowed are specified in Milner's book as follows: [written in our adopted notation]: where the case of an unobservable action τ is omitted; one-step transitions are those that can be inferred from the syn-comm transition by performing it in an allowed context according to the inference system:

synch-comm $\quad N + \underline{out}\ X\langle Y\rangle.P \mid M + \underline{in}\ X[z].Q \to P\{Q\{z \mapsto Y\}\}$

parallel comp. $\dfrac{P \to Q}{P \mid R \to Q \mid R}$ $\qquad$ restriction $\dfrac{P \to Q}{\underline{new}[X]\{P\} \to \underline{new}[X]\{Q\}}$

equiv $\dfrac{P \to Q}{P' \to Q'}$ if $P \equiv P'$ and $Q \equiv Q'$

Of course, when we specify the π-Calculus as a rewrite theory $\mathcal{R}_\pi = (\Sigma_\pi, E_\pi \cup B_\pi, R_\pi)$, what is written above as $P \equiv P'$ will just be the provable equality $P =_{E_\pi \cup B_\pi} P'$ where, hopefully, $\vec{E}_\pi$ will be confluent and terminating modulo axioms B_π, so that process equivalence can be easily implemented. But how can the above <u>restrictions</u> on the <u>contexts</u> in which the <u>synch-comm</u> rule can be applied be <u>modeled</u>?

The key to answering this question is to ask not what the above inference system <u>allows</u>, but what it <u>forbids</u>. What does it forbid?

Obviously, it forbids rewriting <u>under</u> the <u>remaining</u>

operators of the π-calculus, namely:

- $P+Q$
- $\underline{out}\ x\langle y\rangle.\ P$
- $\underline{in}\ x\ [y].\ P$, and
- $!P$

However, when we make substitution explicit, it should also forbid rewriting under substitution application:

- $\sigma\ P$ where σ denotes a substitution.

Q: How is this specified in rewriting logic?

A: By declaring all the above four operators frozen, i.e., by specifying a frozenness map $\phi_\pi : \Sigma_\pi \to \mathcal{P}(\mathbb{N})$, which we can abbreviate as a super-index on the signature thus: $\Sigma_\pi^{\phi_\pi}$, so that our rewrite theory has the form:

$$\mathcal{R}_\pi = \left(\Sigma_\pi^{\phi_\pi},\ E_\pi \cup B_\pi,\ R_\pi\right)$$

So, this takes care of specifying what computation steps are allowed, but we still have left two closely related problems:

1. How to explicitly specify substitution application, and
2. How to make process equivalence easily implementable

As for the λ-calculus, we could deal with substitution application in the standard π-calculus notation; but we have already learned from the λ-calculus experience that this, though possible, is a bad idea. Furthermore, in the π-calculus case with two binding operators is even a worse idea. A second reason not to go this way is the problematic process equivalence equality:

$$P \mid \text{new}[X]\{Q\} = \text{new}[X]\{P \mid Q\} \quad \text{if } X \notin fn(P)$$

which, implicitly, requires applying α-conversion to satisfy the equality's condition. Quite a nightmare to effectively implement.

So, all points in the same direction: Our best chances to specify the π-calculus in an effectively implementable and simple way is to use CINNI!

As shown in the Maude specification in file:

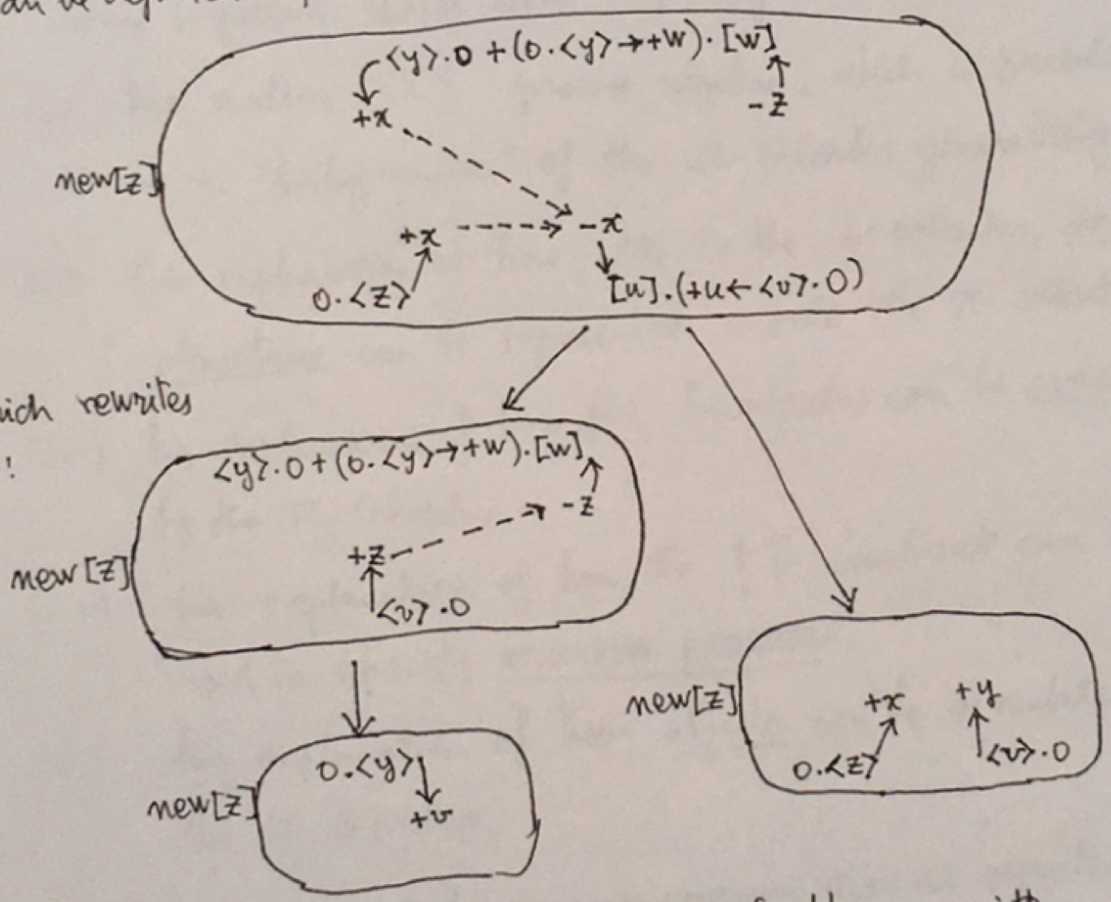
lecture19-A1

we can obtain a full specification of the π-Calculus [leaving aside the inessential unobservable action τ and name complementation $\overline{x}$] as a rewrite theory $\mathcal{R}_\pi$ which is fully executable and yields a notion of process equivalence somewhat less abstract than Milner's, but perfectly reasonable for practical purposes.

Extending the graphical notation to support non-deterministic choice is straightforward. For example, the process:

$$\underline{new}\ [z]\{\{\{(\underline{out}\ x\langle y\rangle.0)+(\underline{in}\ z[w].\underline{out}\ w\langle y\rangle.0)\ |$$
$$\underline{in}\ x[u].\underline{out}\ u\langle v\rangle.0\ |$$
$$\underline{out}\ x\langle z\rangle.0\ \}$$

Can be depicted as follows:



which rewrites to:

where the last two states cannot be further rewritten.

Further Reading

1. The best reference on the π-Calculus is Milner's book:

Robin Milner, "Communicating and Mobile Systems: the π-Calculus," Cambridge U.P., 1999.

Some important issues worth exploring in Milner's book are:

(i) his earlier CCS process algebra, which is presented as a "baby version" of the π-Calculus generalizing it.

(ii) his explanation of how, like in the λ-calculus, data structures can be represented inside the π-calculus

(iii) his explanation of how the λ-calculus can be simulated by the π-Calculus

(iv) his explanation of how the !P construct can be used to specify recursive processes.

(v) his explanation of how objects can be simulated in the π-Calculus.

2. In the π-calculus process communication is synchronous, which has some limitations. However, an asynchronous version exists as well. For a rewriting logic specification of the asynchronous π-Calculus see:

P. Thati, K. Sen, N. Martí-Oliet, "An Executable Specification of Asynchronous Pi-Calculus Semantics and May Testing in Maude 2.0," Electronic Notes in Theoretical Computer Science, 71, 261-281, 2003.

3. A variety of several other π-Calculus-like calculi have been proposed such as:

- L. Cardelli and A. Gordon, "Mobile Ambients", Proc. FOSSACS'98, Springer LNCS 1378, 140-155, 1998
- M. Abadi and A. Gordon, "A Calculus for Cryptographic Protocols: The Spy Calculus," Information and Computation, 148, 1-70, 1999.
- R. De Nicola, G.L. Ferrari, R. Pugliese, "KLAIM: a Kernel Language for Agents Interaction and Mobility," IEEE Trans. Softw. Eng. 24, 315-330, 1998.

4. Using CINNI and I/O Socket objects in Maude, KLAIM has both: (i) (been) given a rewriting logic semantics; and (ii) been implemented in a correct-by-construction way as a distributed system using Maude I/O socket objects.