FAQ on π-Calculus

Jeannette M. Wing
Visiting Researcher, Microsoft Research
Professor of Computer Science, Carnegie Mellon University

27 December 2002

1. What is π-calculus?

π -calculus is a model of computation for concurrent systems.

The syntax of π-calculus lets you represent processes, parallel composition of processes, synchronous communication between processes through channels, creation of fresh channels, replication of processes, and nondeterminism. That's it!

2. What do you mean by process? By channel?

A *process* is an abstraction of an independent thread of control. A *channel* is an abstraction of the communication link between two processes. Processes interact with each other by sending and receiving messages over channels.

3. Could you be a little more concrete?

Let P and Q denote processes. Then

- P | Q denotes a process composed of P and Q running in parallel.
- a(x).P denotes a process that waits to read a value x from the channel a and then, having received it, behaves like P.
- ā⟨x⟩.P denotes a process that first waits to send the value x along the channel a and then, after x has been accepted by some input process, behaves like P.
- (νa)P ensures that a is a fresh channel in P. (Read the Greek letter "nu" as "new.")
- !P denotes an infinite number of copies of P, all running in parallel.
- P + Q denotes a process that behaves like either P or Q.
- **0** denotes the inert process that does nothing.

All concurrent behavior that you can imagine would have to be written in terms of just the above constructs.

4. How about an example? (Impatient readers should feel free to skip this question.)

Suppose you want to model a remote procedure call between a client and a server.

Consider the following function, `incr`, running on the server. `incr` returns the integer one greater than its argument, x:

```
int incr(int x) { return x+1; }
```

First, we model the "incr" server as a process in π-calculus as follows:

$$!\text{incr}(a, x).\bar{a}\langle x+1\rangle$$

Ignoring the ! for now, this process expression says that the incr channel accepts two inputs: one is the name of the channel, a, which we will use to return the result of calling incr, and the other is the argument, x, which will be instantiated with an integer value upon a client call. After the call, the process will send back the result of incrementing its argument, x, on the channel a. The use of the replication operator, !, in the above process expression means that the "incr" server will happily make multiple copies of itself, one for each client interaction.

Now let's model a client call to the "incr" server. In the following assignment statement, the result of calling `incr` with 17 gets bound to the integer variable y:

```
y := incr(17)
```

and would look like this in π-calculus:

$$(\nu a)(\,\overline{\text{incr}}\langle a, 17\rangle \mid a(y)\,)$$

which says in parallel: (1) send on the incr channel both the channel a (for passing back the result value) and the integer value 17, and (2) receive on the channel a the result y. The use of the ν operator guarantees that a private channel of communication is set up for each client interaction with the "incr" server.

Putting the client and server processes in parallel together we get the final process expression:

$$!\text{incr}(a, x).\bar{a}\langle x+1\rangle \mid (\nu a)(\,\overline{\text{incr}}\langle a, 17\rangle \mid a(y)\,)$$

which expresses the client call to the "incr" server with the argument 17 and the assignment of the returned value 18 to y.

5. What is the analogy between π-calculus and λ-calculus?

λ-calculus is to sequential programs as π-calculus is to concurrent programs.

More precisely, λ-calculus is the core language of functional computation, in which "everything is a function" and all computation proceeds by function application; π-calculus is the core calculus of message-based concurrency, in which "everything is a process" and all computation proceeds by communication on channels. λ-calculus can claim to be a *canonical* model of functional computation; however, π-calculus cannot make such a claim for concurrent computation [Pierce95].

Benjamin Pierce puts it best:

> The lambda-calculus holds an enviable position: it is recognized as embodying, in miniature, all of the essential features of functional computation. Moreover, other foundations for functional computation, such as Turing machines, have exactly the same expressive power. The "inevitability" of the lambda-calculus arises from the fact that the only way to observe a functional computation is to watch which output values it yields when presented with different input values.
>
> Unfortunately, the world of concurrent computation is not so orderly. Different notions of what can be observed may be appropriate for different circumstances, giving rise to different definitions of when two concurrent systems have "the same behavior": for example, we may wish to observe or ignore the degree of inherent parallelism of a system, the circumstances under which it may deadlock, the distribution of its processes among physical processors, or its resilience to various kinds of failures. Moreover, concurrent systems can be described in terms of many different constructs for creating processes (fork/wait, cobegin/coend, futures, data parallelism, etc.), exchanging information between them (shared memory, rendezvous, message-passing, dataflow, etc.), and managing their use of shared resources (semaphores, monitors, transactions, etc.).
>
> This variability has given rise to a large class of formal systems called **process calculi** (sometimes **process algebras**), each embodying the essence of a particular concurrent or distributed programming paradigm [Pierce 95].

π-calculus is just one of many such process calculi.

An interesting aside: λ-calculus can be encoded in π-calculus.

6. Why is the term "process algebra" sometimes used? What is a process algebra?

An algebra is a mathematical structure with a set of values and a set of operations on the values. These operations enjoy algebraic properties such as commutativity, associativity, idempotency, and distributivity. In a typical process algebra, processes are values and parallel composition is defined to be a commutative and associative operation on processes.

7. What's the difference between π-calculus and its predecessor process calculi?

What distinguishes π–calculus from earlier process calculi—in particular Robin Milner's own work on Calculus of Communicating Systems (CCS) [Milner80] and Tony Hoare's similar work on Communicating Sequential Processes (CSP) [Hoare85]—is the ability to pass channels as data along other channels. This feature allows you to express process