Communicating Process Architectures 2008
P.H. Welch et al. (Eds.)
IOS Press, 2008

How to Make a Process Invisible

Neil C.C. BROWN
Computing Laboratory, University of Kent
neil@twistedsquare.com

Abstract. Sometimes it is useful to be able to invisibly splice a process into a channel, allowing it to observe (log or present to a GUI) communications on the channel without breaking the synchronous communication semantics. occam-*pi*'s extended rendezvous when reading from a channel made this possible; the invisible process could keep the writer waiting until the real reader had accepted the forwarded communication. This breaks down when it is possible to have choice on outputs (also known as output guards). An extended rendezvous for writing to a channel fixes this aspect but in turn does not support choice on the input. It becomes impossible to keep your process invisible in all circumstances. This talk explains the problem, and proposes a radical new feature that would solve it.

Outline

An occam-π extended rendezvous on a synchronous channel allows a reader to delay a writer (after the data has been communicated) until the reader has also performed an extended action. This can be used to create an invisible process (α in the diagram below), that reads from one channel, c, and writes to another, d, as part of an extended action. The processes writing to channel c (process P) and reading from channel d (process Q) have synchronous communication semantics, as if c and d were the same channel, despite the process α in between (Frigure 1).

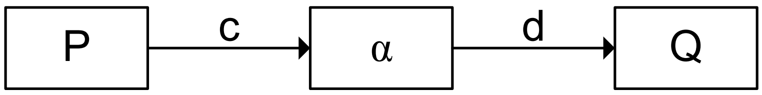

Figure 1. The middle process is not there.

This only holds if choice is available on inputs, but outputs are committed. If P offers an output on c as part of a larger choice, the supposedly-invisible process α will accept the communication immediately and wait for a reader on d during the extended action. Thus P will engage in a communication on c even if Q never reads from d, breaking the synchronous semantics and revealing the presence of the intermediate process α.

An extended output (where a writer performs an extended action after the reader has arrived but before the data is communicated) solves this part of the problem; the invisible process α waits to write on channel d, but then performs an extended action to read from c. But if Q is also involved in a choice, this again breaks the synchronous channel semantics; process P may never write to the channel c, but Q will have chosen to read from d anyway.

We believe that this problem cannot be solved without the introduction of a new feature: the ability to wait for multiple actions. The invisible process α must wait for *both* the reading end of c *and* the writing end of d to be ready, and must perform an extended action on *both* channels to forward the data across. Introducing this ability requires a different implementation and may have far-reaching consequences for how we can construct our process-oriented programs.