Typed Processes in Untyped Contexts

(Abbreviated)

Michele Bugliesi and Marco Giunti
Dipartimento di Informatica, Università Ca’ Foscari di Venezia

1 Background

The use of types to control the behavior of processes in the pi-calculus is a long known and well established technique. The idea was first introduced by Pierce and Sangiorgi in their seminal work on the subject [9], and is best illustrated by their motivating example:

\[ S = (\nu s)!d(s) | !s(x).\overline{\text{print}(x)} \quad C = d(x).\overline{x}(j) \]

\( S \) is a print spooler serving requests from a private channel \( s \) that it communicates to its clients via the public channel \( d \). \( C \) is one such client, that receives \( s \) and uses it to print the job \( j \).

While the intention of the specification is clear, reasoning on its properties is subtler. For instance, given the initial configuration \( S | C \), can we prove that the jobs sent by \( C \) are eventually received and printed? Stated in more formal terms: is there a proof of the following equation?

\[ S | C \equiv S | \overline{\text{print}(j)} \quad (1) \]

Here we take \( P \equiv Q \) to mean that \( P \) and \( Q \) are behaviorally indistinguishable, i.e. they have the same observable behavior when executed in any arbitrary context. Back to our example, (1) is easily disproved by exhibiting a context that interferes with the intended protocol between \( S \) and \( C \). A first example is the context \( C_1[-] = - | d(x) . !x(y) . 0 \), that initially behaves as a client, to receive \( s \), but then steals the jobs intended for \( S \). A second example is the context \( C_2[-] = - | (\nu s')d(s') \), which may succeed in transmitting to \( C \) a dead-ended channel that will never serve the purpose \( C \) expected of it.

As shown in [9], hostile contexts such as those above can be ruled out by resorting to a system of capability types to control the transmission and/or reception of values over channels based on the possession of corresponding type capabilities. In our example, that system allows us to protect against contexts like \( C_1 \) by requiring that clients be only granted write capabilities on the channel \( s \), and by reserving read capabilities on \( s \) to the spooler. Similarly, we may build safeguards against attackers like \( C_2 \) by demanding that clients only have read access on \( d \).

Both the requirements are expressed formally by the typing assumption \( d : ((T)^w)^r \): this typing grants read-only access on \( d \) and write-only access to any name received on \( d \), as desired. We may now refine the equation in (1) into its typed version below (where \( \text{print} : \top \) indicates that

\[ d : ((T)^w)^r \models S | C \equiv S | \overline{\text{print}(j)}. \quad (2) \]
Typed equations of the form $I \vdash P \cong Q$ express behavioral equivalences between processes in any context that typechecks in the type environment $I$. Here $I$ represents the context’s view of the processes under observation, given in terms of a set of typing assumptions on the names shared between the processes and the context itself. Incidentally, but importantly, the typing assumptions on the shared names may in general be different—in fact, more accurate—for the processes than they are for the context.

To illustrate, in (2), a context is only assumed to have read-capabilities on $d$, while for the system to typecheck the name $d$ must be known at the lower (hence more accurate) type $((T)^w)^{rw}$, so that to allow $S$ to write and $C$ to read. Similarly, for the system $S \mid C$ to typecheck, the name $s$ must be known at the type $(T)^{rw}$ including both a write-capability, granted to $S$, and read-capability, granted to any process that receives $s$: the context, instead, will only acquire $s$ at the super-type $(T)^w$ determined by the type of the transmission channel $d$.

# 2 Typed Equivalences Fail in Untyped Contexts

Given the type for $d$ available to the context, it is not difficult to be convinced that (2) above (under appropriate hypotheses on the context’s view of the name print, see Section 4) represents a valid equivalence as no context that typechecks under $d : ((T)^w)^r$ may tell the two processes apart.

Typed equivalences like these are very useful, and effective in all situations in which we have control on the contexts observing our processes, i.e. in all situations in which we may assume that such contexts are well-typed, hence behave according to the invariants enforced by the typing system.

The question we address in this abstract is whether the same kind of reasoning can still be relied upon when our processes are to be deployed in distributed, open environments. Stated more precisely: can we implement our typed processes as low-level agents to be executed in arbitrary, open networks, while at the same time preserving the typed behavioral congruences available for the source processes?

One is readily convinced that no implementation with the desired properties may rely on static typing alone, as distributed and open networks do not validate any useful assumption on the trustworthiness, hence the well-typedness, of the contexts where (the low-level agents representing) our typed processes operate. Rather than assuming that a context satisfies the constraints imposed by a typing assumption, our implementations should enforce them.

The implementation schema we envision here is one in which the statically checked possession and distribution of type capabilities in the source-level processes is realized in terms of the possession and the dynamic distribution of corresponding term-level capabilities in the implementation agents. For instance, each channel could be implemented by means of a pair of cryptographic keys representing the write and read capabilities. If designed carefully, and instrumented with adequate measures to protect against hostile contexts (cf. [1][3][7]) this represents a viable idea to pursue.

The problem remains, however, to make sure that the implementation preserves the desired typed equations of the source calculus: for that to be the case, one must guarantee that for each name, the distribution of the term capabilities in the low-level