A Note on Scope and Infinite Behaviour in CCS-like Calculi*

Pablo Giambiagi[1], Gerardo Schneider[2,3], Frank D. Valencia[2,4]

[1]KTH Royal Institute of Technology, IMIT/LECS, Electrum 229, 164 40 Kista, Sweden
E–Mail: `pgiamb@imit.kth.se`
[2]Uppsala University, Dept. of Computer Systems, Box 337, 751 05 Uppsala, Sweden
E–Mail: {`gerardos`,`frankv`}`@it.uu.se`

1 Introduction

The study of concurrency is often conducted with the aid of process calculi. These calculi treat processes much like the λ-calculus treats computable functions. They provide a language in which the structure of *terms* represents the structure of processes together with an *operational semantics* to represent computational steps. Undoubtedly Milner's CCS [4], a calculus for the modeling and analysis of synchronous communication, remains a standard representative of such calculi.

Nevertheless, there are several variants of CCS in the literature. This is reasonable as a variant may simplify the presentation of the calculus or be tailored to a different set of applications. Given a variant, a legitimate question is then whether some process properties become simpler or harder to deal with, or whether there is loss or gain of *expressive power*. To compare *expressive power* one has to agree on what it means for a variant to be as expressive as the other. A natural way of doing this in CCS is by comparing w.r.t. a standard process equivalence, e.g., *weak bisimilarity*: If for every process P in one variant there is a Q in the other weakly bisimilar to P then we say that the latter variant is as expressive as the former one.

Several studies of variants of CCS and their relative expressive power have been reported in [1]. Also several variants of the π-calculus, itself a generalization of CCS for the study of mobility, have been compared w.r.t. weak-bisimilarity (see [7]). An interesting result is that the π-calculus construction $!P$, whose behavior is expressed by the law $!P \equiv P \parallel !P$, can replace parameterized recursive definitions without loss of expressive power. This is rather surprising since the syntax of $!P$ and its description are so simple. In fact, [2] shows that in CCS such a simulation of recursion with replication is impossible. Another interesting result is that of Palamidessi [6] showing that under some reasonable assumptions the asynchronous version of the π-calculus is strictly less expressive than the synchronous one.

This abstract reports on preliminary results and observations from our study on the relative expressive power of some representative variants of CCS found in the literature. These representatives differ in the way of defining infinite behaviour and name scope. Namely:

- $\mathrm{CCS}_{\mathtt{rec}}$: CCS without relabelling and with infinite behaviour given by *parameterized definitions*.
- $\mathrm{CCS}_{\mathtt{recs}}$: as $\mathrm{CCS}_{\mathtt{rec}}$ but with infinite behaviour given by *parameterless recursion* and *static scoping*.
- $\mathrm{CCS}_{\mathtt{recd}}$: as $\mathrm{CCS}_{\mathtt{recs}}$ but with *dynamic scoping*.
- $\mathrm{CCS}_{\mathtt{rep}}$: CCS with infinite behaviour given by *replication*.

Our preliminary results indicate that w.r.t. weak bisimilarity (I) $\mathrm{CCS}_{\mathtt{rec}}$ and $\mathrm{CCS}_{\mathtt{recd}}$ are equally expressive, and that (II) $\mathrm{CCS}_{\mathtt{recs}}$ and $\mathrm{CCS}_{\mathtt{rep}}$ are equally expressive. This follows from encodings between the corresponding calculi preserving termination and bisimilarity. Now, the termination problem (i.e., whether a given process can diverge) is decidable for a $\mathrm{CCS}_{\mathtt{rep}}$ but undecidable for $\mathrm{CCS}_{\mathtt{rec}}$ [2]. Hence, we would also establish a strong separation result among the calculi follows: Termination is decidable for the calculi in (II) but undecidable for calculi in (I).

We believe that such a strong qualitative distinction between $\mathrm{CCS}_{\mathtt{recd}}$ and $\mathrm{CCS}_{\mathtt{recs}}$ (i.e., static and dynamic scoping) is somewhat surprising since, as we shall see later, their semantics seem to differ very little. Also, the capability of $\mathrm{CCS}_{\mathtt{recd}}$ (i.e., dynamic scoping) to capture the power of parameter-passing in $\mathrm{CCS}_{\mathtt{rec}}$ is also interesting. We also believe that our work may contribute to clarify certain subtle issues in

* In: NWPT'2003 (The 15th Nordic Workshop on Programming Theory). Turku, Finland, October 29-31, 2003.
[3]Work supported by European project ADVANCE, Contract No. IST–1999–29082.
[4]Work supported by European project PROFUNDIS.

the definition of process calculi which may have a strong impact in their expressiveness. E.g., when working with the static scope and parameterless recursion of $\text{CCS}_{\texttt{recs}}$, it is important to be aware of its limitation (i.e., from the above, $\text{CCS}_{\texttt{recs}}$ would not be able to simulate Turing machines.)

In the next section we give a brief description of the various calculi and results previously mentioned.

2 Description of the work

We are interested in classifying some CCS-like calculi that differ in their way of specifying infinite behaviour and scope. Let us begin with the processes that specify finite behaviour only.

Finite Processes. CCS is a formalism of processes that perform actions or synchronize on them. Actions can be either *names* (ranged over by $a, b, \ldots$), *co-names* (ranged over by $\bar{a}, \bar{b}, \ldots$), or the so-called *silent* action τ. Typically, the finite-processes (those that define finite behaviour only) are given by the syntax:

$$P, Q \ldots ::= \sum_{i \in I} \alpha_i.P_i \;\Big|\; P \parallel Q \;\Big|\; P \backslash a \tag{1}$$

Intuitively, $\sum_{i \in I} \alpha_i.P_i$, where I is a finite set of indexes, represents a process able to perform one–but only one–of its α_i's actions and then behave as P_i. We write the summation as **0** if $|I| = 0$, and drop the "$\sum_{i \in I}$" if $|I| = 1$. The process $P \parallel Q$ represents parallel composition; either P or Q may perform an action, or they can also synchronize when one performs an action a and the other performs its complement $\bar{a}$. Finally, $P\backslash a$ behaves as P except that it cannot offer the action a or $\bar{a}$ to its environment (i.e., another process). Another way of describing it is that a is a *private* (or *local*) action only available to P. The names a and $\bar{a}$ in P are said to be *bound* in $P\backslash a$. The *bound names* of P, $bn(P)$ are those with a bound occurrence in P, and the *free names* of P, $fn(P)$ are those with a non-bound occurrence in P. The formal semantic definition of the above processes can be found in [4].

Infinite Processes. In the literature there are at least three alternatives to extend the above syntax for infinite behaviour. We describe these alternatives next.

Parametric Definitions. In this case the syntax in Equation 1 is extended as $P, Q, \ldots := \ldots \;\big|\; A(a_1, \ldots, a_n)$. Each $A(a_1, \ldots, a_n)$ represents a process whose behaviour is given by a unique definition, possibly recursive, $A(b_1, \ldots, b_n) \stackrel{\text{def}}{=} P$ with $fn(P) \subseteq \{b_1, \ldots, b_n\}$. The intuition is that $A(a_1, \ldots, a_n)$ behaves as $P[a_1, \ldots, a_n/b_1, \ldots b_n]$, i.e., as P with each free occurrence of b_i replaced by a_i with the usual renaming of bound names wherever necessary to avoid captures. We call $\text{CCS}_{\texttt{rec}}$ the calculus resulting from this syntax. See [5] for full details.

Replication. In this case Equation 1 is extended as $P, Q, \ldots := \ldots \;\big|!P$. The process $!P$ behaves as $P \parallel P \ldots \parallel P \parallel !P$, i.e, it provides as many copies of P as needed. We call $\text{CCS}_{\texttt{rep}}$ the calculus that results from this syntax (See [5]).

Recursion (or Constant Definitions). In this case Equation 1 is extended as $P, Q, \ldots := \ldots \;\big|\; \mu X.P \;\big|\; X$.We say that $\mu X.P$ binds the occurrences of the *process variable* X in P. The behaviour of $\mu X.P$ is that of $P[\mu X.P/X]$, i.e as P with the free occurrences of X replaced by $\mu X.P$.

A very subtle point of the substitution $[\mu X.P/X]$ in P is whether it *also* involves renaming of *bound names* in P (i.e. α-conversion) to avoid captures. If we require the substitution to perform the necessary name conversions then we say that we have *static name-scoping* and the resulting calculus is $\text{CCS}_{\texttt{recs}}$. Else, if the substitution does not perform any name conversions, we say we have *dynamic name-scoping* and the resulting calculus is $\text{CCS}_{\texttt{recd}}$. The example below justifies the use of the terms "static" and "dynamic."

Example 1. Let $P = (a \,|\, (\overline{a} \parallel X)\backslash a)$. For the static case, the substitution $[\mu X.P/X]$ in P renames the bound a by a fresh name, say b, thus avoiding the capture of $P's$ free variable a in the replacement:

$$P[\mu X.P/X] = (a \,|\, (\bar{b} \parallel \mu X.P)\backslash b)$$

For the dynamic case, however, the substitution makes no name renaming, so it will cause the free occurrence of a in P to get bound, *dynamically*, by the restriction. I.e., we will have

$$P[\mu X.P/X] = (a \mid (\bar{a} \parallel \mu X.P)\backslash a)$$

Our first observation at the semantic level is that, unlike $\mathrm{CCS}_{\mathtt{recd}}$, $\mathrm{CCS}_{\mathtt{recs}}$ allows us to regard as being equivalent processes that differ only by a renaming of bound names (i.e., α-equivalence). This kind of process equivalence is sometimes assumed for CCS-like calculus in the literature. Dynamic scoping for CCS, however, is also used in the literature. E.g. the standard CCS tool, the Concurrency Workbench [3], exhibits the dynamic scoping behaviour illustrated in the above example.

Our second and most important observation is a quantitative distinction between static and dynamic scoping. $\mathrm{CCS}_{\mathtt{recs}}$ is much less expressive than $\mathrm{CCS}_{\mathtt{recd}}$. More precisely, our results indicate that the termination problem is decidable for $\mathrm{CCS}_{\mathtt{recs}}$ while it is undecidable for $\mathrm{CCS}_{\mathtt{recd}}$, which can encode Turing machines. Furthermore, our results also indicate that w.r.t. expressive-power $\mathrm{CCS}_{\mathtt{recd}}$ coincides with $\mathrm{CCS}_{\mathtt{rec}}$ while $\mathrm{CCS}_{\mathtt{rec}}$ coincides with $\mathrm{CCS}_{\mathtt{rep}}$. That is dynamic scoping can encode the full power of parameter-passing present in $\mathrm{CCS}_{\mathtt{rep}}$.

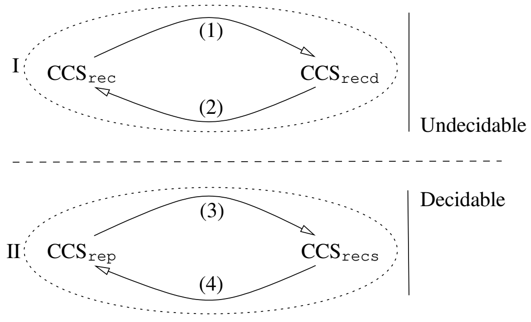

Fig. 1. Classification of CCS-like languages.

In Figure 1 we show the relationship between the different process algebras considered in this work. In group I reachability is *undecidable* while in group II reachability is *decidable*. The arcs between any two of the calculi stand for the *encoding* of one into the other; for instance, the arc labelled by (1) means that $\mathrm{CCS}_{\mathtt{rec}}$ is encoded into $\mathrm{CCS}_{\mathtt{recd}}$. The encodings (2) and (3) are well known but not (1) and (4).

Encoding of $\mathrm{CCS}_{\mathtt{rep}}$ into $\mathrm{CCS}_{\mathtt{recs}}$. We can prove that a slight variation of Milner's encoding of recursion with replication for the π-calculus would work also for this kind of CCS recursion. Decidability of reachability for CCS with replication has been proved in [2].

Encoding of $\mathrm{CCS}_{\mathtt{rec}}$ into $\mathrm{CCS}_{\mathtt{recd}}$. The undecidability of $\mathrm{CCS}_{\mathtt{recd}}$ can be proved directly by encoding two counter machines. But we would like to make it more interesting by proving that the recursion with recursive definition $A(\boldsymbol{x}) \stackrel{\mathrm{def}}{=} P$ can be encoded into recursion with dynamic scoping (no parameters). Our idea of the encoding consists in creating, for each recursive definition $A(x) = P$, an equation A_x and replacing each call $A(y)$ to A in P also by an equation A_y applying α-conversion when necessary, i.e. $A_x = P[A_y/A(y)]$. We define also one equation of the above form for each new variable issue from the α-conversion. There are a finite number of such equations.

3 Discussion

We have presented a classification ofx CCS-like process algebras w.r.t. to the scoping of bounded variables. Although some of the results are known and others are part of the "folklore" of the process algebra community, we believe that the clarification was needed.

References

1. R. Amadio and C. Meyssonnier. On decidability of the control reachability problem in the asynchronous $\pi-$calculus. *Nordic Journal of Computing*, 9(2), 2002.
2. N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. recursive definitions in channel based calculi. In *ICALP'03*, volume 2719 of *LNCS*, pages 133–144. Springer Verlag, 2003.
3. R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, 1993.

4. R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.
5. R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.
6. C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In ACM Press, editor, *POPL'97*, pages 256–265, 1997.
7. D. Sangiorgi and D. Walker. *The $\pi-$calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.