Towards a theory of (self) applicative communicating processes: a short note.

Henk Goeman, Leiden

March 1989

1 Introduction

It is possible to introduce calculi of processes which are direct extensions of the λ-calculus with several notions taken from theories of concurrency (Process Algebra, CCS, etc.).

Such calculi combine notions of abstraction and (self) application taken from the λ-calculus with notions of (non)deterministic choice, concurrent and sequential composition, communication, synchronisation, encapsulation and hiding taken from the theory of concurrent processes.

In a recent paper [1] Gérard Boudol introduced such a λ-calculus for concurrent and communicating processes, where application appears as (a special kind of) communication and where the β-reduction rule appears as (a special kind of) a communicative interaction law.

A source of inspiration for his work was the striking, but of course intentional, similarity of terms of the form $\lambda x.P$ representing the abstraction mechanism in the λ-calculus and terms of the form $\alpha x.P$ representing synchronised input in CCS [2].

In this short note we will introduce an even more direct combination of the λ-calculus with concepts from concurrency, where application is not translated to other (new or existing) primitive constructs, but is itself maintained as a primitive construct in the combined calculus.

We will first propose a possible syntax for such a calculus. Then we will define a possible operational semantics for it by means of a labelled transition system. Finally we will give several examples of process terms which may give some flavor of the expressive power of the calculus.

Some useful comments were given by Frits Vaandrager on an early draft for this paper.

2 Syntax

Let $x, y, z, \ldots$ denote arbitrary values from a set of symbols $V = v_1, v_2, \ldots$, let $\lambda, \mu, \alpha, \beta, \ldots$ denote arbitrary port names from a set of symbols $\Pi = \pi_1, \pi_2, \ldots$, and let s denote an arbitrary port renaming, i.e. an element from $[\Pi \rightarrow \Pi]$. In the following a finite port renaming may be written explicitly as $\alpha_1 \mapsto \alpha'_1, \alpha_2 \mapsto \alpha'_2, \ldots, \alpha_k \mapsto \alpha'_k$.

Now let $P, Q, R, \ldots$ denote arbitrary process terms from the set Γ of process terms inductively defined with the syntax

$$P, Q, R, \ldots \quad ::=$$

$$x \mid (\lambda x.P) \mid (PQ) \mid (\lambda P) \mid (P + Q) \mid (P|Q) \mid (P;Q) \mid (P \backslash \lambda) \mid (P[s]).$$

$(\lambda x.P)$ is called abstraction or input on port λ,
(PQ) is called application,
(λP) is called output on port λ,
$(P + Q)$ is called choice,
$(P|Q)$ is called parallel composition,
$(P;Q)$ is called sequential composition,
$(P \backslash \lambda)$ is called restriction,
$(P[s])$ is called port renaming.

notation

1. outer parentheses are not written;
2. to avoid an excessive use of parentheses the following operator precedences are assumed:
   abstraction $<$ choice $<$ parallel composition $<$ sequential composition $<$ application $<$ output $<$ restriction $<$ renaming;
3. parentheses are also omitted as usual within a repeated abstraction and within a repeated left associative application;
4. the symbol $\equiv$ denotes syntactical equality.

We have chosen for sequential composition as in ACP and for concurrent composition, restriction and port renaming as in CCS. Of course, other possibilities could have been chosen here as well.

3 Semantics

Let a denote an arbitrary action from the set of actions A, where $A = \{\alpha?P,\ \alpha!P,\ \tau \mid \alpha \in \Pi,\ P \in \Gamma\}$. These actions are used as labels in a labelled transition system in $\Gamma \times A \times \Gamma$, generated by the following rules. The rules use also transitions in $\Gamma \times A \times \{\mathsf{nil}\}$.
Please note that nil does not belong to Γ. It should be considered as just a *virtual process*, only used within the transition rules to facilitate the derivations of the real process transitions.

transition rules

1. $\vdash \quad \alpha x.P \xrightarrow{\alpha?Q} P[x := Q]$
2. $P \xrightarrow{\tau} P' \quad \vdash \quad \alpha x.P \xrightarrow{\tau} \alpha x.P'$
3. $P \xrightarrow{\alpha?Q} P' \quad \vdash \quad PQ \xrightarrow{\tau} P'$
4. $P \xrightarrow{\tau} P' \quad \vdash \quad PQ \xrightarrow{\tau} P'Q$
5. $P \xrightarrow{\tau} P' \quad \vdash \quad QP \xrightarrow{\tau} QP'$
6. $\vdash \quad \alpha P \xrightarrow{\alpha!P} \mathsf{nil}$
7. $P \xrightarrow{\tau} P' \quad \vdash \quad \alpha P \xrightarrow{\tau} \alpha P'$
8. $P \xrightarrow{a} \mathsf{nil} \quad \vdash \quad P+Q \xrightarrow{a} \mathsf{nil}$
9. $P \xrightarrow{a} \mathsf{nil} \quad \vdash \quad Q+P \xrightarrow{a} \mathsf{nil}$
10. $P \xrightarrow{a} P' \quad \vdash \quad P+Q \xrightarrow{a} P'$
11. $P \xrightarrow{a} P' \quad \vdash \quad Q+P \xrightarrow{a} P'$
12. $P \xrightarrow{a} \mathsf{nil} \quad \vdash \quad P|Q \xrightarrow{a} Q$
13. $P \xrightarrow{a} \mathsf{nil} \quad \vdash \quad Q|P \xrightarrow{a} Q$
14. $P \xrightarrow{\alpha?R} P',\ Q \xrightarrow{\alpha!R} \mathsf{nil} \quad \vdash \quad P|Q \xrightarrow{\tau} P'$
15. $P \xrightarrow{\alpha?R} P',\ Q \xrightarrow{\alpha!R} \mathsf{nil} \quad \vdash \quad Q|P \xrightarrow{\tau} P'$
16. $P \xrightarrow{\alpha?R} P',\ Q \xrightarrow{\alpha!R} Q' \quad \vdash \quad P|Q \xrightarrow{\tau} P'|Q'$

17. $P \xrightarrow{\alpha?R} P',\ Q \xrightarrow{\alpha!R} Q' \quad \vdash \quad Q|P \xrightarrow{\tau} Q'|P'$

18. $P \xrightarrow{a} P' \quad \vdash \quad P|Q \xrightarrow{a} P'|Q$

19. $P \xrightarrow{a} P' \quad \vdash \quad Q|P \xrightarrow{a} Q|P'$

20. $P \xrightarrow{a} \mathsf{nil} \quad \vdash \quad P;Q \xrightarrow{a} Q$

21. $P \xrightarrow{a} P' \quad \vdash \quad P;Q \xrightarrow{a} P';Q$

22. $P \xrightarrow{\tau} P' \quad \vdash \quad Q;P \xrightarrow{\tau} Q;P'$

23. $P \xrightarrow{\alpha!Q} \mathsf{nil} \quad \vdash \quad P\backslash\beta \xrightarrow{\alpha!Q} \mathsf{nil} \qquad (\alpha \neq \beta)$

24. $P \xrightarrow{\tau} P' \quad \vdash \quad P\backslash\beta \xrightarrow{\tau} P'\backslash\beta$

25. $P \xrightarrow{\alpha?Q} P' \quad \vdash \quad P\backslash\beta \xrightarrow{\alpha?Q} P'\backslash\beta \qquad (\alpha \neq \beta)$

26. $P \xrightarrow{\alpha!Q} P' \quad \vdash \quad P\backslash\beta \xrightarrow{\alpha!Q} P'\backslash\beta \qquad (\alpha \neq \beta)$

27. $P \xrightarrow{\alpha!Q} \mathsf{nil} \quad \vdash \quad P[s] \xrightarrow{s(\alpha)!Q} \mathsf{nil}$

28. $P \xrightarrow{\tau} P' \quad \vdash \quad P[s] \xrightarrow{\tau} P'[s]$

29. $P \xrightarrow{\alpha?Q} P' \quad \vdash \quad P[s] \xrightarrow{s(\alpha)?Q} P'[s]$

30. $P \xrightarrow{\alpha!Q} P' \quad \vdash \quad P[s] \xrightarrow{s(\alpha)!Q} P'[s]$

We will omit here a definition of the substitution construct $P[x := Q]$. This construct should be defined in the usual way where clashes of free and bound occurrences of variables are avoided by means of a suitable renaming of bound variables (α-reduction).

Of course, several rules above may have quite different alternatives, with very natural motivations. It all depends on the desired algebraic properties one may want for the operations on the set of process terms modulo an appropriate observational equivalence, much like the one defined in Boudol's paper [1]. Such an equivalence relation justifies also the equality symbol as used in the next section.

Note especially how the application of a process term P to a process term Q in the proposal above has the effect of making process P behave as a scheduler: it sends process Q to any of its parallel subterms which has evolved to a term with a weak head normal form.

Process application is thus indeed a generalisation of function application!

4 Examples of process terms

1. Let $D \equiv \mu z.\alpha x.\beta(Qx); zz$
   and $O \equiv DD = \alpha x.\beta(Qx); O$.
   The process O represents an object:
   it answers QR on port β for any request R on port α.

2. Let $D \equiv \mu z.(\beta\bot + \alpha x.\beta x); zz$
   and $K \equiv DD = (\beta\bot + \alpha x.\beta x); K$.
   The process K represents a channel with default output $\bot$.

3. Let $D \equiv \mu z.\lambda y.\beta y; zzy + \alpha x.zzx$
   and $R \equiv DD = \lambda y.\beta y; Ry + \alpha x.Rx$
   then $RP = \beta P; RP + \alpha x.Rx$.
   The process RP represents a register with initial content P.
   Note that $\alpha x.Rx$ represents a register without initial content.

4. Let $D \equiv \mu z.\alpha x.\beta x + zz; \beta x$
   and $S \equiv DD = \alpha x.\beta x + S; \beta x$.
   The process S represents a stack to be pushed on port α,
   popped on port β.

5. Let $C \equiv \lambda x.\lambda y.(\,x[\alpha \mapsto \gamma] \mid y[\beta \mapsto \gamma]\,)\backslash\gamma$.
   The process C represents a chaining operator.

Such terms can be used in a very useful way as component process terms in a parallel composition.

Note especially how the usual dichotomy between data objects and program structures is totally absent in our combined calculus: data objects are themselves just component processes.

References

[1] Gérard Boudol, *Towards a Lambda-Calculus for Concurrent and Communicating Systems.*
In: Proc TAPSOFT'89, vol 1: CAAP (J.Díaz,F.Orejas(Eds)), pp 149-161, LNCS 351 (1989).

[2] Robin Milner, *A Calculus of Communicating Systems.*
LNCS 92 (1980).

■