ELSEVIER

Theoretical Computer Science 146 (1995) 341–349

Theoretical Computer Science

Note

# Late and early semantics coincide for testing

Anna Ingólfsdóttir*

*Institute for Electronic Systems, Department of Mathematics and Computer Science, Aalborg University, Aalborg, Denmark*

Received February 1994
Communicated by A.R. Meyer

**Abstract**

Late and early operational semantics are given for CCS. Late and early testing are defined and it is shown that the derived preorders coincide, contrary to what happens for bisimulation-like equivalences.

## 1. Introduction

In the pioneering work of Milner [5] on CCS and Hoare [3] on CSP, processes are allowed to exchange data in communications. In these original calculi the value-passing calculus is interpreted in terms of the pure calculus in which communication is pure synchronization. In CCS a prefixing with an input action, $\bar{a}(x).p$, is interpreted as a nondeterministic choice between pure terms of the form $\bar{a}_v.p[v/x]$, where $v$ ranges over the set of possible values, which in many cases is infinite. In the structural operational semantics given in [5] this is modelled by the infinite set of axioms

$$\forall v \in Val.\bar{c}(x).p \xrightarrow{\bar{c}(v)} p[v/x].$$

Correspondingly, the semantics for output prefixing is given by

$$c(v).q \xrightarrow{c(v)} q$$

and the semantic rule for communication is

$$\frac{p \xrightarrow{\bar{c}(v)} p', \quad q \xrightarrow{c(v)} q'}{p|q \xrightarrow{\tau} p'|q'}.$$

Intuitively, these rules may be interpreted as follows.

* E-mail: annai@iesd.auc.dk.


*SSDI* 0304-3975(94)00304-1

*Input*: The process $\bar{c}(x).p$ tells that it is ready to input any value, $v \in Val$, on channel, $c$. At the same time it tells that by inputting the value $v$ it evolves to the process $p[v/x]$.

*Output*: The process $c(v).q$ proposes to output a value $v$ on channel $c$ and thereby evolve to the process $q$.

*Communication*: The two processes agree on communicating, i.e. sending and receiving the value $v$ on the channel $c$. In doing so they change state by an internal move.

In this approach, two processes that synchronize are both supposed to know each other's *channel* and *value*, i.e. the data variable is instantiated by the potential input values already when the process reports the willingness or ability to communicate on the channel $c$. In more recent work on the $\pi$-calculus [7], this semantic approach is referred to as *early semantics* due to the early instantiation of the data variables as described above. Its counterpart, the *late semantics*, is also introduced in the same reference. Here the idea is that the processes *only* synchronize on the channel name and that the inputting process has to accept whatever value the outputting process has to offer. The result of the reception of the value is delayed until the process has received the value. The inputting process reports the willingness to communicate on a channel $c$ by performing an action of the form $\bar{c}$, and thereby evolves to a function which waits for the value the output counterpart in the communication provides. In order to model this semantic approach the syntax is extended with lambda abstractions on data variables of process expressions to model the inputting processes. Symmetrically, the result of reporting the willingness to output an uninterpreted value on the channel $c$ is modelled by the action $c$. By performing this action the process evolves to a pair consisting of a data expression and a process expression. In [6] these new syntactic constructions have been termed *abstractions* and *concretions*. The rules for input and output actions and that for communication now have the form

*input*: $\bar{c}(x).p \xrightarrow{\bar{c}} \lambda x.p$

*output*: $c(v).q \xrightarrow{c} (v, q)$

*communication*:
$$\frac{p \xrightarrow{\bar{c}} \lambda x.p', \qquad q \xrightarrow{c} (v, q')}{p|q \xrightarrow{\tau} p'[v/x]|q'}$$

This definition naturally leads to an extension of the standard preorders and equivalences defined on labeled transition systems such as those based on the notion of simulation, bisimulation [9], testing [1], etc.

In this note I will try to draw some comparison between the late and early semantics. This comparison will be based on the original value-passing version of Milner's CCS but the syntax has to be extended to cater for the abstractions and concretions of the late approach. The main contribution of the note is to define a late and early testing and show that the derived preorders *coincide*. The motivation for carrying out this proof comes from results by M. Hennessy and myself in 1989 and reported in [2]. In this reference an early semantics and a corresponding testing

equivalence are presented for a CCS-like language with values. Also a denotational model is defined and proved to be fully abstract with respect to the testing semantics. Surprisingly, this model turns out to have some typical late approach characteristics; input actions are modelled by actions of the form $\bar{c}$ and the result of performing the input action by a function which takes a value as an input and delivers an element in the model, i.e. a process. This, combined with the full abstractness result, suggests that the late and early approaches coincide for the testing based semantics. In this note I will give a formal operationally based proof of the conclusion derived from this informal observation.

The structure of the note is as follows: In Section 2, we give the syntax for our example language. The definitions of the operational semantics, both the early and the late one, is the content of Section 3. In Section 4, I give a definition of the late and early testing and show that the derived preorders and equivalences coincide. Finally, in Section 5, I give some concluding remarks.

## 2. Syntax for CCS with values

The example language we use in this note is a suitable sublanguage of the original version of CCS with values introduced in [5]. (As we are mainly interested in communication between processes we omit the renaming and restriction operators.) We refer to this language as *Value* CCS (VCCS). We assume some predefined syntactic category of expressions *Exp* ranged over by $e$, including a set of variable symbols *Var* ranged over by $x$, and a nonempty set of value symbols *Val* ranged over by $v$. We will assume the standard notion of closed expression and substitution. We will also assume a separate syntactic category of boolean expressions *BExp*, ranged over by *be*. Here the set of boolean variables will be denoted by *BVar* and ranged over by $bx$ and the only values are the constants *true* ($T$) and *false* ($F$). Of course, we would expect the language for boolean expressions to use that for expressions in some suitable way.

The set of allowed operators in the language is *NIL* of arity 0 and $+$ and $|$ of arity two. We also need a predefined set of process names *PN* ranged over by upper-case letters such as $P$, $Q$, etc. Instead of the notation $\bar{c}$ from [5] for input on a channel we use $c?$ and to indicate that the channel is used for output we write $c!$. As usual $\tau$ denotes an internal or silent move. We use $be \rightarrow t_1, t_2$ for the conditional choice between $t_1$ and $t_2$. The set of (process) terms, *Term*, is then defined by the following BNF-definition:

$$t ::= NIL \mid t_1 + t_2 \mid t_1 | t_2 \mid P \mid Pre.t_1 \mid recP.t_1 \mid be \rightarrow t_1, t_2$$

$$pre ::= c!e \mid c?x \mid \tau$$

Closed and open process terms and substitution by a process term have the usual meaning. Value-variables may also be bound in process expressions by means of the construct $c?x._-$, giving rise to free and bound value variables. Substitution of value-expression for value variables is also extended to process expressions in the

obvious way. *Proc* is the set of closed process terms, i.e. process terms without occurrence of free data or process variables.

## 3. Early and late operational semantics for VCCS

The language VCCS is given an early operational semantics in the standard way using a labeled transition system, $\langle Proc, Act, \rightarrow_E \rangle$. (The subscript E on the transition relation refers to the early semantic approach.) The set *Act* consists of all input events of the form $c?v$ and all output events of the form $c!v$ where $c \in Chan$ and $v \in Val$. We let $a$ range over *Act* whereas $\mu$ ranges over $Act_\tau = Act \cup \{\tau\}$. The relations $\xrightarrow{\mu}_E$, for $\mu \in Act_\tau$, are defined to be the least relations which satisfy the rules given in Fig. 1. These rules presuppose an evaluation mechanism for closed expressions: $[\![e]\!]$ gives the value in *Val* of the expression $e$ and $[\![be]\!]$ returns either $T$ or $F$. The complementation function $^- : Act \rightarrow Act$ is defined by $\overline{c!v} = c?v$ and $\overline{c?v} = c!v$.

In order to define the late semantics we extend the syntax with the auxiliary syntactic categories *Abs* to denote $\lambda$-abstractions of processes over values and *Con* to denote the output concretions, i.e. pairs consisting of a value and a process. Thus, we add to the syntax the constructions

$$abs ::= \lambda x.t$$

$$con ::= (e, t)$$

1. $c?x.p \xrightarrow{c?v}_E p[v/x]$ for any $v \in Val$
   $c!e.p \xrightarrow{c![\![e]\!]}_E p$
   $\tau.p \xrightarrow{\tau}_E p$

2. $p \xrightarrow{\mu}_E p'$ implies $p + q \xrightarrow{\mu}_E p'$
   $q + p \xrightarrow{\mu}_E p'$

3. $p \xrightarrow{\mu}_E p'$ implies $p \mid q \xrightarrow{\mu}_E p' \mid q$
   $q \mid p \xrightarrow{\mu}_E q \mid p'$

4. $[\![be]\!] = T, p \xrightarrow{\mu}_E p'$ implies $(be \rightarrow p, q) \xrightarrow{\mu}_E p'$
   $[\![be]\!] = F, q \xrightarrow{\mu}_E q'$ implies $(be \rightarrow p, q) \xrightarrow{\mu}_E q'$

5. $$\frac{t[recP.t/P] \xrightarrow{\mu}_E q}{recP.t \xrightarrow{\mu}_E q}$$

6. $p \xrightarrow{a}_E p', q \xrightarrow{\bar{a}}_E q'$ implies $p \mid q \xrightarrow{\tau}_E p' \mid q'$

Fig. 1. Early operational semantics for VCCS.

We call the extended language *Late Value* CCS (VCCS$_L$). We let $t, u$, etc., range over *Term*, $abs_1$, $abs_2$, etc., over *Abs*, $con_1$, $con_2$, etc., over *Con* and $\phi$, $\psi$, etc., over $Abs \cup Con \cup Term$.

As before we give the operational semantics by means of a transition relation. In this new approach the visible actions are of the form $c?$ indicating that the process is waiting for an unspecified input on the channel $c$ and $c!$ indicating that the process is offering a value on the channel $c$. By performing these actions in the first case the process moves to an abstraction which is ready to receive a value and return a process and in the second case to a concretion, a pair consisting of the value offered and the resulting process. If we call this new transition relation $\rightarrow_L$, this may be described formally by defining

$$c?x.t \xrightarrow{c?}_L \lambda x.t$$

$$c!e.p \xrightarrow{c!}_L (v, p) \quad \text{where } v = [\![e]\!]$$

The communication rule now takes the form

$$\frac{p \xrightarrow{c?}_L abs, \qquad q \xrightarrow{c!}_L (v, q')}{p|q \xrightarrow{\tau}_L abs(v)|q'}$$

where the function application is defined in the standard way by $(\lambda x.t)(v) = t[v/x]$. All the other transition rules remain unchanged. Here it should be pointed out that the transition relation is no more between processes but involves other types. Therefore, we are not able to model the operational semantics by means of the usual LTS but use instead *applicative* LTS (ALTS), which is defined in [4].

## 4. Early and late testing

In this subsection we apply the general theory of testing from [1] to VCCS. A *test* $e$ is any process from VCCS which may use in addition to the channels in *Chan* a special channel $w$ for reporting success. A *computation* is a maximal sequence of the form

$$p|e = p_0|e_0 \xrightarrow{\tau}_E p_1|e_1 \cdots p_k|e_k \xrightarrow{\tau}_E \cdots.$$

This computation is said to be *successful* if there exists some $n \geqslant 0$ such that $e_n \xrightarrow{w!v}$ for some value $v$. We say that $p$ *may* $e$ if there is a successful computation which starts in $p|e$ and that $p$ *must* $e$ if every computation starting in $p|e$ is successful. Then $p \sqsubseteq_{may} q$ if for every test $e$,

$$p \text{ may } e \text{ implies } q \text{ may } e$$

and $p \sqsubseteq_{must} q$ if for every test $e$,

$$p \text{ must } e \text{ implies } q \text{ must } e.$$

The derived equivalences are defined in the usual way: $\overline{\approx}_M = \underline{\lesssim}_M \cap \underline{\lesssim}_M^{-1}$, $M \in \{may, must\}$. The reader is referred to [1] for the motivation of these definitions.

This definition of testing also applies for the late semantics. Now the computations are with respect to the late operational semantics and have the form

$$p \mid e \xrightarrow{\tau}_{\mathrm{L}} p_1 \mid e_1 \xrightarrow{\tau}_{\mathrm{L}} \cdots.$$

The tests report success by $e \xrightarrow{w!}$. The derived preorders, $\underline{\lesssim}_{may}$ and $\underline{\lesssim}_{must}$, are defined in the same way as for the early case. Remarkably, unlike for bisimulation, it turns out that the late testing preorders coincide with the early ones as stated in the following theorem.

**Theorem 4.1.** *Late and early testing semantics coincide, i.e. For all $p, q \in VCCS$,*

$$p \underline{\lesssim}_M q \text{ if and only if } p \underline{\lesssim}_M q$$

*for $M = may, must$.*

Theorem 4.1 follows directly from the following lemma which is proved for a richer language, the $\pi$-calculus, in [8].

**Lemma 4.2.**

(i) *For all $p, q$ and $c$*

1. $p \xrightarrow{c?}_{\mathrm{L}} abs$ *implies* $p \xrightarrow{c?v}_{\mathrm{E}} abs(v)$ *for all* $v \in Val$,
2. $p \xrightarrow{c?v}_{\mathrm{E}} p_v$ *implies* $p_v = abs(v)$ *where* $p \xrightarrow{c?}_{\mathrm{L}} abs$.

(ii) *For all $p, q, c$ and $v$, $p \xrightarrow{c!}_{\mathrm{L}} (v, q)$ if and only if $p \xrightarrow{c!v}_{\mathrm{E}} q$.*

(iii) *For all $p$ and $q$, $p \xrightarrow{\tau}_{\mathrm{E}} q$ if and only if $p \xrightarrow{\tau}_{\mathrm{L}} q$.*

**Proof.** May be proved by a simple induction on the length of the derivation tree for the transitions. □

**Proof of Theorem 4.1.** By Lemma 4.2 (iii), we have that

$$e \mid p \xrightarrow{\tau}_{\mathrm{L}} e_1 \mid p_1 \xrightarrow{\tau}_{\mathrm{L}} \cdots$$

if and only if

$$e \mid p \xrightarrow{\tau}_{\mathrm{E}} e_1 \mid p_1 \xrightarrow{\tau}_{\mathrm{E}} \cdots.$$

Further, by Lemma 4.2(ii), $e_n \xrightarrow{w!v}_{\mathrm{E}}$ for some $v$ if and only if $e_n \xrightarrow{w!}_{\mathrm{L}}$. The theorem follows easily from this observation. □

We will end this section by giving an example to compare the testing equivalence to those based on bisimulation.

**Example 4.3.** In this example we use the convention $\mu$ for $\mu.NIL$. Let $c, c_1$ and $c_2$ be channel names where $c_1 \neq c_2$. Furthermore, let the processes $p, p_1, p_2$ and $q, q_1, q_2$ be

defined by

$$p_1 = c?x.x \leqslant 5 \rightarrow c_1!x, c_2!x,$$

$$p_2 = c?x.x \leqslant 5 \rightarrow c_2!x, c_1!x,$$

$$q_1 = c?x.c_1!x, \qquad q_2 = c?x.c_2!x,$$

$$p = p_1 + p_2, \qquad q = q_1 + q_2.$$

Obviously, $p \sim_E q$ as they have isomorphic transition graphs with respect to the early approach. It is also easy to see that $p \not\sim_L q$.

Now assume that $e$ is a test and consider the computations starting in $e|p$ on one hand and $e|q$ on the other. As the late and the early computations are completely identical, we may assume that we are dealing with the early computations. As $p$ and $q$ have isomorphic transition graphs then $e|p$ and $e|q$ must have isomorphic $\tau$-transition graphs too. Further, the testing parts, $e_n$ and $e'_n$ of two isomorphic states, $e_n|p_n$ and $e'_n|q_n$, are syntactically identical. This proves that $p \simeq_M q$ for both $M = may$ and $M = must$.

Example 4.3 suggests that the testing semantics reflects the early approach rather than the late one. On the other hand, as pointed out in the Introduction, the result in [2] suggests the opposite; the input prefixing is interpreted as a function from values to processes, prefixed with elements of the form $c?$. In fact, this is not a contradiction at all as I will explain in the Conclusion.

## 5. Conclusion

The main contribution of this note is to define late and early testing and show that the derived equivalences coincide. I will finish this note by giving a few comments on this result.

The comparison of the late and the early bisimulation in [7] is based on a concrete example language. This result can easily be extended to a more general class of languages. Obviously, the late bisimulation equivalence is stronger than the early one for any language. Furthermore, the distinguishing power of Example 4.3 does not depend on the exact syntax of the language but is only due to the interplay between the nondeterministic choice between inputting processes and the ability to test and branch on inputted values. Therefore, we may conclude that for any language which has the two mentioned facilities, i.e. is able to mix nondeterminism between inputting processes and testing and branching on inputted values, the late bisimulation equivalnece will be strictly finer than the early one. On the other hand, if the language does not have this property, the late and early bisimulation coincide. An example of a language which does not have nondeterministic choice between outputting processes is the broadcasting calculus CBS [10]; for this language the late and early bisimulation semantics coincide.

Also for the testing semantics some general conclusions can be drawn. Theorem 4.1 shows that the late and early semantics coincide for testing. The proof of the theorem does not depend on the concrete language but on the fact that the $\tau$-moves coincide for both approaches and the fact that $p \xrightarrow{c!v}_{\mathrm{E}}$ for some value $v$ if and only if $p \xrightarrow{c!}_{\mathrm{L}}$. Here branching by conditional choice and nondeterminism do not seem to have the same influence as for bisimulation.

As already pointed out, then Example 4.3 suggests that the testing semantics reflects the early approach rather than the late one. On the other hand, the result in [2] supports an opposite conclusion; the process $c?x.p$ is interpreted as $c?_A.\lambda v.A[\![p[v/x]]\!]$, where $A$ refers to the model, i.e. as a prefixing by $c?_A$ of the function $\lambda v.A[\![p[v/x]]\!]$. At first glance this looks like a contradiction but turns out to have a natural explanation; testing eliminates the nondeterministic choice between processes which output on the same channel or more precisely pushes the choice under the prefixing so it becomes a nondeterministic choice between the results of the inputting actions. In [2] this is reflected by the sound equation

$$c?x.p + c?x.q = c?x.(p \oplus q),$$

where $\oplus$ denotes an internal choice between $p$ and $q$. As already argued for, then without this kind of nondeterminism there is no difference between the late and early approach and furthermore both approaches may be modelled by using functions.

Now one may ask for an intuitive explanation of the different results of comparing late and early semantics for bisimulation and testing as described above. Both bisimulation and testing semantics are observational semantics based on the idea of testing the processes by interacting with them and observe their reactions; two processes are identified if they show exactly the same behavioural pattern seen from an external observer. The main difference is the type of observer chosen. In the case of testing the observer is another process. This implies that if the process which is to be tested has two identical "buttons" like, for instance, for $c?x.p + c?x.q$ then the observer has no way of distinguishing between these buttons. Another characteristic of testing by another process is that when the two processes communicate, the observing process has never access to the intermediate states due to the late semantics, in particular the abstractions. Thus, the observer does not have any chance of comparing pointwise two different abstractions due to two identical input actions.

In the case of bisimulation this is different. Here the observer can choose between the identical buttons in a deterministic way and compare the resulting abstractions after having pressed first the "first" one then going back to the original state and then pressing the "second" one.

By using the same kind of reasoning one may conclude that for all "bisimulation-like" equivalences, i.e. equivalences which are based on a direct comparison of the abstractions, and languages with conditional branching on input values and nondeterminism between inputting processes, the late version is strictly stronger than the early one. As examples of bisimulation-like equivalences we can mention equivalences based on simulation, 2/3-bisimulation [11] branching bisimulation [12], etc.

## References

[1] M. Hennessy, *Algebraic Theory of Processes* (MIT Press, Cambridge, MA, 1988).
[2] M. Hennessy, and A. Ingólfsdóttir, A theory of communicating processes with value passing, *Inform. and Comput.* **107** (1993) 202–236.
[3] C.A.R. Hoare, Communicating Sequential Processes, *Comm. ACM* **21** (1978) 666–677.
[4] A. Ingólfsdóttir, Semantic models for communicating process with value-passing, Ph.D. Thesis, Tech. Report 8/94, Sussex University, 1994.
[5] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, Vol. 92 (Springer, Berlin, 1980).
[6] R. Milner, Polyadic $\pi$-calculus. A tutorial, Tech. Report, CS-LFCS-91-180, LFCS, Department of Computer Science, University of Edinburgh, 1991; *Proc. Internat. Summer School on Logic and Algebra of Specification*, Marktoberdorf, August 1991, to appear.
[7] R. Milner, J. Parrow and D. Walker, *Modal Logics for Mobile Processes, Proc. Concur '91*, Lecture Notes in Computer Science, Vol. 327 (Springer, Berlin, 1991) 45–60.
[8] R. Milner, J. Parrow and D. Walker, A calculus of mobile processes, Part I + 2, *Inform. and Comput.* **100** (1992) 1–77.
[9] D. Park, *Concurrency and Automata on Infinite Sequences*, Theoretical Computer Science VII, Lecture Notes in Computer Science, Vol. 104 (Springer, Berlin, 1981).
[10] K.V.S. Prasad, Programming with broadcasts, *Proc. Concur '93*, Lecture Notes in Computer Science, (Springer, Berlin, 1993).
[11] A. Skou, Validation of concurrent processes with emphasis on testing, Ph.D. Thesis,. Aalborg University, 1989.
[12] R.J. van Glabbeek and W.P. Weijland, Branching time and abstraction in bisimulation semantics (extenced abstract), in: G.X. Ritter, ed., *Information Processing 89* (North-Holland, Amsterdam, 1989) 613–618.