CCS without τ's

Rocco De Nicola and Matthew Hennessy*[+]
* Istituto di Elaborazione dell'Informazione, CNR - Pisa
[+]Computer Science Division, University of Sussex - Brighton

Abstract

The main point of this paper is that one can develop an adequate version of CCS which does not use the special combinator τ for internal actions. Instead, the choice operator +, whose semantics is somewhat unclear, is replaced by two new choice operators ⊕ and [], representing internal and external nondeterminism respectively. The operational semantics of the resulting language is simpler and the definition of testing preorders is significantly cleaner. The essential features of the original calculus are kept; this is shown by defining a translation from CCS to the new language which preserves testing preorders.

1. Introduction

In[Mil80], Milner introduced a calculus of communicating systems which is usually referred to as CCS. It consists of a language for defining communicating systems or processes, a semantic theory for these processes and a calculus for syntactically deriving semantic equivalences. The language is algebraic in nature; it consists of recursive definitions which use a small set of combinators. Each recursive definition or term in the language represents a process and each individual combinator represents an intuitive method for composing existing processes to form new ones. Since its publication CCS has been the focus of a considerable amount of research activity, mainly in the definition of alternative algebraic languages, [AB84], [BK84], [Miln85], [ISO86] and in the development of alternative semantic theories, [BHR84], [DH84], [dBZ82]. In fact these two activities are not unrelated: often the success and elegance of a tractable semantic theory depends on the syntax of the language to which it is applied. The purpose of the present paper is to show that by changing the syntax of CCS slightly, but retaining all of its essential features, we can obtain a much simpler semantic theory than both that originally presented in [Mil80] or in papers such as [DH84].

We start with a review of the language CCS and restrict our attention to the so-called pure version. The most complicated combinator is parallel composition |: p | q represents a process which has two subprocesses running in parallel, p and q. To explain this construction we need to understand some conventions which are used in CCS. Communication takes place via ports, such ports can be barred, like β^{-}, or unbarred like β. These pairs of ports are said to be *complementary* and processes communicate via

complementary ports: a communication or synchronization is taken to be the simultaneous occurrence of two actions:

i) accepting a signal at a port such as β;

ii) sending a signal at a complementary port such as β^-.

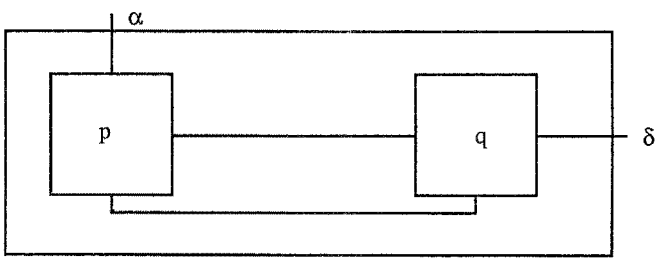
Thus the combined process above, p | q, can either communicate with the external environment via any one of the ports of its subcomponents or there can be internal synchronizations between the subcomponents, one via the pair of complementary ports such as β, β^-.

There are other combinators which may be used to modify existing processes. For each action, say β, there is a restriction operator $\backslash\beta$ which hides the ports β and β^-, i.e. it makes them unavailable for external communications. There is also a relabelling operation which simply relabels port names.

The combinators we have seen so far do not allow us to define the dynamic behaviour of processes. To do so we have other combinators. For example for each port name α there is a unary combinator $\alpha.$: $\alpha.r$ is a process which can synchronize via a port labelled α and then continue to act as process r. Thus processes can be viewed as machines which can perform certain kinds of actions. In pure CCS there is essentially one kind of action, synchronizing at a port. One possible definition of the processes p and q mentioned above, is given by the recursive definitions:

$$p \mathrel{<==} \alpha.\beta.\gamma^-.p \qquad \text{and} \qquad q \mathrel{<==} \beta^-.\delta.\gamma.q.$$

Conceptually processes can be viewed as black boxes with labelled ports at which they may communicate or synchronize. For example the combined process $(p \mid q)\backslash\beta\backslash\gamma$, can be represented by:



the only external synchronizations the process can perform are via α and δ. Moreover these are constrained by the internal synchronizations between β and β^- and γ and γ^- (represented by connected unlabelled ports since they are invisible to the external environment) in such a way that they can only be performed sequentially as in the process r defined by:

$$r \mathrel{<==} \alpha.\delta.r.$$

Indeed the semantics of CCS proposed in [Mil80] is such that these two different processes $(p \mid q)\backslash\beta\backslash\gamma$ and r are semantically equivalent and the laws of the calculus allow one to prove this equivalence using syntactic manipulations. However, two further combinators are needed both for expressiveness and for facilitating these syntactic manipulations. For example the simple process $\alpha.p \mid \beta.q$ exhibits *nondeterministic behaviour*: it can either perform α or β. Semantically it is equivalent to the nondeterministic process $\alpha.(p \mid \beta.q) + \beta.(\alpha.p \mid q)$. Here we have used the new combinator + : in general p + q can act either like p

or like q. So this term represents a process which can either do an α and then do (p | β.q), i.e. the residual of α.p | β.q after performing α, or do β followed by the residual after β, α.p | q. This new combinator is not sufficient to express all the forms of nondeterminism which CCS process can exhibit. For example the process (α.x + β.y) | ($\bar{\alpha}$.x' + γ.z)\α can either perform β or γ or there can be an *internal synchronization* between the ports α and $\bar{\alpha}$. To express this, a special action symbol, τ, is introduced which represents an internal synchronization which can not be influenced by the external world. Then we have the semantic equivalence:

(α.x + β.y) | ($\bar{\alpha}$.x' + γ.z)\α = β.... + γ.... + τ.(x | x').

(Here we have not included the residuals after β or δ). The extra term τ.(x | x') indicates the possibility of the process doing an internal move and then acting like the resulting process x | x'.

Indeed, these two combinators + and τ play a fundamental role in the theory of CCS. They are used in the *Expansion Theorem* of [Mil80] to show that every CCS process is equivalent to a purely nondeterministic process. In general specifications of process are given as nondeterministic processes while their implementations are usually built by combining subcomponents to run in parallel. The Expansion Theorem is used extensively to prove that specifications are equivalent to their implementations. Nevertheless the above mentioned combinators are unsatisfactory from many points of view.

The semantic equivalences used in [Mil80] and [DH84], are not preserved by +: they are not congruences. Also + exhibits a rather complicated mixture of intuitively different forms of nondeterministic behaviour, often referred to as *internal* and *external nondeterminism*, see [Hoa85] or [OH86]. In the process α.p + β.q there is external nondeterminism: if the user requests an α synchronization the process will oblige and subsequently act like p whereas if β is requested it will also be performed and the process will continue as q. Internal nondeterminism is exhibited in α.p + α.q and α.p + τ.q. In the first cases the process will oblige when asked to perform an α synchronization but the user will have no control over which of p or q the process will evolve to. The behaviour of processes such as αp + τq is difficult to describe and the operational semantics given in [Mil80] is not very illuminating for such terms. Moreover the need for a special symbol τ to represent internal actions is counterintuitive; if the actions are internal and invisible there should be no need to refer to them in the language or calculus. The laws governing the manipulation of τ in the calculus [HM85] are rather mysterious and to date nobody has been successful in providing an intuitive and acceptable model which explains the nature of τ.

Our suggestion is to replace these two troublesome combinators + and τ with two new combinators [] and $\oplus$; intuitively [] represents external nondeterminism and $\oplus$ internal nondeterminism. Both p[]q and p$\oplus$q act either like p or like q but in the former the environment or the user decides whereas in the latter the decision is made internally and it can not be influenced by the user. The resulting language has all the desirable properties of the original CCS, at least if we base our semantics on the theory of Testing, [DH84], [DeN85b], and not on observational equivalence as in [Mil80]. The equivalence is preserved by all of the combinators, i.e. it is a congruence; we have a modified *Expansion Theorem* and a *complete set of laws* and the natural model *Strong Acceptance Trees* [Hen85b] provides a fully-abstract model. In particular this model gives an intuitive explanation of the two combinators [] and $\oplus$ as functions over trees.

We now give an outline of the paper. We start with a basic language BCCS (for Basic CCS) which contains recursive definitions using the main combinators of CCS:

• **prefixing of actions, parallel composition, restriction** and **renaming**.

From this basic language we build two additional languages by adding new combinators.

CCS (or more precisely "pure" CCS) is obtained by adding:

• **nondeterministic choice** + and **internal action** τ.

The new version of CCS, which we call TCCS for Testing CCS, is obtained by adding instead:

• **external nondeterminism** [] and **internal nondeterminism** $\oplus$.

For both CCS and TCCS we define a semantic preorder (which generates an equivalence in the natural way) denoted by $\leq$ and $\leqq$ respectively. In the case of CCS this coincides with the **must** version of the testing preorders defined in [DH84], and for TCCS it is the natural modification of this preorder. This semantic equivalence is quite different than that employed in [Mil80], observational equivalence, both in its definition and in the kinds of processes it equates. Indeed, to obtain our results it is essential to use testing equivalence rather than observational equivalence as in the latter setting the new operators can not express all the nondetermism expressible in the basic language BCCS. The definitions of the semantic preorders rely on an operational semantics for the languages and that for TCCS is somewhat simpler than the usual one for CCS. All of this is presented in Section 2.

In Section 3, we recall the appropriate results for CCS from [DH84], show the modified Expansion Theorem for the new language TCCS and give a complete set of laws for the semantic preorder applied to this language. This last subject is merely sketched as it relies heavily on similar results in papers such as [Hen83], [DeN85b]. In Section 4, we give a translation from CCS to TCCS with the intention of showing that the use of the new operators does not change the essence of CCS. We hope that this is evident from the following properties of the translation

- for every process p in the basic language, BCCS, p and its translation $\mathbf{tr}(p)$ are identical;
- for every pair of processes in CCS $p \leq q$ if and only if $\mathbf{tr}(p) \leqq \mathbf{tr}(q)$.

We end in Section 5. with some remarks about application of observational equivalence to the new languag and a discussion of related work. All proofs are omitted; they will be given in the complete version of th paper.

2. Two Languages for Communicating Systems CCS and TCCS

In this section we first present the two languages, then discuss the experimental setting for defining testin equivalences. Both languages consist of a set of operators for constructing new terms from preexisting one Agents of the languages will be closed terms (i.e. terms without free variables) which can be generated b the following BNF-like schema:

$$t ::= x \mid op(t_1, \ldots, t_k), op \in \Sigma^k \mid \text{rec } x.\ t$$

where x is a variable and Σ^k is a set of operators of arity k. We use Σ to denote $\cup \{\Sigma^k \mid k \geq 0\}$; the set of recursive terms which can be obtained once we have fixed Σ will be denoted by REC_Σ. $CREC_\Sigma$ is used to denote the set of all closed terms.

We will assume an uninterpreted set of elementary (atomic) actions which will be the basic constructors of our processes. In particular we will let

- $\Delta = \{\alpha, \beta, \gamma \ldots\}$ be a fixed set and $\Delta^- = \{\alpha^- \mid \alpha \in \Delta\}$;
- $\Lambda = \Delta \cup \Delta^-$ (ranged over by λ) be the set of *visible actions*.

The two language we will consider will share a number of constructors listed below together with a short comment on their intended meaning:

Inaction the term **NIL** is used to represent a process which never performs any action.

Undefined the term Ω is used to represent the totally undefined process.

Action if p is a term then λp is a term which represents the process which can perform action λ and behave like p.

Restriction if p is a term then p\α is a term which represents the process which can perform the same actions as p apart for α and α^-.

Renaming if p is a term then p[Φ] is a term which represents the process whose actions are renamings via Φ of all the actions of p.

Synchronization if p and q are terms then p | q is a term which represents the process which can perform an arbitrary interleaving of the actions of p and q and additionally synchronize their complementary actions.

We will call the language consisting of the above operators *Basic* CCS (BCCS) and use it to build both CCS and the proposed new language which we will call *Testing* CCS (TCCS). Terms of all the three languages will be built from the BNF-like schema above. In particular, when dealing with BCCS, we will have:

$\Sigma^0_{BCCS} = \{\Omega, NIL\}$;
$\Sigma^1_{BCCS} = \Lambda \cup \{\backslash\alpha \mid \alpha \in \Delta\} \cup \{[\Phi] \mid \Phi$ is a renaming of Λ which preserves complementation$\}$;
$\Sigma^2_{BCCS} = \{|\}$;
$\Sigma^k_{BCCS} = \emptyset$ if $k \geq 3$.

2.1. CCS

In this subsection we will give a brief summary of CCS and of the experimental setting and results presented in [DH84]. The resumé will guide us toward defining experiments, preorders and equivalences on the modified version of CCS. CCS agents are closed terms which can be generated by the syntax above when we take $\Sigma = \Sigma_{CCS} = \Sigma_{BCCS} \cup \{\tau, +\}$ where τ is a distinguished atomic *invisible* action not in Λ, and + is the so called *choice operator*. If t and u are CCS terms then **t + u** is a CCS term which denotes a process which can behave either like t or like u and the choice depends sometimes on the external environment some others it is made internally. In the sequel we will let $\Lambda \cup \{\tau\}$ be ranged over by μ. Moreover, we will let $REC_{\Sigma CCS}$ denote the set of all CCS terms generated by the above syntax, ranged

over by t, u,... and $CREC_{\Sigma_{CCS}}$ will be used to denote the set of all closed CCS terms, with p, q,... as metavariables.

CCS has been equipped with an interleaving operational semantics based on labelled transition systems, the transition relation of which is defined by a set of transition rules over agents. A relation $\text{—}\mu\text{→}$, called *derivation relation*, is defined, in the SOS style [Plo81], with the intuition that agent t_1 may evolve to become agent t_2 either by reacting to a λ-stimulus from its environment ($t_1\text{—}\lambda\text{→}t_2$) or by performing an internal action which is independent of the environment ($t_1\text{—}\tau\text{→}t_2$).

Definition 2.1.1

Milner's derivation relation $t_1\text{—}\mu\text{→}t_2$ is defined as the least relation satisfying the following axiom and inference rules.

Act) $\mu t \text{—}\mu\text{→} t$

Res) $t_1 \text{—}\mu\text{→} t_2$ **implies** $t_1\backslash\alpha \text{—}\mu\text{→} t_2\backslash\alpha, \quad \mu \notin \{\alpha,\alpha^-\}$

Rel) $t_1 \text{—}\mu\text{→} t_2$ **implies** $t_1[\Phi] \text{—}\Phi(\mu)\text{→} t_2[\Phi]$

Sum) $t_1 \text{—}\mu\text{→} t_2$ **implies** $t_1+t \text{—}\mu\text{→} t_2$ **and** $t+t_1 \text{—}\mu\text{→} t_2$

Com) $t_1 \text{—}\mu\text{→} t_2$ **implies** $t_1|t \text{—}\mu\text{→} t_2|t$ **and** $t|t_1 \text{—}\mu\text{→} t|t_2$

$t_1 \text{—}\lambda\text{→} t_2$ **and** $t'_1 \text{—}\lambda^-\text{→} t'_2$ **implies** $t_1|t'_1 \text{—}\tau\text{→} t_2|t'_2$

Rec) $t_1[\text{rec x. } t_1/x] \text{—}\mu\text{→} t_2$ **implies** $\text{rec x. } p_1 \text{—}\mu\text{→} p_2$. ♦

The derivation relation above completely specifies the operational semantics of CCS; a second level of CCS semantics is defined on top of this to obtain more abstract descriptions of system's behaviours. To this purpose, a notion of testing is introduced in [DH84] which is then used to define equivalence relations on CCS terms which allow one to identify agents which are "behaviourally" equivalent. In [DH84], processes which react in the same way to experiments performed by *external observers* are considered as equivalent. **Observers** are just terms over $\Sigma_{CCS} \cup \{w\}$, where w is a distinguished action symbol, not in Λ, used as a special action which "reports success" of an experiment. This theory leads to three preorders on processes which are based on the possibilities processes have of always (*must*) or sometimes (*may*) satisfying observers. We will concentrate on one of the preorders discussed there, namely on the one which considers as satisfactory only those experiments (sets of interactions between a process and an observer) which always report a success.

The outcomes of the interaction between processes and observers are obtained by studying the set of computations which take place because of synchronizations between processes and observers or because of silent transitions. To this purpose the notion of **complete computation**, i.e. of a computation which is either infinite or such that the terminal pair, <state of the process, state of the observer>, can not perform a further synchronization or silent move is very important. Also, to be able to describe the outcomes of experiments on partially specified objects and on terms specified via unguarded recursive definitions, [Mil80], a predicate $\downarrow$ on CCS terms is also required:

Definition 2.1.2.
Let $\downarrow$ be the least predicate on terms which satisfies

i. NIL$\downarrow$, $\lambda p\downarrow$,

ii. $p\downarrow$ and $q\downarrow$ implies $(p + q)\downarrow$, $(p|q)\downarrow$, $(p[\Phi])\downarrow$ and $(p\backslash\alpha)\downarrow$

iii. $(t[\text{rec } x.\ t/x])\downarrow$ implies $(\text{rec } x.\ t)\downarrow$ ♦

The converse of $\downarrow$ is denoted by $\uparrow$, i.e. $p\uparrow$ (read p diverges) if not $p\downarrow$ (read p converges).

Based on the notions above and on CCS operational semantics we have:

Definition 2.1.3
If o is an observer in $REC_{\Sigma CCS} \cup \{w\}$ then:
p **must satisfy** o if whenever $p|o = p_0|o_0 \text{–}\tau\rightarrow p_1|o_1 \text{–}\tau\rightarrow \ldots$. is a *maximal* computation then there exists $n \geq 0$ such that $o_n \text{–}w\rightarrow$ and $p_k|o_k\uparrow$ implies $o_h \text{–}w\rightarrow$ for some $h \leq k$. ♦

This predicate is the basis of the preorder on CCS terms reported below. In its definition and in the rest of the section, $\mathbf{O}$ is used to denote the set of all the observers in $REC_{\Sigma CCS} \cup \{w\}$.

Definition 2.1.4.
$p \leq q$ if and only if $\forall o \in \mathbf{O}$, p **must satisfy** o implies q **must satisfy** o . ♦

This preorder is by and large well behaved and has many interesting properties. However it not preserved by all the CCS operators; in particular it is not preserved by the operator +, in the sense that we have αNIL $\leq \tau\alpha$NIL but *not*(βNIL + αNIL $\leq \beta$NIL + $\tau\alpha$NIL). As usual a new preorder $\leq^c$ can be defined which is based on $\leq$ but is preserved by all CCS operators. To show this we use a notion of context, $\mathbf{C}[\ ]$, an expression with zero or more "holes", to be filled by terms. We write $\mathbf{C}[t]$ for the result of inserting t into each hole. The notion of context allows then to define

$p \leq^c q$ if and only if for every CCS context $\mathbf{C}[\ ]$ we have $\mathbf{C}[p] \leq \mathbf{C}[q]$;

In [DH84] it is also shown that a direct characterization can be given for $\leq^c$.

Proposition 2.1.5.
$p \leq^c q$ if and only if $p \leq q$ and ($p\downarrow$ and $q\text{–}\tau\rightarrow$) implies $p\text{–}\tau\rightarrow$. ♦

2.2. Testing CCS an alternative to CCS

The TCCS agents are closed terms which can be generated by the general BNF-like schema above with $\Sigma = \Sigma_{TCCS} = \Sigma_{BCCS} \cup \{[], \oplus\}$ where $\oplus$ and [] are two new binary operators called *internal and external choice* respectively.

As with p + q we have that if p and q are TCCS terms then $\mathbf{p} \oplus \mathbf{q}$ and **p [] q** are TCCS terms denoting processes which can behave either like p or like q. The choice in the case of p [] q depends on the external

environment while in the case of $p \oplus q$ is taken internally without the environment having any control over it.

We will follow the same notational conventions of CCS also for TCCS terms but we will use capital letters instead of lower case ones. Namely, we will let $REC_{\Sigma TCCS}$ denote the set of all TCCS agents, T, U,... will be used to range over it, and $CREC_{\Sigma TCCS}$ will be used to denote the set of all closed TCCS terms, ranged over by P, Q,... . Also TCCS will be given a two level semantics; the first level is given by:

Definition 2.2.1

The *new derivation relation* consists of a pair of arrows, a labelled one, $T_1 \sim\lambda\sim> T_2$, and an unlabelled one, $T_1 \sim\sim> T_2$. It is defined as the least pair of relations satisfying the following axioms and inference rules.

Act) $\lambda T \sim\lambda\sim> T$

Res) $T_1 \sim\lambda\sim> T_2$ **implies** $T_1\backslash\alpha \sim\lambda\sim> T_2\backslash\alpha, \quad \lambda \notin \{\alpha,\alpha^-\}$

$T_1 \sim\sim> T_2$ **implies** $T_1\backslash\alpha \sim\sim> T_2\backslash\alpha,$

Rel) $T_1 \sim\lambda\sim> T_2$ **implies** $T_1[\Phi] \sim\Phi(\lambda)\sim> T_2[\Phi]$

$T_1 \sim\sim> T_2$ **implies** $T_1[\Phi] \sim\sim> T_2[\Phi]$

Ext) $T_1 \sim\lambda\sim> T_2$ **implies** $T_1 [] T \sim\lambda\sim> T_2$ **and** $T [] T_1 \sim\lambda\sim> T_2$

$T_1 \sim\sim> T_2$ **implies** $T_1 [] T \sim\sim> T_2 [] T$ **and** $T [] T_1 \sim\sim> T [] T_2$

Int) $T_1 \oplus T_2 \sim\sim> T_1$ **and** $T_1 \oplus T_2 \sim\sim> T_2$

Com) $T_1 \sim\lambda\sim> T_2$ **implies** $T_1|T \sim\lambda\sim> T_2|T$ **and** $T|T_1 \sim\lambda\sim> T|T_2$

$T_1 \sim\lambda\sim> T_2$ **and** $T'_1 \sim\lambda^-\sim> T'_2$ **implies** $T_1|T'_1 \sim\sim> T_2|T'_2$

$T_1 \sim\sim> T_2$ **implies** $T_1|T \sim\sim> T_2|T$ **and** $T|T_1 \sim\sim> T|T_2$

Rec) $\text{rec } x.\, T \sim\sim> T[\text{rec } x.\, T/x]$

Und) $\Omega \sim\sim> \Omega$ ♦

The operational semantics of the two choice operators and the new invisible move, $\sim\sim>$, which is different from $-\tau\rightarrow$ deserve some comments. We have that the two rules for $\oplus$ simply say that process $P \oplus Q$ could exhibit P's or Q's behaviour since it can perform an invisible move to any of them. In the case of [] we have that process P [] Q can take a final decision as to which behaviour to exhibit only after performing a visible action; invisible moves leave the choice still open. As for the other operators, the above operational semantics is very similar to the one given in Definition 2.1.1 for CCS. Some differences are however worth noting. There is an axiom also for the undefined process Ω and we take a different approach for determining the moves of recursive terms; instead of inferring the moves of a recursively defined term from the moves of their unwindings we simply have an axioms which allows unwinding. Note that they are the particular nature of the nondeterministic operator [] and the different kind of invisible moves which allow us to do this; had we done it for the original CCS and used τ to unwind recursive terms, the resulting semantics would have been very different because unwinding could preempt occurrences of other actions.

As with CCS in the previous section, we can define a set of observers and a set of experiments to define a testing preorder on the new language. **Observers** are just terms over $\Sigma_{TCCS} \cup \{w\}$. The set of all such observers will be denoted by $\mathbf{O}_{\mathcal{T}}$. Now, the machinery outlined above for the original CCS and the operational semantics for TCCS allow us to have:

Definition 2.2.2.
Given an observer $O \in \mathbf{O}_{\mathcal{T}}$ we have:
P **MUST SATISFY** O if whenever $P|O = P_0|O_0 \rightsquigarrow P_1|O_1 \rightsquigarrow \ldots$ is a *maximal* computation then there exists $n \geq 0$ such that $O_n \xrightarrow{w}$ ♦

Again, by using the above predicate a preorder on TCCS terms can be defined. Note that because of the new semantics for Ω and for recursive terms, we do not need to define any divergence predicate to be able to evaluate the effect of experimentations on underspecified processes or on processes specified via unguarded recursive definition. Indeed in these cases, we will always have an infinite computation from P|O which never reports success, and this means that for any observer O, which does not report success before starting experimentations, we have *not*(P **MUST SATISFY** O) whenever P is equal to Ω or unguarded.

Definition 2.2.3.
$P \ll Q$ if and only if $\forall O \in \mathbf{O}_{\mathcal{T}}$, P **MUST SATISFY** O implies Q **MUST SATISFY** O . ♦

We have that, while the testing preorder for CCS presented in the previous section is not preserved by the + contexts, the testing preorder for the new language is preserved by all the TCCS operators and admits a direct algebraic characterization, discussed in the next section.

Proposition 2.2.4.
$P \ll Q$ if and only if for every TCCS context $\mathcal{C}[\ ]$, $\mathcal{C}[P] \ll \mathcal{C}[Q]$ ♦

3. Algebraic Characterizations

In [DH84] three sound and complete proof systems for CCS testing precongruences based on testing preorders are introduced which consist essentially of a set of axioms to manipulate process expressions, the usual rules about transitivity and commutativity and a form of ω-induction. In [DH84] it is also proved that the set of axioms can be used to obtain three fully abstract models for CCS, i.e. models within which processes are distinguished if and only if they are distinguished by the associated set of tests. Moreover, the models, at first built in a very abstract way from the syntax of the language via a set of axioms [GTWW77], are proved isomorphic to a particular class of trees called *Representation Trees*. The complete set of axioms relative to the must based precongruence given in [DH84] can be ideally divided in two groups, namely those about the basic processes NIL and Ω, the visible actions, the invisible action τ and the choice operator + and those about restriction, relabelling and parallel composition. Indeed, the axioms in the second group show that the last three operators are not primitive; every finite CCS term containing |, $\backslash\alpha$ or $[\Phi]$ can be reduced to an equivalent one which does not contain these operators.

Also for the new language we are able to exhibit a complete proof system, which is based directly on the testing preorder and differs from the previous one only because of the axioms about the parallel composition operators and (obviously) because of the axioms about the new choice operators.

Below, we show that the operator for parallel composition can also be expressed in TCCS in terms of more basic operators such as $[]$, $\oplus$, NIL, Ω and λ. First, we fix some notation: If $I = \{i1, i2, \ldots, in\}$, we will let

- $\Sigma \{P_i \mid i \in I\}$ denote $P_{i1}\ [] P_{i2}\ [] \ldots []P_{in}$ if $I \neq \{\}$ and denote NIL otherwise.
- $\Pi \{P_i \mid i \in I\}$ denote $P_{i1} \oplus P_{i2} \oplus \ldots \oplus P_{in}$ whenever $I \neq \{\}$.

The new expansion theorem basically says that a process which can perform an internal communication can be seen as a process which can either perform one of these internal moves to become a new process or stay idle while all potential, visible and invisible, actions are possible.

Theorem 3.1 (*New Expansion theorem*)
If $P = \Sigma \{\lambda_i P_i \mid i \in I\}$ and $Q = \Sigma \{\nu_j Q_j \mid j \in J\}$ then

$P \mid Q = EXT$ if $COM = \emptyset$;
$= (EXT\ []\ \Pi COM) \oplus \Pi\, COM$ otherwise;

where $COM = \{P_i \mid Q_j \mid \lambda_i = \nu_j^- \text{ and } i \in I, j \in J\})$ and
$EXT = \Sigma \{\lambda_i(P_i \mid Q) \mid i \in I\}[] \Sigma \{\nu_j(P \mid Q_j \mid j \in J\}$ ♦

$X\ []\ X = X$	EXT1
$X\ []\ Y = Y\ []\ X$	EXT2
$X\ []\ (Y\ []\ Z) = (X\ []\ Y)\ []\ Z$	EXT3
$X\ []\ NIL = X$	EXT4
$X \oplus X = X$	INT1
$X \oplus Y = Y \oplus X$	INT2
$X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$	INT3
$X \oplus Y \leq X$	INT4
$\lambda X \oplus \lambda Y = \lambda(X \oplus Y)$	MIX1
$\lambda X\ []\ \lambda Y = \lambda(X \oplus Y)$	MIX2
$X \oplus (Y\ []\ Z) = (X \oplus Y)\ []\ (X \oplus Z)$	MIX3
$X\ []\ (Y \oplus Z) = (X\ []\ Y) \oplus (X\ []\ Z)$	MIX4
$\Omega \leq X$	UND1
$\Omega\ []\ X \leq \Omega$	UND2

TABLE 3.1.: Axioms for primitive TCCS

We are now ready to give the set of axioms for the new language which are sound and complete with respect to the notion of testing defined in Section 2.2. As with CCS, we can define a complete proof system for TCCS and exhibit a concrete fully abstract model based on trees. The tree model turns out to be that of Strong Acceptance Trees discussed in [Hen85a] and [Hen85b]; it is also a subdomain of Strong Representation Trees of [DH84] which is obtained by removing some anomalies of the original domain, namely the dishomogeneity in the treatment of the labels for the root and the other nodes of the trees, which were introduced to deal properly with internal actions.

The actual axioms are presented in Table 3.1 and Table 3.2, they have been split to separate those about TCCS primitive operators from those about the derivable operators.

$NIL[\Phi] = NIL$	REL1
$(X \,[]\, Y)[\Phi] = X[\Phi] \,[]\, Y[\Phi]$	REL2
$(X \oplus Y)[\Phi] = X[\Phi] \oplus Y[\Phi]$	REL3
$\mu X[\Phi] = \Phi(\mu)X[\Phi]$	REL4
$NIL\backslash\alpha = NIL$	RES1
$(X \,[]\, Y)\backslash\alpha = X\backslash\alpha \,[]\, Y\backslash\alpha$	RES2
$(X \oplus Y)\backslash\alpha = X\backslash\alpha \oplus Y\backslash\alpha$	RES3
$(\mu X)\backslash\alpha = \mu(X\backslash\alpha)$ if $\mu \neq \{\alpha, \alpha^-\}$	
NIL otherwise	RES4
$(X \oplus Y) \mid Z = (X \mid Z) \oplus (Y \mid Z)$	PAR1
If $P = \Sigma\{\lambda_i P_i \mid i \in I\}$ *and* $Q = \Sigma\{\nu_j Q_j \mid j \in J\}$ *then*	
$P \mid Q =$ EXT *if* $COM = \emptyset$;	
$=$ (EXT $[]\ \Pi COM) \oplus \Pi\, COM$ otherwise;	
where $EXT = \Sigma\{\lambda_i(P_i \mid Q) \mid i \in I\} [] \Sigma\{\nu_j(P \mid Q_j \mid j \in J\}$	
and $COM = \{P_i \mid Q_j \mid \lambda_i = \nu_j^-$ and $i \in I, j \in J\})$	PAR2
$\Omega[\Phi] = \Omega$	UND3
$\Omega\backslash\alpha = \Omega$	UND4
$P \mid \Omega = \Omega$	UND5
$\Omega \mid P = \Omega$	UND6

TABLE 3.2.: Axioms for TCCS derived operators

Indeed we have that the inequations in Table 3.1 are exactly those of [Hen85a], which are there used to characterize the initial algebra isomorphic to the domain of Strong Acceptance Trees. On the other hand we have that the laws in Table 3.2, apart for the expansion theorem, are the same as the ones given in [DH84]. These laws show that the axioms in Table 3.2 are sufficient to reduce every finite TCCS term containing |,

$\backslash\alpha$ or $[\Phi]$ to an equivalent one which does not contain these operators.

The theorem below states soundness and completeness with respect to testing equivalence of the set of axioms contained in the two tables. We use $A \vdash T \leq U$ to indicate that $T \leq U$ can be derived from the axioms in Table 3.1 and Table 3.2 by using ω-induction, and other natural properties of substitutive partial orders, [Hen85a] and [Gue81].

Theorem 3.2.
$A \vdash T \leq U$ if and only if $T \underline{\ll} U$. ♦

4. Translation: CCS → TCCS

In this section we present a translation function, **tr**, which given any CCS term uses induction on its structure to translate it into a TCCS term. The translation leaves most of the language unchanged. In fact, **tr**, when applied to Basic CCS terms, is just the identity function. It only erases τ actions and translates a term whose main operator is + to a term whose main operator is either [] or $\oplus$ depending on whether + represents internal or external choice. In particular if the term which is translated can perform silent (internal) moves, both the internal and external choice operators are used for the translation; the internal choice operator is however the main one. If the term to be translated can only perform initial visible moves then only the external choice operator is used to express it.

Definition 4.1.
tr is a function from $REC_{\Sigma CCS}$ to $REC_{\Sigma TCCS}$ defined by structural induction as follows

$\mathbf{tr}(NIL) = NIL$
$\mathbf{tr}(x) = x$
$\mathbf{tr}(\Omega) = \Omega$
$\mathbf{tr}(\lambda p) = \lambda \mathbf{tr}(p)$
$\mathbf{tr}(\tau p) = \mathbf{tr}(p)$
$\mathbf{tr}(p\backslash\alpha) = (\mathbf{tr}(p))\backslash\alpha$
$\mathbf{tr}(p[\Phi]) = (\mathbf{tr}(p))[\Phi]$
$\mathbf{tr}(\text{rec x. p}) = \text{rec x. } \mathbf{tr}(p)$
$\mathbf{tr}(p \mid q) = \mathbf{tr}(p) \mid \mathbf{tr}(q)$
$\mathbf{tr}(p + q) = \mathbf{tr}(p)\ [] \ \mathbf{tr}(q)$ if *not* ($p \xrightarrow{\tau} p'$ or $q \xrightarrow{\tau} p'$ for some p')
$\mathbf{tr}(p + q) = (\mathbf{tr}(p)\ [] \ \mathbf{tr}(q)) \oplus \Pi\{\mathbf{tr}(p') \mid (p \xrightarrow{\tau} p' \text{ or } q \xrightarrow{\tau} p')\}$. ♦

Some examples should help in understanding how the actual translation works.

Example 4.2

i) $\mathbf{tr}(\alpha NIL + \tau\beta NIL) = (\alpha NIL \,[]\, \beta NIL) \oplus \beta NIL$;

ii) $\mathbf{tr}((\alpha NIL + \beta NIL) \mid (\bar{\alpha} NIL + \gamma NIL)) = (\alpha NIL \,[]\, \beta NIL) \mid (\bar{\alpha} NIL \,[]\, \gamma NIL)$;

iii) $\mathbf{tr}(\text{rec } x.\ \tau x + \alpha NIL) = \text{rec } x.\ (x \,[]\, \alpha NIL) \oplus x$. ♦

Example i) above shows how we translate processes which can perform initial invisible actions. On the other hand, Example ii) shows that the translation of parallel processes is immediate, we simply translate separately the two parallel processes and put them in parallel. Finally, Example iii) shows how recursive terms which contain variables which are guarded by invisible actions are translated into terms defined via unguarded recursion; such terms will lead to diverging computations via infinite applications of the rule rec) of Definition 2.2.1.

The translation given above does not change the essence of CCS. In fact, as we have already seen, the two languages share most of their operators, and we also have that the testing preorder defined for CCS and the testing preorder defined for TCCS induce the same identifications when applied to the common sublanguage BCCS. Moreover we can exhibit a theorem which shows that the use of new CCS operators for describing nondeterministic processes does not change the nature of the language whenever testing equivalence is used to describe abstract behaviours. In fact, we have that two CCS terms are identified by a testing equivalence based semantics if and only if also their translations in TCCS via $\mathbf{tr}$ are identified.

Theorem 4.3.

If $p, q \in CREC_{\Sigma BCCS}$ and $o \in CREC_{\Sigma BCCS} \cup \{w\}$ then

p **must satisfy** o if and only if p **MUST SATISFY** o. ♦

Theorem 4.4.

If p, q are two CCS agents then $p \leq q$ if and only if $\mathbf{tr}(p) \lessapprox \mathbf{tr}(q)$. ♦

5. Discussion

The main point of this paper is that one can can develop an adequate version of CCS which does not use the special combinator τ for internal actions. Instead we replace the choice operator +, whose semantics is also somewhat unclear, by two new choice operators $\oplus$ and [], representing internal and external nondeterminism respectively. The operational semantics for the resulting language is much simpler and the definition of the testing preorder is also significantly cleaner.

The reader familiar with [Hoa85] will have already recognised our new operators: [] is the strict version, developed in [DeN85a], of the operator with the same name proposed in [BHR84] and $\oplus$ is the version of $\sqcap$ discussed in [Hoa85] based on the operational semantics given in [Hen83]. In other words the purely nondeterministic part of our language more or less coincides with that of Hoare's language, TCSP. Both languages, TCCS and TCSP share this basis and differ only in the form of parallelism and abstraction they use. So here we have merely employed the theory developed in [Hen83], [Hen85a], [DeN85b] for

nondeterministic machines to explain the particular choices of parallelism and abstraction used in CCS (although the transition system on which our new operational semantics is based is somewhat different than that of [Hen83]).

We believe that this theory of nondeterministic machines is sufficiently powerful to explain many other choices such as those in [BK84], [Hoa85] and [Miln85], at least if one is willing to accept an interleaving semantics. Indeed, the semantics developed in [Hoa85] and [BHR84] for TCSP is entirely denotational and the choice of model has always remained formally unjustified. [Bro83] proposes a set of transition rules for TCSP, however there no account is given of the relationships between the induced operational semantics and the original denotational semantics based on refusal sets. [OH86] proposes another set of transition rules, very similar to ours, and uses the observational equivalence of [Mil80] to abstract from unwanted details, thus obtaining a new operational semantics for TCSP. The denotational semantics based on refusal sets is then proved consistent with respect to the obtained operational semantics, in the sense that it preserves all the operational equivalences between processes. However, no completeness result is given, which guarantees that the denotational equivalences between terms are only those operationally provable. Indeed, we have that the proposed denotational semantics is more abstract than the operational one.

The testing preorder developed in Section 2 provides the necessary machinery for defining an operational semantics which is in full agreement with the denotational semantics in the sense that it identifies all and only the processes identified by the denotational approach based on refusal sets. In fact, it is a simple matter to extend the testing preorder to the form of parallel composition used in TCSP. The domain of Bounded Refusal Sets [DeN85a], a modification of the original model dealing properly with diverging processes which is isomorphic to the *Strong Acceptance Trees* of [Hen85b], is then fully abstract (consistent and complete) with respect to this behavioural equivalence.

It is worth pointing out that the new operators [] and $\oplus$ are no longer sufficiently expressive if we use observational equivalence in place of testing equivalence. One can easily argue that the term $\alpha NIL + \tau\beta NIL$ can not be observationally equivalent to any purely nondeterministic process expressible in the language which has $\oplus$ and [] in place of + and τ, although we have not explained formally how observational equivalence is defined for the new language.

References

[AB84] Austry,D. and Boudol,G. Algebre de Processus et Synchronization. *Theoret. Comput. Sci.* Vol. 30, No. 1 North Holland, Amsterdam, (1984).

[dBZ82] de Bakker,J. and Zucker,J., Processes and the Denotational Semantics of Concurrency. *Information and Control*, Vol 44, Nos. 1-2, pp.136-176, (1982).

[BHR83] Brookes,S.D. A Model for Communicating Sequential Processes. Ph.D. Thesis, University of Oxford. Also Carnegie Mellon University Internal Report, CMU-CS-149, (1983).

[BHR84] Brookes,S.D., Hoare,C.A.R. and Roscoe,A.D. A Theory of Communicating Sequential Processes. *Journal of ACM*, Vol. 31, No. 3, pp. 560-599, (1984).

[BK84] Bergstra,J. and Klop,G. Process Algebra for Synchronous Communication, *Information and Control*, Vol 60, pp.109-137, (1984).

[DH84] De Nicola,R. and Hennessy,M. Testing Equivalences for Processes. *Theoret. Comput. Sci.*, Vol.34, pp. 83-133, North Holland, Amsterdam, (1984).

[DeN85a] De Nicola, R. Two Complete Set of Axioms for a Theory of Communicating Sequential Processes. *Information and Control*, Vol 64, Nos. 1-3, pp.136-176, (1985).

[DeN85b] De Nicola, R. Fully Abstract Models and Testing Equivalences for Communicating Processes. Ph.D. Thesis, University of Edinburgh CST-36-85, (1985).

[GTWW77] Goguen,J.A., Thatcher,J.W., Wagner,E.G. and Wright,J.B. Initial Algebra Semantics and Continuous Algebras. *Journal of ACM*, Vol. 24, No. 1, pp. 68-95, (1977).

[Gue81] Guessarian,I. *Algebraic Semantics*. LNCS 99, (1981).

[Hen83] Hennessy,M. Synchronous and Asynchronous Experiments on Processes. *Information and Control* Vol. 59, Nos. 1-3, pp. 36-83, (1983).

[Hen85a] Hennessy,M. An Algebraic Theory of Processes. Lecture Notes, Aarhus University, (1985).

[Hen85b] Hennessy,M. Acceptance Trees. *Journal of ACM*, Vol. 32, No 4, pp. 896-928, (1985).

[HM85] Hennessy,M., Milner,R. Algebraic Laws for Nondeterminism and Concurrency. *Journal of ACM*, Vol.32, No. 1, pp. 137-161, (1985).

[Hoa85] Hoare,C.A.R. *Communicating Sequential Processes*. Prentice Hall (1985).

[ISO86] International Standard Organization, LOTOS - A Formal Description Technique. Internal Report Twente University of Technology and ISO/TC97/SC21 Draft Proposal 8807, (1986)

[Mil80] Milner,R. *A Calculus of Communicating Systems*, LNCS 92, (1980).

[Miln85] Milne,G. CIRCAL and the Representation of Communication, Concurrency and Time. *ACM Toplas* Vol. 7, No. 2, pp. 270-298, (1985).

[OH86] Olderog, E-R, Hoare C.A.R. Specification-Oriented Semantics for Communicating Processes, *Acta Informatica* Vol. 23, pp. 9-66, (1986).

[Plo81] Plotkin,G. A Structural Approach to Operational Semantics, Lecture Notes, Aarhus University, (1981).