Using the Axiomatic Presentation of Behavioural Equivalences for Manipulating CCS Specifications[1]

R. De Nicola♦, *P. Inverardi*♦, *M. Nesi*♥
♦ IEI - CNR, Via S. Maria 46, I-56100 Pisa, ITALY
♥ Consorzio Pisa Ricerche, Via Risorgimento 9, I-56100 Pisa, ITALY

Abstract
An interactive system for proving properties of CCS specifications is described. This system allows users to take advantage of all three views of CCS semantics (the transitions, the operationally defined equivalences and the axioms) and to define their own verification strategies for moving from one view to another. The system relies on term rewriting techniques and manipulates only the symbolic representation of specifications without resorting to any other kind of internal representation.

1. Introduction

It is by now evident that the analysis of concurrent systems, even elementary ones, is very delicate and error-prone. In this field computer assistance is essential, both to make the analysis possible and to ensure correctness. Indeed, very often, the verification of system properties consists of many repetitive, error-prone steps intermixed with a few conceptual ones which need human intervention.

Process algebras are generally recognized to be a convenient tool for describing concurrent, or more generally reactive, systems at different levels of abstraction [Mil80, BK84, DH84, BB87]. In general, these formalisms are equipped with one or more notions of equivalence or preorder. These notions are used to study the relationships between the different descriptions of the same system and to perform, in this way, part of the analysis. In these cases, the need for automatic tools to prove that two different accounts of the same system are equivalent or that one is a good approximation of the other is evident even when dealing with toy-examples.

There are already a few verification environments in which properties of concurrent systems can be proved: Concurrency Workbench [CPS88], TAV [GLZ89], Auto [Ver86], Squiggles [BC89]. All of these environments work in the framework of Milner's Calculus of Communicating Systems (CCS) [Mil80] and provide tools for analysing properties of CCS specifications; all of these can be used to verify equivalence of specifications and some also to decide whether a specification satisfies a logical (modal) property. It should be noted that most of these systems resort to a finite state machine representation of terms and use a generalized partitioning algorithm [BS87, KS83]. The only exception is the TAV system, directly based on

[1] This research has been partially supported by the CEC under ESPRIT project 2304 - LOTOSPHERE.

the definition of bisimulation. The main problem is that most of the time these systems deliver a yes/no answer and very frequently the answer is no; in this situation the user is not provided with any suggestion about what went wrong and in which part of the specification the error was located. The only notable exception is again TAV which, whenever two specifications are not equivalent, provides a Hennessy-Milner logic formula which differentiates them.

Another well-known problem of the approaches based on a finite state machine internal representation of specifications is state explosion. For these reasons, it is important that together with clues about the errors in the specifications, the user should be also provided with tools which allow him to control the flow of the proof so that he is also able to prune the state space.

Indeed, we think that the user should have a high control over the verification process; he might occasionally use some automatic tools, but in general he should perform the proof or at least should be able to control the interactions of the needed tools. If the verification environment is to control and program tools flexibly, it cannot rely on different internal representations of the symbolic terms, but has to manipulate the specifications in a "homogeneous" way.

In this paper we describe ongoing work on a system for manipulating and executing CCS specifications. The semantics of this calculus are defined in terms of labelled transition systems, by following the Structural Operational Semantics (SOS) approach [Plo81], and by using various notions of behavioural equivalence used to identify those agents which exhibit the same behaviour according to given classes of external observations. Some of the proposed equivalences possess a complete axiomatic presentation, which gives rise to another characterization of the semantics of the calculus.

One of the main goals of our system is to take advantage of all three views of CCS specifications, namely the transitions, the operationally defined equivalences and, whenever possible, the axioms which completely characterize the equivalences. This would allow users to define their own verification strategies and move from one view to another, thus using each time the most convenient one. For example, we want to offer the possibility of executing the operational semantics and reducing terms by means of behavioural equivalences within a flexible and open-ended system, that makes tools and not policies available to perform verification. To do this, we keep a homogeneous view of the specifications within the system; we stick to the symbolic representation of specification and never resort to any other kind of internal representation (finite state machines or similar).

We follow an *equational approach* to take advantage of the complete axiomatizations and rely on techniques borrowed from logic-functional programming: both the operational semantics and the axioms for behavioural equivalences are seen as Horn theories. This makes it possible to manage at metalevel (via metaprogramming) the intertwining between operational semantics and behavioural equivalences according to different user defined proof strategies. In this way we obtain the functionality that will be the basis for developing a language to define strategies.

When relying on axiomatic presentations, executing behavioural equivalences consists in transforming processes to equivalent normal forms and deciding congruence of two CCS processes by checking whether their normal forms are "similar". The axiomatic presentation of behavioural equivalences can be exploited in two different ways. It can be seen as an equational theory and executed by means of its associated canonical term rewriting system [HO80] and it can also be used to perform elementary transformations by applying single axioms on demand.

Summing up, our system offers three main functionalities. It can be used to run the operational semantics of a CCS process. It permits to obtain a reduced form of a specification by relying on the axiomatic characterization of the behavioural equivalences. Finally, it offers the user the possibility of manipulating the actual specification on demand, e.g. he can specify which axiom to apply or which transition to perform.

The actual system is presented in Section 2. Section 3 describes ongoing work to extend the present version. An example of use of the system is presented in Section 4. Future developments and conclusions are given in Section 5. The syntax and the operational semantics of CCS and the axiomatic presentations of several behavioural equivalences are reported in the Appendix.

2. Overview of the system

The system results from the integration of a semi-automatic tool [Sca88] and an automatic one [GIN88, Nes88] for executing and manipulating CCS processes.

The architecture of the system is shown in Figure 1; all the boxes stand for Prolog modules. By means of the user interface, it is possible to perform a certain set of operations on processes. Processes can be input according to the usual syntax checked via the Analyzer CCS module and the available operations are displayed by means of the menu. All other modules implement system functionalities. The Operational Semantics module implements the operational semantics. The Observational Semantics module implements an (incomplete) term rewriting system associated with the observational congruence [Mil80, HM85]. The Expansion Theorem module implements the expansion theorem in its full generality. The Axioms module implements the axiomatic presentation of the observational congruence and allows the user to apply single axioms to transform a process.

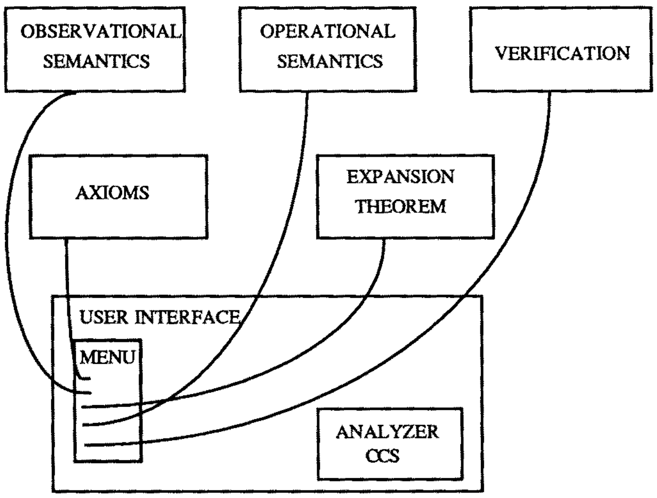

Figure 1

Finally, the Verification module implements a specific verification strategy, which proves the observational equivalence of two specifications S1, S2 trying to transform S1 into S2 [Mil80]. In this strategy S1 is rewritten by executing it according to the Operational Semantics module and by reducing the intermediate specifications according to the Observational Semantics module. A check based on replacing subexpressions with identifiers permits recursion to be dealt with and prevents infinite rewritings.

It should be noted that, although the actual system implements only observational congruence, the approach followed and the strategies developed can be extended to other behavioural equivalences: it is simply a matter of adding new boxes. At present, our system does not offer any built-in facility for defining verification strategies or tactics; an "ad hoc" metalanguage has not yet been defined. The user has to implement his own strategy as a separate module which could be accessed from within the system, exactly as we have done for the Verification module. The current interface of the system makes it possible to easily switch from one theory (module) to another, i.e. applying the operational semantics and then reducing to normal form by means of the observational module, etc. In Section 4, we give an example to show both the usage of the built-in verification tactic and of a user directed strategy.

The current version of the system has certain limits: these mainly concern the automatic tools and in particular the module for observational semantics. More specifically, the present implementation does not adequately deal with the associative and commutative (AC) axioms. In this respect, it is useful to remind the reader that we have taken a term rewriting approach to the exploitation of the axiomatic presentation of behavioural equivalences. The actual implementation, due to the AC axioms, divides the observational congruence theory (see Appendix) into two subtheories: $Eq_{Rewrite}$ consisting of the basic axioms A5-A11 and $Eq_{sum+\tau}$ consisting of the basic axioms A1-A4 and the τ-laws T1-T3. First, the equivalences in $Eq_{Rewrite}$, considered as rewrite rules directed from left to right, are applied to terms to rewrite a CCS specification *spec* into an equivalent one *spec'*, with no occurrences of relabelling, restriction and parallel composition operators. Afterward, the equivalences in $Eq_{sum+\tau}$ are applied to the action-tree of *spec'* to avoid explicitly dealing with the AC axioms. In this way, we obtain an incomplete decision procedure. In the next section, we discuss how this problem can be dealt with.

Another limit is the class of processes which can be treated. The term rewriting approach, in its basic definitions, works on canonical systems, thus the termination property, in addition to the confluence property, of our term rewriting system has to be guaranteed. In general, as soon as the rules for the recursion operator are introduced, we obtain a non terminating rewriting. Although this might appear to be a more restrictive requirement than confining attention to finite state machines, which permit instead the manipulation of a particular class of infinite processes, it should be noted that, even in the present version of the system, we have many different ways of treating recursive processes at the metalevel. The Verification module is an example of the possibilities available to a user for building his own strategy and for adding functionalities not supported by the kernel system. In fact, Verification provides the necessary facilities to cope with recursive processes, while no rule for recursion either in the Operational Semantics or in the Observational Semantics module has been provided.

3. Completing the axiomatizations of behavioural equivalences

Correct and complete axiomatic presentations for several behavioural equivalences (see Appendix) have been provided for finite CCS without value passing. These presentations are equational theories and we aim at verifying equivalences by applying the axioms. An equational theory can be executed by completing it into an equivalent canonical term rewriting system (if any), which can be obtained by means of a completion algorithm [HO80]. All the axiomatic presentations for the behavioural equivalences of CCS contain the AC axioms for the '+' operator and they can be properly treated by using a completion algorithm modulo AC [Hue80, PS81, JK86, BD87]. We have tried to complete, via the REVE system [Les83] by using the AC-completion, the axiomatic presentation of several behavioural equivalences. It comes out that some of these equivalences admit a finite canonical term rewriting system. Indeed, this is the case for the axiomatic presentation of trace congruence given in [DH84], of observational congruence for a subcalculus of CCS which does not contain the parallel composition operator [HM85] and of branching bisimulation congruence [GW89].

On the contrary, the axiomatic presentation of observational congruence [Mil85, HM85] and of testing congruence [DH84] do not admit a finite canonical term rewriting system. Indeed, the AC-completion of the theory, consisting in the τ-laws of observational congruence and the axioms for the '+' operator, results in an infinite term rewriting system since infinite critical pairs are generated. The same problem arises when trying to complete testing congruence; by checking the conditions given in [Her89a] we have already traced out one source of divergence.

Thus, for those behavioural equivalences which have an equivalent finite term rewriting system we provide a decision procedure: given two specifications, it is always possible to answer yes/no about their equivalence according to such theories. From those behavioural equivalences whose completion diverges, we have picked out the most popular one, namely the observational congruence. From [Mil85] and [HM85], we know that there exist normal forms for the observational congruence (from now on, we refer to them as obs-normal forms, see Appendix); indeed, their existence has been used to prove the completeness of its axiomatization. The actual completeness proof and in particular the proof of the so-called *absorption lemma* shows how the obs-normal form of a finite term can be obtained. We have simulated the absorption lemma in the framework of term rewriting and we have derived a strategy based on a specific rewriting relation. The main feature of this strategy is that the completion of the theory is never attempted. Instead, complete reductions are obtained by alternatively applying all axioms of the theory as reduction rules and some τ-laws as expansion rules. We have proved that the strategy applies all (and only) those reductions which could be obtained by means of the lemma. The strategy will be implemented by using parts of the REVE system and is outlined in the next subsection.

A rewriting strategy for observational congruence

Let OBS be the equational theory given by the basic axioms A1-A4 and the τ-laws for the observational congruence T1-T3 (see Appendix). Let R_{OBS} be the term rewriting system obtained by simply directing the axioms of OBS according to a fixed term ordering > [HO80].

As mentioned above, the proposed strategy for the verification of the observational congruence of finite CCS terms relies on the absorption lemma [HM85] reported below; see Appendix for missing definitions:

Absorption Lemma

If P' is a μ-derivative of P and P' $=_{OBS}$ Q, then P + μ.Q $=_{OBS}$ P.

This means that, in order to reduce a term, we can delete those summands μ.Q which are "semantically contained" in others by means of the notion of μ-derivative.

The proof of the lemma makes use of structural induction; the inductive hypothesis and the two τ-laws T2-T3 are applied as expansion rules to transform P + μ.Q into an equivalent term, which is congruent to a term without μ.Q. Once μ.Q has been deleted, the resulting term is rewritten by applying the inductive hypothesis and the axioms of OBS as reduction rules to eventually obtain the reduced term P.

Our strategy does "simulate" the proof of the lemma without using any induction. We remind the reader that we aim at applying the strategy to the symbolic representation of the term and implementing it in the framework of term rewriting. Thus, the strategy rewrites P + μ.Q using *only* the two τ-laws T2-T3 as expansion rules, until it is possible to apply a rule of R_{OBS} to delete μ.Q. Once μ.Q has been deleted, since the previous transformations on P + μ.Q may also have changed P, P is rebuilt by applying those reductions, which are exactly opposite to the previous expansions, on the current term.

In order to define the above strategy we have to:

- cope with the problem of stopping the expansion process;
- find reliable criteria to perform reductions when rebuilding P.

Given a rewriting relation $\rightarrow_R$, let $\rightarrow_R{}^*$ be its reflexive-transitive closure. A term P is in normal form according to R if P cannot be further reduced according to the rules of R. Let an AC-rewriting relation be a rewriting relation modulo the AC axioms. Let $\rightarrow_{R_{OBS}}$ be the AC-rewriting relation according to R_{OBS}, the term P" be in normal form according to R_{OBS}, $\rightarrow_\tau$ be the AC-rewriting relation expanding terms according to the axioms T2-T3. Finally, let $\rightarrow_{R'_{OBS}}$ and $\rightarrow_{R''_{OBS}}$ be AC-rewriting relations according to R_{OBS} following specific redex selection criteria. Our strategy is a rewriting relation P $\rightarrow_{strat}$ P' defined as follows:

Absorption Strategy

$P \rightarrow_{strat} P' \;=\; P \rightarrow_{R_{OBS}}{}^* P'' \,(\rightarrow_\tau{}^* \bullet \rightarrow_{R'_{OBS}}{}^*)\, {}^* P''' \bullet \rightarrow_{R''_{OBS}}{}^* P'.$

Given a term P, the rewriting relation $\rightarrow_{R_{OBS}}$ computes a normal form P" of P according to R_{OBS}. P" is checked for obs-normal form by looking for summands to be deleted according to the absorption lemma. Thus, the rewriting relation $\rightarrow_\tau{}^* \bullet \rightarrow_{R'_{OBS}}{}^*$ performs expansions by the axioms T2-T3 on P" and, as soon as possible, deletes the unnecessary summands of P". The relation $\rightarrow_{R'_{OBS}}$ performs reductions according to R_{OBS} by a redex selection criterion that prevents those reductions which are exactly opposite to the previous expansions by $\rightarrow_\tau$.

Rewriting steps according to $\rightarrow_\tau{}^* \bullet \rightarrow_{R'_{OBS}}{}^*$ are repeatedly applied until there are summands to delete; they return the term P"'. Next, $\rightarrow_{R''_{OBS}}$ rewrites P"' into P' by applying those reductions which are exactly opposite to the previous expansions; it makes use of a specific redex selection criterion selecting the smallest redex according to the fixed term ordering.

Let us now consider the correspondence between our strategy and the absorption lemma. First, we note that the strategy is *sound*: every rewriting step consists in the application of an axiom of OBS, thus it preserves the observational congruence among terms. Moreover, it can be proved [IN89] that, if the strategy with input P returns P', then P' is an obs-normal form of P (*correctness*) and, if P has an obs-normal form P', then the strategy with input P returns P' or a sumcongruent term (*completeness*).

4. Using the system: an example

Now, let us show a possible use of the system by proving observational equivalence of two CCS specifications. We will take advantage of two different verification techniques, namely the built-in verification tactic and a user directed strategy.

We consider the *scheduling problem* explained in [Mil80]. The problem is the following. Let p1 and p2 be two processes performing a certain task repeatedly. We want to design a scheduler to ensure that p1 and p2 perform the task in rotation, starting from p1. The two processes communicate with the scheduler sch by requesting initiation with the actions a1 and a2 and signaling completion with the actions b1 and b2. The specification of sch which guarantees that p1 and p2 begin their task in rotation starting from p1 is the following:

(~a1.~a2)*

Now, the scheduler sch can be implemented by linking an agent s, starting the scheduler, and two components c1, c2, called cyclers, having the same behaviour and communicating with the processes for initiation and completion of their task. These components synchronize by means of the internal channels g1 and g2.

A possible implementation of sch is the following:

sch = (s | c1 | c2)\g1\g2

where the components are defined as follows:

s = ~g1.nil

c1 = g1.~a1.(~b1.~g2.c1 + ~g2.~b1.c1)

c2 = g2.~a2.(~b2.~g1.c2 + ~g1.~b2.c2)

To prove that the implementation of the scheduler satisfies its specification, we have to prove that, given sch and forgetting all synchronizations along the channels b1, b2, the following observational equivalence, denoted by ≈, holds:

(sch | (b1* | b2*))\b1\b2 ≈ (~a1.~a2)* .

Namely, if we let

sch1 = (sch | (b1* | b2*))\b1\b2

we are left to prove

sch1 ≈ ~a1.~a2.sch1. (+)

In [Mil80] the verification of (+) is structured in the following steps.

The first one rewrites sch1 into an equivalent agent by means of some properties of CCS operators

sch1 ≈ (s | (c1 | b1*)\b1 | (c2 | b2*)\b2)\g1\g2 . (1)

The next step consists of introducing two new agents:

c1p = (c1 | b1*)\b1 c2p = (c2 | b2*)\b2

and of determining that

sch1 ≈ (s | c1p | c2p)\g1\g2 .

Two additional steps permit verifying (+).

The first one consists of proving that c1p and c2p are observational equivalent to their unfoldings; for c1p, this means proving

c1p ≈ g1.~a1.~g2.c1p . (2)

The last step proves (+) taking into account the equivalences established by the previous steps.

In our system we will implement Milner's tactic by proving (1) above by means of a semi-automatic strategy and (2) and (+) by means of the fully automatic strategy of the Verification module.

Semi-automatic Verification

The semi-automatic strategy is totally and explicitly driven by the user which tells Analyzer-CCS the command to execute at each step. In the proof of (1) we make use of some axioms which can be easily derived from the basic ones of observational equivalence.
Let X, Y be CCS agents, $\alpha, \beta \in \Delta$ and $\mu \in \Delta \cup \sim\Delta \cup \{\tau\}$; the axioms we add are the following:

Ax1.	$(X \mid Y)\backslash \alpha = X\backslash\alpha \mid Y\backslash\alpha$	if for each action α of X, $\sim\alpha$ is not an action of Y and conversely
Ax2.	$X\backslash\alpha = X$	if for each action μ of X $\mu \notin \{\alpha, \sim\alpha\}$
Ax3.	$(X\backslash\alpha) \mid Y = (X \mid Y)\backslash\alpha$	(derived from Ax1. and Ax2.)
Ax4.	$X\backslash\alpha\backslash\beta = X\backslash\beta\backslash\alpha$	
Ax5.	$(X \mid Y) \mid Z = X \mid (Y \mid Z)$	
Ax6.	$(X \mid Y) \mid (Z \mid W) = (X \mid Z) \mid (Y \mid W)$.	

The interaction between the user (bold typed) and the system (plain text), sometimes interrupted by few comments (written in italic), is reported below. The command "cmd -> t" has the effect of writing the result of cmd in the variable t which can then be used in future steps.

Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
/ The environment with the definitions of sch1, sch, b1*, b2*, s, c1 and c2 is typed in */*
| **def(sch1).**
Enter a process : **(sch | b1* | b2*)\b1\b2**
Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
| **def(sch).**
Enter a process : **(s | c1 | c2)\g1\g2**
....
Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
/ The hidden actions g1, g2 are pushed outside the parallel operator by Ax3; the result is in t2 */*
| **apply(Ax3, sch1) -> t1.**
Trying to apply axiom ...
....
((s | (c1 | c2)) | (b1* | b2*))\g1\g2\b1\b2
Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
/ Associativity and commutativity axioms of the parallel operator are repeatedly used */*
| **apply_sub(Ax5, t2) -> t3.**
To which subterm? **(s | (c1 | c2)) | (b1* | b2*)**
Trying to apply axiom ...
(s | ((c1 | c2) | (b1* | b2*)))\g1\g2\b1\b2
Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
| **apply_sub(Ax6, t3) -> t4.**

```
    To which subterm? (c1 | c2) | (b1* | b2*)
    Trying to apply axiom ...
    (s | ((c1 | b1*) | (c2 | b2*)))\g1\g2\b1\b2
    Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
/* The actions b1, b2 and g1, g2 are swapped by applying Ax4 four times; the result is in t8 */
    | apply_sub(Ax4, t4) -> t5.
    To which subterm? (s | ((c1 | b1*) | (c2 | b2*)))\g1\g2\b1
    Trying to apply axiom ...
    ..........
    (s | ((c1 | b1*) | (c2 | b2*)))\b1\b2\g1\g2
    Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
/* The actions b1, b2 are pushed inside the parallel operator by applying (four times) Ax1.
   All useless restrictions of b1, b2 are deleted via Ax2 */
    | apply_sub(Ax1, t8) -> t9.
    To which subterm? (s | ((c1 | b1*) | (c2 | b2*)))\b1
    Trying to apply axiom ...
    (s\b1 | (((c1 | b1*) | (c2 | b2*))\b1))\b2\g1\g2
    Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
    | apply_sub(Ax2, t9) -> t10.
    To which subterm? s\b1
    Trying to apply axiom ...
    (s | (((c1 | b1*) | (c2 | b2*))\b1))\b2\g1\g2
    Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
    | apply_sub(Ax1, t10) -> t11.
    To which subterm? ((c1 | b1*) | (c2 | b2*))\b1
    Trying to apply axiom ...
    ..........
    (s | (((c1 | b1*)\b1\b2) | ((c2 | b2*)\b2)))\g1\g2
    Enter Analyzer-CCS Option or 'h' for help (terminated with '.') :
    | apply_sub(Ax2, t15) -> t16.
    To which subterm? (c1 | b1*)\b1\b2
    Trying to apply axiom ...
    (s | (((c1 | b1*)\b1) | ((c2 | b2*)\b2)))\g1\g2
```

This permits concluding that (1) holds. To prove the general claim, we will prove (2) and (+) by means of the Verification module .

Automatic Verification

The Verification module proves observational equivalence of processes E1, E2 by transforming E1 into E2. It applies reductions modulo observational equivalence and replaces subexpressions with identifiers, to deal with recursion while preventing infinite rewritings.

To use the Verification module, when prompted by the system, we type the command "verification", type in the environment and the two processes to be proved observationally equivalent. We will type "[]" to tell the system that the whole environment has been input. In the following we report an excerpt of the interaction between the user and the system.

```
/* Proving (2) */
    Enter Analyzer-CCS Option or 'h' for help (terminated with'.') :
    | verification.
/* The environment with the definitions of c1p, c1 and b1* is typed in */
    Enter a process identifier : c1p
    Enter a process : (c1 | b1*)\b1
    .........
    Enter a process identifier : []
/* The processes to be proved observationally equivalent are now input */
    Enter the first process : c1p
    Enter the second process : g1.~a1.~g2.c1p
    Trying to verify observational equivalence ...
/* The following are the intermediate processes into which c1p is rewritten */
    c1p  =  ((c1 | b1*))\b1
    c1p  =  g1.((~a1.(~b1.~g2.c1 + ~g2.~b1.c1) | b1*))\b1
    c1p  =  g1.~a1.(((~b1.~g2.c1 + ~g2.~b1.c1) | b1*))\b1
    c1p  =  g1.~a1.(~g2.((~b1.c1 | b1*))\b1 + tau.((~g2.c1 | b1*))\b1)
    c1p  =  g1.~a1.(~g2.tau.c1p + tau.~g2.c1p)
    The two processes you entered are observational equivalent.

/* Proving (+) */
    Enter Analyzer-CCS Option or 'h' for help (terminated with'.') :
    | verification.
/* The environment with the definitions of sch1, s, c1p, c2p is typed in */
    Enter a process identifier : sch1
    Enter a process : (s | c1p | c2p)\g1\g2
    .........
    Enter a process identifier : []
/* The processes to be proved observationally equivalent are now input */
    Enter the first process : sch1
    Enter the second process : ~a1.~a2.sch1
    Trying to verify observational equivalence ...
    sch1  =  (((s | (c1p | c2p)))\g1)\g2
    sch1  =  tau.(((nil | (~a1.~g2.c1p | c2p)))\g1)\g2
    sch1  =  tau.~a1.(((nil | (~g2.c1p |c2p)))\g1)\g2
    sch1  =  tau.~a1.tau.(((nil | (c1p | ~a2.~g1.c2p)))\g1)\g2
    sch1  =  tau.~a1.tau.~a2.(((nil | (c1p | ~g1.c2p)))\g1)\g2
    sch1  =  tau.~a1.tau.~a2.sch1
    The two processes you entered are observational equivalent.
```

5. Conclusions and future work

We have presented a term rewriting approach to verify behavioural equivalences between CCS specifications. This approach may be the basis for developing an *environment of theories*, which the user can use at the metalevel by developing his own proof strategies. This environment makes it possible to move from a theory to another and manipulating specifications by their symbolic representation.

We have also seen that this approach has certain weakness and deficiencies, see for example the divergence of completion and non terminating rewritings. Nevertheless, we have shown that, within the framework of the term rewriting approach itself, it is possible to recover such deficiencies. Both the Verification module and the strategy $\rightarrow_{strat}$ that we have defined for the verification of the observational congruence are examples of a definition of proof strategies in an environment of theories. Hence, we have been able to deal with the deficiencies mentioned in an "ad hoc" way.

However, within the framework of term rewriting, there are other techniques (both consolidated and "ad hoc"), which can be investigated in order to cope with the problem of the divergence of completion. One of these is based on *meta-rules* and *meta-rewriting* [Kir87]; another technique deals with a different form of completion guided by the structure of the two terms whose equivalence is "queried". In particular, as far as the observational congruence is concerned, a verification technique based on a *lazy completion* [Her89b] appears very promising. This technique relies on the observation that the left hand sides of the infinite rules generated during the completion process do increase monotonically according to a fixed ordering. Thus, when dealing with a set of processes P, a lazy completion driven by P can be performed by completing the theory only until a rewrite rule, which is greater than (and thus can never be applied to) any term in P, has been generated.

Our system can be extended by adding new modules to implement the canonical term rewriting systems associated to the trace equivalence, the observational equivalence for a subcalculus of CCS, which does not contain the parallel composition operator, and the branching bisimulation. At present, we are working on the implementation of the module for $\rightarrow_{strat}$ and on some extensions. We will introduce axioms for the recursion operator and study a proof strategy for verifying testing congruence via its axiomatic characterization. In fact, the strategies developed for the observational congruence do not straightforwardly apply to testing congruence. One of the main difficulties is represented by the normal forms of this congruence: completeness of its axiomatization has been proved by means of structured and balanced, but not minimal, normal forms, whereas the normal form according to a rewriting relation is always a minimal form.

A further step in our work will be the definition of a *meta-environment* and a *meta-language* which will allow the user to define and use his own proof strategies by applying different theories in a modular and flexible way.

Acknowledgments

We should like to thank Adriana Scalera for implementing part of the system and Miki Hermann for many helpful discussions about divergence of completion.

Appendix: A Calculus of Communicating Systems - CCS

We briefly introduce the relevant definitions of Milner's Calculus of Communicating Systems [Mil80, Mil85]. The concrete syntax of "pure" CCS, i.e. without value passing, is as follows:

$E ::= NIL \mid \mu.E \mid E\backslash\alpha \mid E[\phi] \mid E+E \mid E|E \mid x \mid rec\ x.\ E$

where x is a variable; $\Delta = \{\alpha,\beta,\gamma,...\}$; $\sim\Delta = \{\sim\alpha \mid \alpha\in\Delta\}$; $\tau\notin\Delta$; $\Delta \cup \sim\Delta \cup \{\tau\}$ is the set of *basic actions*, ranged over by μ; $\Lambda = \Delta \cup \sim\Delta$ is ranged over by λ; ϕ is a permutation of $\Lambda \cup \tau$ which preserves τ and the operation ~ of complementation. CCS *agents* are terms generated by the above BNF, the variables of which are bound within a recursive definition (*closed*).

NIL represents an agent which cannot perform any action. μ.E denotes an agent which can only perform action μ and then behaves like E. The actions of **E[ϕ]** are renamings via ϕ of those of E. Agent $E\backslash\alpha$ behaves like E but cannot perform actions α and $\sim\alpha$. Agent $\mathbf{E_1 + E_2}$ can act either as E_1 or as E_2. Agent $\mathbf{E_1|E_2}$ can perform in parallel the actions of E_1 and E_2; moreover they can synchronize, through a τ, whenever they are able to perform complementary actions. Agent **rec x.E** denotes a recursive agent.

The *interleaving operational semantics* is given in terms of labelled interleaving transitions over CCS agents, defined by the following inference rules:

act) $\mu.E \xrightarrow{\mu} E$

res) $E_1 \xrightarrow{\mu} E_2$ *and* $\mu \notin \{\alpha, \sim\alpha\}$ *imply* $E_1\backslash\alpha \xrightarrow{\mu} E_2\backslash\alpha$

rel) $E_1 \xrightarrow{\mu} E_2$ *implies* $E_1[\phi] \xrightarrow{\phi(\mu)} E_2[\phi]$

sum) $E_1 \xrightarrow{\mu} E_2$ *implies* $E_1+E \xrightarrow{\mu} E_2$ *and* $E+E_1 \xrightarrow{\mu} E_2$

com) $E_1 \xrightarrow{\mu} E_2$ *implies* $E_1|E \xrightarrow{\mu} E_2|E$ *and* $E|E_1 \xrightarrow{\mu} E|E_2$
$E_1 \xrightarrow{\lambda} E_2$ *and* $E'_1 \xrightarrow{\sim\lambda} E'_2$ *imply* $E_1|E'_1 \xrightarrow{\tau} E_2|E'_2$

rec) $E_1[rec\ x.E_1/x] \xrightarrow{\mu} E_2$ *implies* $rec\ x.E_1 \xrightarrow{\mu} E_2$.

The axiomatic presentations for finite CCS of the behavioural equivalences we have considered are characterized by a number of common axioms, which we refer to as Basic Axioms, plus a set of τ-laws distinguishing one equivalence from another. It is worth noting that by means of the Basic Axioms any finite CCS term may be proved equal to one containing only NIL, action prefix and summation operators.

Basic Axioms

A1. $X + (Y + Z) = (X + Y) + Z$

A2. $X + Y = Y + X$

A3. $X + X = X$

A4. $X + NIL = X$

A5. $NIL[\phi] = NIL$

A6. $(X + Y)[\phi] = X[\phi] + Y[\phi]$

A7. $(\mu.X)[\phi] = \phi(\mu).X[\phi]$

A8. $NIL\backslash\alpha = NIL$

A9. $(X + Y)\backslash\alpha = X\backslash\alpha + Y\backslash\alpha$

A10. $(\mu.X)\backslash\alpha = \begin{cases} \mu.(X\backslash\alpha) & \mu \notin \{\alpha, \sim\alpha\} \\ NIL & \text{otherwise} \end{cases}$

A11. If $X = \Sigma\,\mu_i.X_i$ and $Y = \Sigma\,\nu_j.Y_j$ then

$$X \mid Y = \Sigma_i\ \mu_i.(X_i \mid Y) + \Sigma_j\ \nu_j.(X \mid Y_j) + \Sigma\ \{\tau.(X_i \mid Y_j) \mid \mu_i = \sim\nu_j\}$$

Trace Congruence [DH84]

S1. $\tau.X = X$

S2. $\mu.(X + Y) = \mu.X + \mu.Y$

Testing Congruence [DH84]

D1. $\tau.(X + Y) + \tau.Y = \tau.Y + X$

D2. $\mu.(\tau.X + \tau.Y) = \mu.X + \mu.Y$

D3. $\tau.(\mu.X + \mu.Y + Z) = \mu.X + \tau.(\mu.Y + Z)$

Observational Congruence [HM85]

T1. $\mu.\tau.X = \mu.X$

T2. $X + \tau.X = \tau.X$

T3. $\mu.(X + \tau.Y) + \mu.Y = \mu.(X + \tau.Y)$

Observational Congruence [HM85]
(without the parallel composition operator)

H1. $X + \tau.X = \tau.X$

H2. $\mu.(X + \tau.Y) = \mu.(X + Y) + \mu.Y$

Branching Bisimulation Congruence [GW89]

B1. $\mu.\tau.X = \mu.X$

B2. $\mu.(\tau.(X + Y) + X) = \mu.(X + Y)$

Obs-Normal Form [HM85]

The notion of normal form according to the observational congruence is reported below:

- P' is a μ-*derivative* of P (and we write $P =\mu\Rightarrow P'$) if $P -\tau^*\rightarrow -\mu\rightarrow -\tau^*\rightarrow P'$.
- Two terms P and Q are *sumcongruent* if P = Q can be proved by using only the AC axioms.
- A term $\Sigma\ \mu_i.P_i$ is a *proper normal form* if
  - i) it does not take the form $\tau.P'$;
  - ii) each P_i is a proper normal form;
  - iii) for $k \neq j$ no μ_k-derivative of $\mu_j.P_j$ is sumcongruent to P_k.
- An *obs-normal form* is either P or $\tau.P$, where P is a proper normal form.

References

[BB87] Bolognesi, T., Brinksma, E. Introduction to the ISO Specification Language LOTOS, Computer Networks and ISDN Systems, **14**, (1987), 25-59.

[BC89] Bolognesi, T., Caneve, M. Squiggles - A Tool for the Analysis of Lotos Specifications, in Formal Description Techniques (K. Turner, ed.), North-Holland, (1989), 201-216.

[BD87] Bachmair, L., Dershowitz, N. Completion for Rewriting modulo a Congruence, Proc. RTA '87, LNCS **256**, (May 1987), 192-203.

[BK84] Bergstra, J.A., Klop, J.W. Process Algebra for Synchronous Communication, Information and Control, **60**, No. 1/3, (1984), 109-137.

[BS87] Bolognesi, T., Smolka, S. A. Fundamental Results for the Verification of Observational Equivalence : A Survey, Proc. IFIP TC6/WG 6.1, (1987).

[CPS88] Cleaveland, R., Parrow, J., Steffen, B. The Concurrency Workbench: Operating Instructions, Tech. Note of Sussex University, (1988).

[DH84] De Nicola, R., Hennessy, M. Testing Equivalences for Processes, Theoret. Comp. Sci. **34**, (1984), 83-133.

[GIN88] Gnesi, S., Inverardi, P., Nesi, M. A logic-functional approach to the execution of CCS specifications modulo behavioural equivalences, Concurrency 88, LNCS **335**, 181-196.

[GLZ89] Godskesen, J.C., Larsen, K.G., Zeeberg, M. TAV Users Manual, Internal Report, Aalborg University Center, Denmark, (1989).

[GW89] van Glabbeek, R.J., Weijland, W.P. Branching Time and Abstraction in Bisimulation Semantics, Proc. IFIP 11th World Computer Congress, San Francisco, (1989).

[Her89a] Hermann, M. Crossed Term Rewriting Systems, Research Report 89-R-003, Centre de Recherche en Informatique de Nancy, (1989).

[Her89b] Hermann, M. Lazy Completion of Term Rewriting Systems (draft), Centre de Recherche en Informatique de Nancy, (1989).

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

[HO80] Huet, G., Oppen, D.C. Equations and Rewrite Rules : A Survey, In "Formal Language Theory: Perspectives and Open Problems", Book R.V.(ed.), A.P. (1980), 349-405.

[Hue80] Huet, G. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, Journal of ACM, **27**, No. 4, (October 1980), 797-821.

[IN89] Inverardi, P., Nesi, M. A Rewriting Strategy to Verify Observational Congruence, I.E.I. Internal Report n° B4-38, (August 1989).

[JK86] Jouannaud, J.P., Kirchner, H. Completion of a Set of Rules modulo a Set of Equations, SIAM J. Comput., **15**, No. 4, (1986), 1155-1194.

[Kir87] Kirchner, H. Schematization of Infinite Sets of Rewrite Rules. Application to the Divergence of Completion Processes, Proc. RTA '87, LNCS **256**, (1987), 180-191.

[KS83] Kanellakis, P.C., Smolka, S.A. CCS Expressions, Finite State Processes and Three Problems of Equivalence, Dept. Comp. Sci., Brown University, (1983).

[Les83] Lescanne, P. Computer Experiments with the REVE Term Rewriting System Generator, Proc. 10th ACM POPL Symposium, Austin, Texas, (1983), 99-108.

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

[Mil85] Milner, R. Lectures on a Calculus for Communicating Systems, NATO ASI Series, Vol. F14, Control Flow and Data Flow: Concepts of Distributed Programming, (1985), 205-228.

[Nes88] Nesi, M. Un approccio logico-funzionale all'esecuzione di linguaggi di specifica concorrenti (CCS) modulo equivalenze comportamentali, Tesi di Laurea, Univ. Pisa, (1988).

[Plo81] Plotkin, G. D. A Structural Approach to Operational Semantics, DAIMI FN-19, Computer Science Department, Aarhus University, Denmark, (September 1981).

[PS81] Peterson, G.E., Stickel, M.E. Complete Sets of Reductions for Some Equational Theories, Journal of ACM, **28**, No. 2, (April 1981), 233-264.

[Sca88] Scalera, A. Uno strumento per la verifica di equivalenze fra agenti concorrenti e comunicanti, Tesi di Laurea, Univ. Pisa, (1988).

[Ver86] Vergamini, D. Verification by means of observational equivalence on automata, Rapports de Recherche, INRIA n° 501, (1986).