Specification and Verification of TCSP Systems by Means of Partial Abstract Data Types

Ricardo Peña
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya. E-08028 Barcelona. Spain
Luis M. Alonso
Departamento de Informática
Universidad del Pais Vasco. E-20080 San Sebastián. Spain

Abstract : A formal framework and a technique for the specification, refinement and correctness proving of parallel systems are presented. Processes are objects in the TCSP model and are specified by means of an auxiliary partial abstract data type. Part of the proofs are made in the abstract data type framework, so more powerful deductive methods can be used. Examples of specifications and of proving a refinement correct are included.

1. Introduction

During the last years, mathematical models for parallel systems such as CCS [Mil 80] and CSP [BHR 84, BrRo 85, Hoa 85] have been proposed giving a formal framework to specification, refinement and verification activities so that they could be accomplished with the necessary rigour. Unfortunately, there is not always an agreement on what the specification of a parallel program should be, neither on which notion of satisfaction of a specification by an implementation is the most appropiate one (e.g. see [Hen 86], [OlHo 86]). Moreover, verification is undecidable in general in the context of those models. For that reason, several methodological proposals have been made, some of them based on the algebraic manipulation of processes (e.g. [HoJi 85]), and others in gradually transforming the specification into a correct implementation (e.g. [Old 86, Hen 86]).

One of the important needs for proving properties of parallel systems (correctness is just one of them), is to have powerful deductive methods supported as far as possible by automatic tools. The above mentioned models satisfy a rich set of algebraic laws [Old 86, Nico 85] but, due to the undecidability problem, most of the proofs have to be made mainly by hand and, in many cases, by finding "eureka" lemmas that solve the difficult steps.

On the other hand, a field that have received most attention in the last decade is the study of models and deductive methods for abstract data types. The models are also algebraic, although they are simpler than those proposed for parallel systems, mainly due to the non existence of infinite objects. According to this simplicity, there are more decidable questions, more algorithms and more tools. In recent years, the so-called rewriting laboratories and, in general, many tools for algebraic theorem proving, have arisen (e.g. Reve, ERIL, RAP, CEC).

In this paper, we present a formal framework and a technique for the specification, implementation and verification of parallel systems trying to bring together the advantages of both fields. The formal framework is borrowed from two sources: the parallel model proposed in [BHR 84, BrRo 85, Hoa 85] that, in the rest of the paper, will be referred to as TCSP, and the algebraic models of abstract data types, in particular, those called *partial abstract types* as developed in [BrWi 82, BMPW 86] that we will refer to as PAT. The technique consists of a restricted way of specifying a process, by means of a system of mutually recursive TCSP equations, that we will call the *normal form* of the process. Process equality will then be proved by comparing their normal forms. This idea already appears in [HoJi 85]. The differences here are mainly two: in one hand, the index set that subindexes the variables of the TCSP equations, is specified by means of a PAT. We choose for this index set the most general one: the process traces. Moreover, this set of traces will have a non monomorphic semantics. Its non isomorphic models represent different process *implementations*, with different number of internal states. On the other hand, to prove process equality we do not require identical sets of recursive equations, as in [HoJi 85], but a weaker relation that turns out to be an implementation relation between two PAT's. Besides that, we show how part of the correctness proof and the proving of other properties, can be made entirely in the PAT framework, so taking advantage of the more powerful deductive aids in this field.

The technique is hierarchical because the design and verification tasks are accomplished *a refinement* at a time. By *a refinement* we mean the decomposition of a process S , representing the specification of a subsystem, into a net N of processes $N_1,\ldots,N_r$ composed by the operator ||. If I are the internal events of the net, that is, the synchronization or communication events between the net components, the correctnes of the refinement consists of proving:

$$(N_1 || \ldots || N_r) \setminus I \ \underline{\mathbf{sat}}\ S$$

where \ is the *hide* operator. In this work we choose, for simplicity and compositionality reasons, the sat relation to be the estrict equality. This implies that all valid implementations have to be as deterministic as the specification S (other satifaction notions allow the implementation to be more deterministic than the specification, e.g. [Old 86]).

The organization of the paper is as follows: Section 2 summarizes the concepts about partial algebras that will be used in the rest of the paper. In section 3 we define the *normal form* of a process to be an infinite set of mutually recursive equations with the index set being the traces of the process. The semantics of this construction is given first, in terms of the PAT models to interpret the set of traces and then, in terms of the TCSP model to define the process. The partial algebra approach gives a non monomorphic semantics to a specification. It is shown that the denoted TCSP process is independent of the concrete model chosen for it. It is also shown that the normal form is expressive enough to denote any non divergent TCSP process. Section 4 is dedicated to explain how safety and liveness properties of a process, including deadlock freedom, can be proved using only deduction in the PAT framework. It is shown that, if there exists an implementation relation between two PAT's then they both denote the same TCSP process, provided that the rest of the normal form is identical.

In section 5 we give the laws to combine processes in normal form, using operators || and \ . While, in the first case, the normal form of the result can be, with some restrictions, mechanically obtained, it does not happen so with the hiding operator, which introduces several problems. An attempt to overcome them is done in section 6. It presents a method for verifying refinements consisting, in essence, of algebraically manipulating the net after hiding and trying to reduce it to normal form. This task needs to use both the laws of the TCSP model and the laws deductible from the PAT specification. It is illustrated with a small example which is thoroughly developed in an appendix.

2. Partial Algebras

In this section we summarize the main partial abstract type concepts we will use later on. A complete description can be found in [BrWi 82, BMPW 86, BrKr 86, GrBr 87].

Given a signature $\Sigma = (S, F)$ with $F = \{F_{w,s}\}_{w\in S^*,\ s\in S}$ the set of operation symbols, a **partial Σ-algebra** A (from now on, a Σ-algebra) is a pair $((s^A)_{s\in S}, (f^A)_{f\in F})$ such that,

- $(s^A)_{s\in S}$ is a family of carrier sets having one carrier for every sort $s\in S$.
- $(f^A)_{f\in F}$ is a family of partial mappings of the form $f^A : s1^A \times\ldots\times sn^A \rightarrow s^A$ for every $f \in F_{s1 \ldots sn,s}$ i.e. $f^A(a_1,\ldots,a_n)$ can be undefined for some tuples $(a_1,\ldots,a_n)$.

By T_Σ we denote the set of **ground terms**, by $T_\Sigma(X)$ the set of **terms with variables** of $X = (X_s)_{s\in S}$ and by t^A the **evaluation** of a ground term in A. Let us observe that for certain $t \in T_\Sigma$, t^A may be undefined. A Σ-algebra is **finitely generated** (term generated) if the evaluation of terms is surjective .

Given two Σ-algebras A and B a **weak Σ-homomorphism** $h : A \rightarrow B$ is a family of partial mappings $\{h_s : s^A \rightarrow s^B\}_{s\in S}$ such that $h_s(f^A(a_1, ..., a_n)) = f^B(h_{s1}(a_1), ..., h_{sn}(a_n))$. The symbol "=" stands for **strong equality** i.e. two values a1 and a2 in A are equal if either *both* are defined and equal, or *both* are undefined. Let us note that, because h is partial, f^A can be defined on more terms than f^B. If h is total, then f^B can be more defined than f^A. It is called a **total** homomorphism. An homomorphism both weak and total is called **strong**. In this case, f^A and f^B are defined on the same terms.

Given a subsignature of Σ, $\Sigma' \subseteq \Sigma$ (i.e. $S' \subseteq S$ and $F' \subseteq F$) a Σ'-algebra A is a **Σ'-subalgebra** of the Σ-algebra B if : (1) for every s in S' $s^A \subseteq s^B$and (2) for every f in F', $f^A = f^B|_A$, where $f^B|_A$ denotes the restriction of f^B to the carriers s^A and "=" is strong equality.

A **partial abstract type specification** (PAT) is a 3-tuple SP = (Σ, E, P) where P = (Σ', E'), $\Sigma' \subseteq \Sigma$ and E, E' are sets of axioms with $E' \subseteq E$. P is called the specification of the **primitive type**. It is required for P to have initial model and that booleans are included in P. An axiom e in E has the form:

$$F_1 \wedge ... \wedge F_n \Rightarrow G_1 \wedge ... \wedge G_m \ , \ n \geq 0 \ , m > 0$$

where F_i, G_i are atomic formulas of the form def(t) or $t_1 = t_2$ with $t, t_1, t_2 \in T_\Sigma(X)$. The satisfaction of an atomic formula by a Σ-algebra A is as follows:

- $A \models def(t)$ if $\overline{\sigma}(t)$ is defined in A
- $A \models t_1 = t_2$ if $\overline{\sigma}(t_1) = \overline{\sigma}(t_2)$

for every $\sigma : X \rightarrow A$, interpreting the symbol "=" as strong equality.

A Σ-algebra A **satisfies** an axiom , $A \models (F_1 \wedge ... \wedge F_n \Rightarrow G_1 \wedge ... \wedge G_m)$ if, whenever $A \models F_i$ holds for every i=1...n, then $A \models G_i$ holds for every i=1...m. Given a PAT specification SP = (Σ, E, P) a partial Σ-algebra A is a **model** of SP if:

a) A is finitely generated
b) A preserves the hierarchy i.e. the Σ'-reduct of A must be isomorphic to the initial model of P = (Σ', E'). Moreover, A satisfies, for the boolean sort, $true^A \neq false^A$
c) A satisfies all the axioms in E

We will call GEN(SP) the class of **all minimally defined models** of SP. Minimally defined models in a class C, are those models A such that, for all $t \in T_\Sigma$, $A \models def(t)$ if and only if $C \models def(t)$ i.e. a term is either defined or undefined in *all* models (using the [BrWi 82] terminology, GEN(SP) = MDEF(PGEN(SP))). In general, models in GEN(SP) are not isomorphic except for their boolean reducts. However, the *observable behaviour* for terms of primitive sort is identical in all models.

In contrast with total algebras, in GEN(SP) the initial and/or final models could not exist. In [BrWi 82] sufficient conditions for the existence of those algebras are given. Specifically, for the existence of initial model in a category with total homomorphisms, it is required for the PAT to be *hierarchy-consistent* and *partial complete* w.r.t the primitive type, and *consistent* w.r.t to the boolean subspecification. A PAT specification with these properties will be called **conservative**. As GEN(SP) contains only minimally defined models, total homomorphisms are also strong.

An atomic formula e is **deducible** from a specification SP, denoted by SP $\vdash$ e, if e is deducible from the axioms of E using the first order logic inference rules adapted to partial algebras [BrWi 82, WiBr 81]. If SP is *partial complete* w.r.t. the primitive type then deduction is complete for ground equations i.e. GEN(SP) $\models t_1 = t_2$ if and only if SP $\vdash t_1 = t_2$ for every t_1, t_2 in T_Σ. It is not complete in the case of theorems with variables i.e. GEN(SP) $\models t_1 = t_2$ if SP $\vdash t_1 = t_2$ for every t_1, t_2 in $T_\Sigma(X)$. Theorems valid in GEN(SP) but not deducible are called *inductive*. Inductive theorems can be proved by structural induction on terms.

We will need also the algebraic implementation concept developed in [BMPW 86] which we restrict to strong homomorphisms: Let SP = (Σ, E, P) and $SP^+ = (\Sigma^+, E^+, P)$ be two specifications with identical primitive type $P = (\Sigma', E')$ such that $\Sigma \subseteq \Sigma^+$ and $GEN(SP^+) \neq \emptyset$. The specification SP^+ is an **algebraic implementation** of SP if for every Σ^+-algebra A^+ in $GEN(SP^+)$ there exists a strong homomorphism from its finitely generated Σ-subalgebra A, to some Σ-algebra B in GEN(SP). Let us consider some interesting remarks about this definition:

1) When defining an implementation, first of all, the implementing type must be enriched with the operations of the implemented type obtaining Σ^+ such that $\Sigma \subseteq \Sigma^+$
2) Considering for every A^+ in $GEN(SP^+)$ its finitely generated Σ-subalgebra A is equivalent, not only to forgetting sorts and operations not present in the implemented type SP, but also the objects of A^+ which are not implementation of objects of SP (they are not the evaluation of any Σ- ground term). The objects of s^{A+} not in s^A are usually called "junk".
3) Every model A^+ of SP^+ has to implement some model B of SP, but not necessarily the opposite direction. So, the definition is liberal enough to enable a reasonable choice of SP^+ in contrast to other implementation notions, e.g. [EKMP 82], where to implement a concrete model of SP is required.
4) The strong homomorphism from A to B induces a congruence on A whose meaning is the identification of objects in A which implement the same object in B.

3. Process Specifications

In this section we first define the syntax of a process in normal form and then its associated semantics. Two examples of process specifications illustrate the definitions.

Definition 1 (syntax)

Let A be an alphabet representing the set of observable events of a process. A **process specification in normal form** is a pair SPP = (M, SP) where:

a) M is a family of non empty subsets of A, $M \subseteq \mathcal{P}_f(A)$, called the *menu*. It is required for all the unitary sets to be pairwise disjoint. To avoid an explicit mention to A in SPP, it is assumed $A = \bigcup_{m\in M} m$.
b) SP is a conservative PAT specification whose primitive type consists of the usual boolean specification and of that of the sort *event*. The carrier for that sort must be isomorphic to A in all $B \in GEN(SP)$. There exists in SP a distinguished sort *trace* with, at least, the following operation symbols:

<> :		$\rightarrow$ trace		{ empty trace }
_^_ :	trace event	$\rightarrow$ trace		{ register an event into a trace }
G_m :	trace	$\rightarrow$ bool	, $m \in M$	{ boolean guard for option m of the menu }

The following axioms, which specify the definedness predicate on traces, will be assumed to be included in SP although not explictly written:

$\forall$ s:trace
def (<>)
$\{ \text{def}(s) \wedge G_m(s) \Rightarrow \bigwedge_{e\in m} \text{def}(s\text{^}e) \}_{m\in M}$ ♦

The sets of the menu M, are supposed to be in deterministic choice and, inside each set, the events are in nondeterministic choice. Assuming $M = \{m_1,\ldots,m_p\}$,we will usually write a specification SPP with the following syntax:

```
spec name
        ...
        Signature and equations of SP with the usual syntax
        ...
        process S (s) Δ
            [   G_1(s) ⇒  ⊓  e
                         e∈m_1

            []  ...
            []  G_p(s) ⇒  ⊓  e
                         e ∈ m_p
            ]
end spec
```

The boolean guards $G_m(s)$ determine, for each trace *s*, the possible and/or mandatory events for the process after that trace. The axioms for the *def* predicate establishes that only possible traces are defined.

Example 1: a mutual exclusion system

We wish to specify a system of two "users", each one consisting of an infinite loop of the sequence $o_i \to cr_i \to fr_i \to \ldots$, i=1,2, where events o_i mean "other things", cr_i mean "begining of critical region", and fr_i, "end of critical region". Before o_i, a user is in state *t*, for "thinking"; before cr_i it is in state *h*, for "hungry"; and between cr_i and fr_i, the state is *e*, for "eating". The specification must forbid a state in which both users are simultaneously eating. Also, if both users are hungry, is not determined which one will succeed in entering the critical region. The specification of this system is as follows:

```
spec mutual_exclusion
   sorts trace, event, state
   ops    <>              :                 → trace
          o_i, cr_i, fr_i  :                 → event              {i = 1,2}
          _^_             : trace event     → trace
          st_i            : trace           → state              {i = 1,2}
          _eq _           : state state     → bool
          t, h, e         :                 → state
          A_i , E_i       : trace           → bool               {i = 1,2}
          B, C, D         : trace           → bool
   eqns   ∀ s:trace
          st_i (<>) = t                                          {i = 1,2}
          st_i (s^o_j) = if eq(i,j) then h else st_i (s)         {i,j = 1,2}
          st_i (s^cr_j) = if eq(i,j) then e else st_i (s)        {i,j = 1,2}
          st_i (s^fr_j) = if eq(i,j) then t  else st_i (s)       {i,j = 1,2}
          A_i (s) =  st_i (s) eq t                               {i = 1,2}
          B (s) = st_1(s) eq h ∧ st_2 (s) eq t
          C (s) = st_2(s) eq h ∧ st_1 (s) eq t
          D (s) = st_1(s) eq h ∧ st_2 (s) eq h
          E_i (s) = st_i (s) eq e                                {i = 1,2}
          ... equations for "eq"...
   process  S (s) Δ [
          i[] A_i (s)     ⇒ o_i
          [] B (s)        ⇒ cr_1
          [] C (s)        ⇒ cr_2
          [] D (s)        ⇒ cr_1 ⊓ cr_2
          i[] E_i (s)     ⇒ fr_i
                          ]
end  spec                                                        ♦
```

Now, we give the semantics of this construction in two steps: first we interpret the SP part of SPP in terms of PAT models. Then, we define the semantics of the **process** part in terms of the TCSP model. There are at least two reasons for adopting partial algebras as models of SP:

a) not every trace generated by the constructor operations <> and $\_\hat{}\_$, is a possible or valid process trace.
b) the operations which determine the possible events after a given trace *s* are the boolean consultors of the sort *trace* appearing in guards. There exist many admissible models for the carrier of sort *trace* with

identical behaviour with respect to those consultor operations.

Fact (a) suggests considering the constructor operation _^_ as partially defined so that only valid traces are constructed. The *def* predicate of partial algebras, excludes invalid traces from all the models. Fact (b) suggests adopting a polymorphic semantics for SP, as distinct models could present the same behaviour with respect to the boolean consultors. We will define the semantics of SPP in terms of the SP initial model and then generalize the definition for any model by proving that the denoted process is independent of the choice.

Definition 2 (semantics)

Let SPP = (*M*, SP) with $M = \{m_1, ..., m_p\}$be a process specification in normal form. Its semantics is given by:

a) The semantics of SP is GEN(SP). Let I be the initial model and $trace^I$ be the carrier in I for the sort trace.
b) The semantics of SPP is the component $X_{<>}$ of the unique least fix point of the following system of mutually recursive equations :

$$\mathbf{X} = \mathbf{F}(\mathbf{X})$$

where $\mathbf{X} = <X_s>_{s\in trace^I}$ is a family of TCSP process variables and $\mathbf{F} = < F_s>_{s\in trace^I}$ is a family of functions each one taking as argument a $trace^I$-indexed family of processes, and giving a process as a result. For each $s\in trace^I$, the function F_s is defined by:

$$F_s(X) = \underset{i\in J_s}{[]} \Big(\underset{e\in m_i}{\sqcap} (e \rightarrow X_{s\wedge e})\Big)$$

where $J_s = \{ i \in \mathbf{N} \mid i \in \{1,...,p\} \wedge G_i^I(s) = \text{true} \}$ ♦

J_s is the set of indexes of the menu *M* such that the guards $\{G_i^I(s)\}_{i\in J_s}$ evaluate to *true* in I. If $J_s = \varnothing$, the above expression means the blocked process stop. If an m_i is unitary, then the $\sqcap$ operator disappears of the corresponding branch. [] is the deterministic choice, $\sqcap$ is the nondeterministic choice, and $\rightarrow$ is the prefix, all of them TCSP operators.

The intuition behind this normal form definition is that, when a process begins to "execute", its history of events is the empty trace <>. Then, at each "iteration", it offers a choice of sets of events based on its whole past history *s*. The event finally done *e*, is a combination of external and internal decissions and the rest of the process is a new iteration of the normal form where the history has been modified to register the event *e*.

Let us note that it is not possible to represent in this way a divergent process, as all the variables of the equation system are prefixed by one event. Comparing this with other TCSP process normal forms, [HoJi 85, Nico 85], there the order of choices is, first, a nondeterministic one between sets of events and then a deterministic one between the events of each set. The family of sets of the first choice must be *saturated.* Each process expressed in that form can be translated into our form by applying the distributive laws of choice operators at the price, may be, of some redundancy. We give a small example to show the translation process:

$$((a\rightarrow A)\ []\ (b\rightarrow B)) \sqcap (c\rightarrow C) = ((a\rightarrow A) \sqcap (c\rightarrow C))\ []\ ((b\rightarrow B) \sqcap (c\rightarrow C))$$

It has been shown [Nico 85], that the four operators $\sqcap$, [] ,$\rightarrow$ y stop, plus recursion, are enough to represent *any* non divergent process. So, the strategy with derived operators, such as || and\, will be to try to eliminate them from the process specification.

Proposition 1

A process specification SPP = (*M*, SP) defines a unique TCSP process independently of the model chosen for SP.

Proof

Let A and A' be in GEN(SP) and be $s \in T_{\Sigma,\ trace}$. If GEN(SP) $\not\models$ def(s) then, s^A and $s^{A'}$ are both undefined. In both cases the process defined by SPP is stop, as none guard evaluates to *true*.

If s is defined, let $b[y] \in T_\Sigma(X)$ be a boolean context with y a variable in X_{trace}. All the boolean reducts of the models of SP are isomorphic to {true, false } as this is the unique finitely generated model of *bool* satisfying *true* $\neq$ *false*. By the partial completeness of SP w.r.t. the sort *bool* , either SP |- b[s] = *true*, or SP |- b[s] = *false*, or $\neg$(SP |- def(b[s])) holds, so $b[s]^A = true^A$ if and only if $b[s]^{A'} = true^{A'}$. Therefore, the set of open guards, and so, the set of events *e* admissible for X_s are identical in both models. The continuation expressions $X_{s^\wedge e}$ have syntactically identical subindices, so the argument on X_s can be extended to any variable of the equation system and then to the least fix point. Hence, the process associated to every term s, in particular to s = <>, is independent of the models of SP. ♦

This proposition allows us to generalize *definition 2* and to take any model $B \in$ GEN(SP) as the set of subindices of the equation system.

Example 2: a boolean semaphore

We specify a semaphore for two users, with two sets of events $\{p_i, v_i\}_{i=1,2}$, one for synchronizing with each user.

```
spec semaphore
    sorts trace, event
    ops    <>          :              → trace
           p_i, v_i    :              → event              {i = 1,2}
           free        : trace        → bool
    eqns   ∀ s:trace
           free (<>) = true
           free (s^p_i ) = false                            {i = 1,2}
           free (s^v_i ) = true                             {i = 1,2}
    process SEM (s) Δ [
           i[] free (s)     ⇒ p_i
           i[] ¬ free(s)    ⇒ v_i
                              ]
end spec
```

The SP semantics is the class GEN(SP) . The initial model $trace^I$ will contain in the carrier for *trace* all the valid traces (in the usual sense of a trace as a sequence of events) and all of them will be different:

$$trace^I = \{<>, <p_i>, <p_i v_j>, <p_i v_j p_k>, \dots\}$$

The final model $trace^F$ will have the least number of histories which are distinguishable by the boolean consultors. In this case, it is only necessary to keep two valid histories: the one which has an even number of events and the one which has an odd number of them.

$$trace^F = \{even, odd\}$$

$free^F(even) = true^F$	$even\ ^{\wedge F}\ p = odd$	$even\ ^{\wedge F}\ v = undefined$
$free^F(odd) = false^F$	$odd\ ^{\wedge F}\ p = undefined$	$odd\ ^{\wedge F}\ v = even$

By *proposition 1,* both the initial and the final model or any other model in GEN(SP), define the same TCSP process *semaphore.* ♦

4. Proof of properties of parallel processes

Many properties of a TCSP process **P** are provable from the set S of its possible traces. In particular, the so called *safety* properties [OlHo 86] expressing that **P** is not allowed to do any trace which do not belong to S.

These kind of properties can be proved in the partial algebra framework, using the deductive power of algebraic specification , as SP characterizes the set of *all* valid traces of the process. Proving a property is equivalent to proving theorems of the the form $t_1 = t_2$, or def (t), which have to be satisfied by all the models of SP.

In the mutual exclusion system of *example 1* , an operation expressing this property can be defined in the following way:

```
ops     mutex : trace → bool
eqns    ∀ s : trace
        mutex(s) = ¬( st_1(s) eq e ∧ st_2(s) eq e )
```

The theorem needed in order to assure that the process satisfies the property is: GEN(SP) $\models$mutex(s) = *true*, which is an inductive theorem.

Even, some *liveness* properties [OlHo 86] are provable by noting that, in the process specification we have, not only the "past" of the process (its possible traces) but also its "future" in the form of possible events after every trace *s*. The set of these events is completely determined by the set of open guards $G_m(s)$.

In particular, *deadlock freedom* can be expressed as the process capability for participating in some event after *every* possible trace *s*. If $M = \{ m_1,\dots,m_p\}$, this amounts to prove the following algebraic theorem :

$$\text{GEN(SP)} \models G_1(s) \vee \dots \vee G_p(s) = \text{true}$$

This property can be studied, either at the subsystem specification level, or at the net, before hiding, level. If the net does not diverge, both proofs are equivalent [RoDa 86], and the first will always be simpler. More complex liveness properties can also be expressed. For instance, that under condition B(s) some subset of the events will always be offered.

Algebraic deductive methods will also be useful in section 6 where a certain number of transformation rules for processes are given. Most of them need to be based on algebraic theorems satisfiable by SP. That means that one part of the algebraic manipulation of TCSP processes can be carried out in the abstract data type framework and its associated deductive methods.

A specially interesting property is to establish under what conditions two process specifications SPP1 = (*M*, SP1), SPP2 = (*M*, SP2), with identical menu structure, define the same TCSP process. This is equivalent to compare SP1 and SP2 in order to determine if they define the same models or, more generally, if either GEN(SP2) $\subseteq$ GEN(SP1) or GEN(SP1) $\subseteq$ GEN(SP2) holds. We assume implicitly that SP1 and SP2 have the same signature. To illustrate this, let us consider the following example.

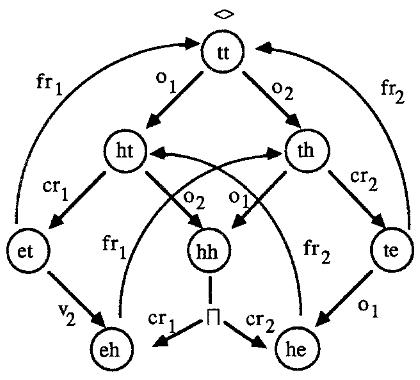
Example 3: another mutual exclusion system

We specify a second mutual exclusion system based on the idea that the minimun information that the system must keep in order to accept or refuse an event, is a pair of states (*t*, *h* or *e*), one for each user. The operation *suc(s)* gives the state next to *s* in circular order (i.e. t, h, e, t,...).

```
spec mutual_exclusion_2
    sorts trace, event, state
    ops     [_,_]          : state state    → trace
            <>             :                → trace
            o_i, cr_i, fr_i :               → event              {i = 1,2}
            _^_            : trace event    → trace
            st_i           : trace          → state              {i = 1,2}
            _eq _          : state state    → bool
            suc            : state          → state
            t, h, e        :                → state
            A_i ,  E_i     : trace          → bool               {i = 1,2}
            B, C, D        : trace          → bool
    eqns    ∀ s_1, s_2 : state, ∀ s:trace
            <> = [t, t]
            [s_1, s_2] ^ o_1 = [s_1, s_2] ^ cr_1 = [s_1, s_2] ^ fr_1 = [suc(s_1), s_2]
            [s_1, s_2] ^ o_2 = [s_1, s_2] ^ cr_2 = [s_1, s_2] ^ fr_2 = [s_1, suc(s_2)]
```

```
        $st_1 ( [s_1, s_2] ) = s_1$
        $st_2 ( [s_1, s_2] ) = s_2$
        $A_i (s) = st_i (s) eq t$                                {i = 1,2}
        $B (s) = st_1(s) eq h \wedge st_2 (s) eq t$
        $C (s) = st_2(s) eq h \wedge st_1 (s) eq t$
        $D (s) = st_1(s) eq h \wedge st_2 (s) eq h$
        $E_i (s) = st_i (s) eq e$                                {i = 1,2}
        ... equations for "eq" and "suc"...
   process S (s) $\Delta$ [
        i[] $A_i (s)$     $\Rightarrow o_i$
        [] B (s)      $\Rightarrow cr_1$
        [] C (s)      $\Rightarrow cr_2$
        [] D (s)      $\Rightarrow cr_1 \sqcap cr_2$
        i[] $E_i (s)$     $\Rightarrow fr_i$
                      ]
end spec
```

It seems obvious that this process is the same as that of *example 1*, even though the set of admissible models for the sort *trace* is much more restrictive in SP2 than in SP1. Forgetting operations [_,_] and *suc* in SP2 to get identical signatures, we have in this example GEN(SP2) $\subseteq$ GEN(SP1). In fact, GEN(SP2) contains only one model, which coincides with the final model of SP1, and whose expression in the form of a transition system can de depicted as follows:



♦

The approach that seems better suited to compare SP1 and SP2 in order to know whether they denote or not the same process, consists of defining an **implementation relation** between them. The specification with less models implements the one with more models, as the intuition behind implementation notions uses to be. The algebraic implementation concept surveyed in section 2 formalizes this idea by establishing that finitely generated Σ-subalgebras A of models A^+ of SP^+, have to be homomorphic to models B of SP. Then, the boolean Σ-terms will be evaluated in the same way, both in A and in B.

Proposition 2

Let SPP = (*M*, SP) and $SPP^+ = (M^+, SP^+)$ two process specifications in normal form. If $M = M^+$ and SP^+ implements SP, then SPP and SPP^+ denotes the same TCSP process.

Proof

Follows from the above considerations and the same arguments than those of the *proposition 1* proof. ♦

The algebraic implementation concept will also be useful in section 6 to formalize the idea of refining a TCSP process into a net of parallel processes. There will appear "junk" values in the implementing type corresponding to net states not reachable through traces of visibles events. Also it may happen that several net states be "confused" in the same visible state.

5. Process composition

To refine a process into a net of processes, we first specify every component, by writing their normal form expressed in terms of its own trace variables. Then, we must combine these normal forms with the parallel operator. For simplicity, we choose the original one ||. We then require the net to be *triple-disjoint* [RoDa 86] i.e. every event is in the alphabet of at most two processes. Under these assumptions, the resulting process is deterministic if and only if every component is so. We will assume this case.

Definition 3

Let SPP1 = (*M1*, SP1) and SPP2 = (*M2*, SP2) be the specifications of two deterministic processes i.e. *M1* and *M2* are formed exclusively by unitary disjoint sets. Let, respectively A1, A2 be the alphabets of the processes and *s1*, *s2* the distinguished sorts of both specifications. We define the normal form SPP = (*M*, SP) **parallel composition** of SPP1 and SPP2 in the following way:

- the alphabet of SPP is $A = A1 \cup A2$
- SP is the unión of SP1 and SP2, including once the primitive type, and renaming the operations of SP1 and SP2 if necessary. Besides that, a new sort *s*, that will be the distinguished sort of SP, will be included, together with the following operations and axioms:
- a tupling operation $<\_ , \_> : s1\ s2 \rightarrow s$, which define the traces of the net, as tuples of traces of the components, with the following axioms:
  $< x, y >\hat{}e = < x\hat{}e, y >$, if $e \in A1$ and $e \notin A2$
  $< x, y >\hat{}e = < x, y\hat{}e >$, if $e \notin A1$ and $e \in A2$
  $< x, y >\hat{}e = < x\hat{}e, y\hat{}e >$, if $e \in A1$ and $e \in A2$
- the menu of SPP is $M = M1 \cup M2$
- for each guard $G_{m1} \in$ SP1 with $m1 = \{e1\}$ and $e1 \notin A2$, a guard operation $G_m : s \rightarrow bool$ is introduced, where $m = \{e1\} \in M$, with axiom:
  $G_m(< x, y >) = G_{m1} (x)$
- simetrical reasoning for guards of SP2
- for each event $\{e\} \in M1 \cap M2$ with guards $G_{m1} \in$ SP1 and $G_{m2} \in$ SP2, a guard operation $G_m : s \rightarrow bool$ is introduced, where $m = \{e\} \in M$, with axiom:
  $G_m(< x, y >) = G_{m1} (x) \wedge G_{m2} (y)$
- the definedness predicate for the new distinguished sort *s*, is specified with the implicit axioms mentioned in *definition 1*, which are assumed to be part of the normal form. (Note in this respect that, not every tuple $< x, y >$ of defined individual traces, is a defined trace of the net). ♦

Proposition 3

The process denoted by SPP, parallel composition of SPP1 and SPP2, is the TCSP process SPP1 || SPP2.

Proof

It follows inmediatly from the TCSP laws for the || operator , in the presence of deterministic components. The common events need the cooperation of both processes so this explain the $\wedge$ composition of the individual guards. The non common events need only to take into account the state of one of the processes. ♦

Example 4: *A net of two users and a semaphore*

We specify a user process with synchronization events $\{p_i, v_i\}$and compose in parallel two of them with the boolean semaphore of *example 2*. For brevity, we present only the **process** part of the resulting normal form. The state of a user is now a natural number from 1 to 5. Operation $\equiv$ is equality on naturals.

```
spec user_i                    {i = 1,2}
   sorts trace, event
   ops    <>                  :                    → trace
          o_i, p_i,cr_i, fr_i ,v_i :               → event
          _^_                 : trace event        → trace
          st_i                : trace              → nat
          A_i,B_i,C_i,D_i, E_i : trace             → bool
   eqns   ∀ s:trace, ∀ e:event
          st_i (<>) = 1
          st_i (s^e) = ( st_i (s) mod 5 ) + 1
          A_i (s) =  st_i (s) ≡ 1
          B_i (s) =  st_i (s) ≡ 2
          C_i (s) =  st_i (s) ≡ 3
          D_i (s) =  st_i (s) ≡ 4
          E_i (s) =  st_i (s) ≡ 5
   process U_i (s) Δ          [
          [] A_i (s)           ⇒ o_i
          [] B_i (s)           ⇒ p_i
          [] C_i (s)           ⇒ cr_i
          [] D_i (s)           ⇒ fr_i
          [] E_i (s)           ⇒ v_i
                               ]
end spec

spec net
   ...
   process R (s) Δ     [
          i[] st_i (s) ≡ 1                    ⇒ o_i
          i[] st_i (s) ≡ 2  ∧ free (s)        ⇒ p_i
          i[] st_i (s) ≡ 3                    ⇒ cr_i
          i[] st_i (s) ≡ 4                    ⇒ fr_i
          i[] st_i (s)  ≡ 5 ∧ ¬free (s)       ⇒ v_i
                        ]
end spec                                                          ♦
```

The next step to compare the net behaviour with the system specification, is to hide all the net internal events. Some problems appearing in attempting to reduce the resulting process to normal form are the following:

- the alphabet of the process is reduced, but the the set of traces must be kept unmodified. If not, the definition of the guard operations will no longer be valid. This implies an inconsistency between the alphabet that comes from the menu, and the carrier of sort *event* from the models of SP.
- after hiding, there will be in the process specification branches with no events i.e. it will no longer be in normal form. This specification would correspond to a system of recursive equations with non guarded variables. Non divergence is not guaranteed as it could be expected.
- if, in a deterministic selection, some of the events but not all, are hidden, following TCSP laws, the nondeterministic operator $\sqcap$ must appear to take into account the possibility that the process chooses to do an internal event. To make appear these terms, we need to know explicitly the conditions i.e. the guards, under which these "mixed" selections will take place.

The above problems imply, among other things, that it will not always be possible to transform a process after hiding into a normal form. In many cases, the difficulty to do that will probably indicate that the net diverges. In the next section, we give a method that obviates, when it is possible, all the above problems and arrives successfully to a process in normal form.

6. Process refinement

The main activity in the development of parallel programs consists of implementing a process S, which represents a subsystem specification, by a net $R=R_1 || \ldots ||R_n$ of synchronized parallel processes. This net will be called *a refinement* of S. Verifying a refinement consists of proving that the net behaviour is contained in the range of non determinism of S, taking only into account the visible events i.e. those which are not internal synchronization events of the net (let us call I these last events). In this work, we are assuming that S exactly specifies the desired degree of non determinism. Therefore, verifying a refinement, consists of proving $S = R \setminus I$. Moreover, we are studying the case that R is deterministic.

The strategy followed here is to try to reduce $R \setminus I$ to its normal form (*M'*, SP'), in terms of a data type SP' which defines the possible traces of the net after hiding and then to compare it with the normal form of S = (*M*, SP), in terms of its own data type SP which defines the traces of the specification. If the same syntactic structure is obtained (more precisely, if $M = M'$) and it is proved that SP' is a correct implementation of SP then, by *proposition 3*, the denoted TCSP processes will be the same.

Problems arise when applying the hiding operator \, as we have seen in section 5. Let us remind some TCSP laws [Hoa 85] explaining how the operator \ interacts with the choice operators $\square$ and $\sqcap$:

$$\begin{array}{ll}
\text{T1)} & ((a \rightarrow P) \,\square\, Q) \setminus \{a\} = (P \setminus a) \sqcap ((P \,\square\, Q) \setminus a) \\
\text{T2)} & ((a \rightarrow P) \,\square\, (b \rightarrow Q)) \setminus \{a, b\} = (P \setminus \{a, b\}) \sqcap (Q \setminus \{a, b\}) \\
\text{T3)} & (P \sqcap Q) \setminus \{a\} = (P \setminus \{a\}) \sqcap (Q \setminus \{a\})
\end{array}$$

Let us consider the following deterministic process in normal form, representing the net R, where $I=\{h_1, ..., h_n\}$ are the events to be hidden:

$$\begin{array}{llll}
R(s)\,\Delta & [& B_1(s) & \Rightarrow e_1 \\
 & & \ldots & \\
 & \square & B_n(s) & \Rightarrow e_n \\
 & \square & BH_1(s) & \Rightarrow h_1 \\
 & & \ldots & \\
 & \square & BH_m(s) & \Rightarrow h_m \;]
\end{array}$$

The attainment of the normal form of $R(s) \setminus \{h_1, ..., h_m\}$ will imply the following phases:

a) guard folding
b) application of the \ operator laws
c) unfolding of the non guarded expressions
d) reduction to normal form
e) implementation relation

In the appendix, the net of *example 4*, formed by two "users" and a boolean semaphore, is reduced to normal form and shown to be the same process as the mutual exclusion system specified in *example 1*. All the transformation steps explained here are followed there, so the reader may wish to consult the appendix while reading what follows.

a) guard folding

The following laws, easily derived from the normal form definition, allow to explicitly introduce the $\square$ operator in branches, to eliminate it if it is explicit, or to group redundant branches:

$$\begin{array}{llllll}
\text{L1)} & [\ \ldots & & & [\ \ldots & \\
 & \square & B_1 \Rightarrow e_1 & & \square & B_1 \Rightarrow e_1 \\
 & \square & B_2 \Rightarrow e_2 & = & \square & B_2 \Rightarrow e_2 \\
 & \ldots & & & \square & B_1 \wedge B_2 \Rightarrow e_1 \,\square\, e_2 \\
 & \ldots &] & & \ldots &] \\
\text{L2)} & [\ \ldots & & & [\ \ldots & \\
 & \square & B \Rightarrow e_1 \,\square\, e_2 & & \square & B \Rightarrow e_1 \\
 & \ldots & & = & \square & B \Rightarrow e_2 \\
 & \ldots &] & & \ldots &] \\
\text{L3)} & [\ \ldots & & & [\ \ldots & \\
 & \square & B \Rightarrow e_1 & & \square & B \vee B' \Rightarrow e_1 \\
 & \square & B' \Rightarrow e_1 & = & \ldots &] \\
 & \ldots &] & & &
\end{array}$$

The first transformation of process R(s) will be the grouping of guards by applying law L1, in such a way that the events to be hidden appear in explicit determinisc choice with every external event and with the other internal events. The resulting process will have the form :

$$
\begin{array}{lll}
R(s)\,\Delta & [& B_1 \Rightarrow e_1 \\
 & [] & B_1 \wedge H_1 \Rightarrow e_1 \, [] \, h_1 \\
 & & \dots \\
 & [] & B_1 \wedge H_m \Rightarrow e_1 \, [] \, h_m \\
 & & \dots \\
 & [] & B_n \wedge H_m \Rightarrow e_n \, [] \, h_m \\
 & [] & H_1 \Rightarrow h_1 \\
 & [] & H_1 \wedge H_2 \Rightarrow h_1 \, [] \, h_2 \\
 & & \dots \\
 & [] & H_{m-1} \wedge H_m \Rightarrow h_{m-1} \, [] \, h_m \;]
\end{array}
$$

Let us note that the original branches $B_i \Rightarrow e_i$ and $H_i \Rightarrow h_i$ are still present, in order to consider the states in which e_i and h_i are offered without offering, at the same time, other internal events. The number of resulting branches is proportional to max (m.n, m^2). Usually, most of them can be eliminated since, from the specification SP, the corresponding guards are provable to be false.

b) application of the \ operator laws

Next, the internal events are hidden by the application of the \ operator TCSP laws T1 to T3. They can be reliably applied as it is explicitly shown, which events the hidden ones are in deterministic choice with. The nondeterminisric operator $\sqcap$ will probably appear in this step. We need to show explicitly the process expressions which are continuation of each hidden event. These expressions will not be prefixed by any event (they will be, in TCSP terminology, *non guarded*). So, the risk for divergence shows up. Let us call RH(s) the net process R(s) after hiding.

c) unfolding of the non guarded expressions

To continue towards the normal form, we must *unfold* the non guarded process expressions, substituting them by their definitions. These consist of applying the process definition RH(s), to traces of the form *s^h*, where *h* is a hidden event. This is done under the certainty of guard $G_m(s)$ that "protects" the occurrence of event *h*.. Under assumption $G_m(s)$, it will be possible to simplify the definitions by proving many guards of the form G_m (s^h) to be false. We continue this unfolding scheme until all process definitions contain no unguarded expressions. Sometimes, we will need induction on the structure of *s* to show that the number of unfoldings is finite. The resulting process is obtained by combining the original process with the definitions of all the unfolded expressions to get a single *global* definition with no unguarded expressions. This is achieved by creating guards G'_m that are conjunctions of guards from different nested definitions. A number of simplifications are possible in this step, which can be done by using deduction in SP'.

d) reduction to normal form

The process obtained in (c) is not in normal form yet, due to two reasons: (1) operator [] is explicit in some branches, and (2) several branches offer the same single event (unitary sets in *M'* would not be disjoint). Problem (1) can be eliminated by applying law L2 i.e. by unfolding branches with explicit operator []. Problem (2) could apparently be eliminated by using law L3, but we find here another problem: continuation expressions after the same *visible* event *e*, will not usually be the same in all the branches which offer *e*. The hidden events, if any, registered previously to the visible one, could be different. The situation can be depicted in this way:

$$
\begin{array}{l}
\dots \\
[]\, B_1(s) \Rightarrow e \rightarrow X_{s\hat{}h1\hat{}\dots\hat{}hn\hat{}e} \\
[]\, B_2(s) \Rightarrow e \rightarrow X_{s\hat{}j1\hat{}\dots\hat{}jm\hat{}e} \\
\dots
\end{array}
$$

where $h_1,\dots, h_n, j_1,\dots, j_m$ are hidden events. The problem can be solved by defining in SP' a new operation _&_ , meaning "register in *s* a visible event", with the following axioms:

$$\begin{array}{lll} \forall s\text{: trace, e: event} & B_1(s) = \mathit{true} & \Rightarrow s\&e = s\hat{}h_1\hat{}\dots\hat{}h_n\hat{}e \\ & B_2(s) = \mathit{true} & \Rightarrow s\&e = s\hat{}j_1\hat{}\dots\hat{}j_m\hat{}e \end{array}$$

then, in the process definition, we substitute the old registering operation _^_, by the new one _&_ , and apply law L3, obtaining:

$$\begin{array}{ll} \dots & \\ [\,]\, B_1(s) \vee B_2(s) \Rightarrow & e \rightarrow X_{s\&e} \\ \dots & \end{array}$$

By repeatedly applying these rules we get finally the normal form for RH(s) = (M', SP$^+$), where SP$^+$ is the original net PAT SP', enriched with the new operation _&_, and with as many boolean consultors, as subsets are in M'. The definition of each consultor G_m, is just an axiom stating that it is identical to the guard of the corresponding branch. This guard will be a combination, through the boolean operators $\wedge$ and $\vee$, of the guards of R (the net before hiding).

e) implementation relation

Now, we have to show that the specification process S = (M, SP) and the net process after hiding RH = (M', SP$^+$), both in normal form, are in fact the same. We consider the _&_ operation of SP$^+$ to have the same name as the _^_ registering operation of SP. We make use of the result of section 4 and state that, S = RH if:

1) $M=M'$ and,
2) SP$^+$ implements SP

Let us note that, if the first condition is not satisfied, the second might have even not sense, as $\Sigma \subseteq \Sigma^+$ could not be true, due to the different number of consultors in both specifications. Let us also note that, if we take a model of SP$^+$, the subalgebra generated by <> and _&_ will not contain, in general, all the values of the model (those generated by <> and _^_ , corresponding to valid traces of the net before hiding). The unreachable values are the "junk" of the implementation. It could also be possible that different reachable values in that model, are confused in the corresponding model of SP. In the appendix example the final model of the implementing specification is depicted and the "junk" and "confusion" phenomena are illustrated.

7. Conclusions

A normal form for the definition of non divergent TCSP processes has been proposed, around the following ideas:

- a process is defined by a system of mutually recursive equations with some syntactical restrictions. The set of subindices is the most general one: the process traces
- the set of traces is specified by means of partial abstract types and several non isomorphic model are possible for it. The process denoted is proved to be independent of the model.

Deductive methods for abstract data types can be used, to proof *any* security property and *some* liveness properties of a process, including deadlock freedom. Also, an important part of a refinement correcness proving (e.g. that SP$^+$ implements SP, and the simplifications during the reduction of R\I to normal form) can be entirely done in the abstract data type framework.

A formal framework has been proposed in which the refinement of a process into a net can be understood as an implementation relation between two abstract data types: the internal traces are an implementation of the external ones. We believe that this idea clarifies the effect of the hide operator on a net of parallel processes.

Based on this framework, a technique for verifying the correctness of a refinement has been developed an illustrated by an example. Up to this moment, it has been applied to simple examples but the hope exists that it can be used in more sophisticated ones, in particular, those with a variable number of processes. Also, large parts of the proofs are expected to be supported by automatic tools.

Acknowledgements

We gratefully acknowledge many and helpful suggestions made by Mª Pilar Nivela and Fernando Orejas to an early version of this paper.

References

[BHR 84] Brookes, S.D.; Hoare, C.A.R.; Roscoe, A.W. *A Theory of Communicating Processes.* Journal of the ACM, Vol. 31, No. 3, July 1984, pp.560-59

[BMPW 86] Broy, M.; Möller, B.; Pepper, P.; Wirsing, M., *Algebraic implementation preserve program correctness.* Science of Computer Programming 7 (1986) 35-37.

[BrKr 86] Broy, M.; Krieg-Brückner, B., *PROSPECTRA training course,* given at Alcatel-SESA, Madrid, dec. 1986

[BrRo 85] Brookes, S.D.; Roscoe, A.W. *An Improved Failures Model for Communicating Processes.* LNCS No. 197, Springer_Verlag 1985, pp. 281-305

[BrWi 82] Broy, M.; Wirsing, M., *Partial abstract types.* Acta Informatica **18** (1) (1982) 47-64.

[EhMa 85] Ehrig, H.; Mahr, B., *Fundamentals of algebraic specification 1,* EATCS Monographs on Theor. Comp. Sc., Springer Verlag, 1985.

[EKMP 82] Ehrig, H.; Kreowski, H.-J.; Mahr, B.; Padawitz, P., *Algebraic implementation of abstract data types,* Theoret. Comp. Sc. **20** (1982) 209-263.

[GrBr 87] Grünler, T.; Broy, M., *Theoretical Foundation of Algebraic Specification and Implementation in* $PA^{nn}dA$*-S.* PROSPECTRA, ESPRIT project #390, report M.2.2.S1-R-1.1, Oct.1987.

[Hen 86] Hennessy, M. *Proving Systolic Systems Correct.* ACM TOPLAS, Vol. 8, No. 3, July 1986, pp. 344-387

[Hoa 85] Hoare, C.A.R. *Communicating Sequential Processes* . Prentice-Hall, 1985

[HoJi 85] Hoare, C.A.R.; He Jifeng. *Algebraic Specification and Proof of Properties of Communicating Sequential Processes.* Technical Monograph PRG-52, Oxford University Comp. Lab., Nov. 1985

[Mil 80] Milner, R. *A Calculus of Communicating Systems.* LNCS, No. 92, Springer-Verlag 1980

[Nico 85] De Nicola, R. *Two Complete Axiom Systems for a Theory of Communicating Sequential Processes.* Information and Control, 64, 1985, pp. 136-172

[OlHo 86] Olderog, E.R.; Hoare, C.A.R. *Specification-Oriented Semantics for Communicating Processes.* Acta Informatica, Vol. 23, 1986, pp. 9-66

[Old 86] Olderog, E.R. *Process Theory: Semantics, Specification and Verification.* LNCS No. 224, Springer-Verlag 1986

[RoDa 86] Roscoe, A.W.; Dathi. Naiem. *The pursuit of deadlock freedom.* PRG-57, Oxford Univ. Computing Laboratory, Nov. 1986

[WiBr 81] Wirsing, M.; Broy, M., *An analysis of semantic models of algebraic specifications.* In Broy, M., Schmidt, G. (eds.) Theoretical Foundations of Programming Methodology, International Summer School, Marktoberdorf, Aug. 1981, pp. 351-412

Appendix: *Reduction to normal form of the net of example 4, after hiding the internal events*

We reproduce the process part of the normal form of the net obtained in *example 4:*

$$
\begin{array}{lllll}
R(s)\,\Delta\,[& i\square & B_i \Rightarrow o_i & & B_i = st_i(s) \equiv 1 \\
 & i\square & C_i \Rightarrow cr_i & & C_i = st_i(s) \equiv 3 \\
 & i\square & D_i \Rightarrow fr_i & & D_i = st_i(s) \equiv 4 \\
 & i\square & H_i \Rightarrow p_i & & H_i = st_i(s) \equiv 2 \wedge \mathrm{free}(s) \\
 & i\square & I_i \Rightarrow v_i &] & I_i = st_i(s) \equiv 5 \wedge \neg\mathrm{free}(s)
\end{array}
$$

where $\{ p_i, v_i \}_{i=1,2}$ are the events to be hidden. The following theorems can be deduced from the specification SP associated with R(s):

$$
\begin{array}{lll}
i, j=1, 2 : \ SP \vdash & B_i \wedge H_i = \mathrm{false},\ B_i \wedge I_i = \mathrm{false}, & C_i \wedge H_j = \mathrm{false},\ C_i \wedge I_j = \mathrm{false} \\
 & D_i \wedge H_j = \mathrm{false},\ D_i \wedge I_j = \mathrm{false}, & H_i \wedge H_j = \mathrm{false},\ I_1 \wedge I_2 = \mathrm{false}
\end{array}
$$

a) Guard folding: The resulting process after eliminating the branches with false guards is:

$$
\begin{array}{llll}
R(s)\,\Delta\,[& i\square & B_i & \Rightarrow o_i \\
 & \square & B_1 \wedge H_2 & \Rightarrow o_1 \square p_2 \\
 & \square & B_1 \wedge I_2 & \Rightarrow o_1 \square v_2 \\
 & \square & B_2 \wedge H_1 & \Rightarrow o_2 \square p_1 \\
 & \square & B_2 \wedge I_1 & \Rightarrow o_2 \square v_1 \\
 & i\square & C_i & \Rightarrow cr_i \\
 & i\square & D_i & \Rightarrow fr_i \\
 & i\square & H_i & \Rightarrow p_i \\
 & \square & H_1 \wedge H_2 & \Rightarrow p_1 \square p_2 \\
 & i\square & I_i & \Rightarrow v_i \quad]
\end{array}
$$

b) Application of operator \: After applying the hiding of $\{p_i, v_i\}_{i=1,2}$, the resulting process will be:

$$
\begin{array}{llll}
RH(s)\,\Delta\,[& i\square & B_i & \Rightarrow o_i \\
 & \square & B_1 \wedge H_2 & \Rightarrow RP_2 \sqcap (o_1 \square RP_2) \\
 & \square & B_1 \wedge I_2 & \Rightarrow RV_2 \sqcap (o_1 \square RV_2) \\
 & \square & B_2 \wedge H_1 & \Rightarrow RP_1 \sqcap (o_2 \square RP_1) \\
 & \square & B_2 \wedge I_1 & \Rightarrow RV_1 \sqcap (o_2 \square RV_1) \\
 & i\square & C_i & \Rightarrow cr_i \\
 & i\square & D_i & \Rightarrow fr_i \\
 & i\square & H_i & \Rightarrow RP_i \\
 & \square & H_1 \wedge H_2 & \Rightarrow RP_1 \square RP_2 \\
 & i\square & I_i & \Rightarrow RV_i \quad]
\end{array}
$$

where $RP_i = RH(s\char`^p_i)$ and $RV_i = RH(s\char`^v_i)$.

c) Unfolding of the non guarded expressions: To do the unfolding of the non guarded expressions we will make use of theorems known to be true in SP'. So, we get the following definitions:

$$
\begin{array}{llll}
B_2(s) \wedge H_1(s) = \mathrm{true}\ \text{implies}\ RH(s\hat{}p_1)\,\Delta & [\ \ \mathrm{true} \Rightarrow o_2 & & \\
 & \square\ \ \mathrm{true} \Rightarrow cr_1\ ] & = & [\ \mathrm{true} \Rightarrow o_2 \square cr_1\ ] \\
\text{because } B_2(s) \wedge H_1(s) = \mathrm{true}\ \text{implies} & B_1(s\hat{}p_1) = \mathrm{false} \quad \wedge & & \\
 & B_2(s\hat{}p_1) = B_2(s) = \mathrm{true} \wedge & & \\
 & H_1(s\hat{}p_1) = \mathrm{false} & & \\
 & \ldots\mathrm{etc}\ldots & &
\end{array}
$$

the rest of the unfoldings can be developed in the same way, obtaining:

$$
\begin{array}{llll}
H_1(s) = \mathrm{true}\ \text{implies}\ RH(s\hat{}p_1) & \Delta\ [& B_2(s\hat{}p_1) \Rightarrow o_2 & \\
 & \square & \mathrm{true} \Rightarrow cr_1\ ] & \\
H_1(s) \wedge H_2(s) = \mathrm{true}\ \text{implies}\ RH(s\hat{}p_1) & \Delta\ [& \mathrm{true} \Rightarrow cr_1\ ] & \\
B_2(s) \wedge I_1(s) = \mathrm{true}\ \text{implies}\ RH(s\hat{}v_1) & \Delta\ [& \mathrm{true} \Rightarrow o_1 & \\
 & \square & \mathrm{true} \Rightarrow o_2\ ] & = \quad [\ \mathrm{true} \Rightarrow o_1 \square o_2\ ] \\
I_1(s) = \mathrm{true}\ \text{implies}\ RH(s\hat{}v_1) & \Delta\ [& \mathrm{true} \Rightarrow o_1 & \\
 & \square & B_2(s\hat{}v_1) \Rightarrow o_2 & \\
 & \square & B_1(s\hat{}v_1) \wedge H_2(s\hat{}v_1) \Rightarrow RH(s\hat{}v_1\hat{}p_2) \sqcap (o_1 \square RH(s\hat{}v_1\hat{}p_2)) & \\
 & \square & H_2(s\hat{}v_1) \Rightarrow RH(s\hat{}v_1\hat{}p_2) \quad] &
\end{array}
$$

where $SP \vdash B_2(s\hat{}p_1) = B_2(s\hat{}v_1) = B_2(s) \wedge B_1(s\hat{}v_1) = \mathrm{true} \wedge H_2(s\hat{}v_1) = H_2(s)$

$$
\begin{array}{lll}
H_2(s) = \mathrm{true} \wedge I_1(s) = \mathrm{true}\ \text{implies}\ RH(s\hat{}v_1\hat{}p_2)\,\Delta & [\ \ \mathrm{true} \Rightarrow o_1 & \\
 & \square\ \mathrm{true} \Rightarrow cr_2 & \\
 &] & = \quad [\ \mathrm{true} \Rightarrow o_1 \square cr_2\ ]
\end{array}
$$

The developments of $RH(s\hat{}p_2)$ and $RH(s\hat{}v_2)$ are simetrical to these. Next, we substitute the definitions of the unfolded expressions in the original definition of RH(s) and, after some simplifications, we get:

$$\begin{array}{llll}
RH(s)\ \Delta\ [& i\,[\,]\,B_i & \Rightarrow o_i & \\
 & [\,]\,B_1 \wedge H_2 & \Rightarrow o_1\,[\,]\,cr_2 & \{p_2\} \\
 & [\,]\,B_1 \wedge I_2 & \Rightarrow o_1\,[\,]\,o_2 & \{v_2\} \\
 & [\,]\,B_2 \wedge H_1 & \Rightarrow o_2\,[\,]\,cr_1 & \{p_1\} \\
 & [\,]\,B_2 \wedge I_1 & \Rightarrow o_1\,[\,]\,o_2 & \{v_1\} \\
 & i\,[\,]\,C_i & \Rightarrow cr_i & \\
 & i\,[\,]\,D_i & \Rightarrow fr_i & \\
 & i\,[\,]\,H_i & \Rightarrow cr_i & \{p_i\} \\
 & [\,]\,H_1 \wedge B_2 & \Rightarrow o_2 & \{p_1\} \\
 & [\,]\,H_2 \wedge B_1 & \Rightarrow o_1 & \{p_2\} \\
 & [\,]\,H_1 \wedge H_2 & \Rightarrow cr_1 \sqcap cr_2 & \{p_1 \sqcap p_2\} \\
 & i\,[\,]\,I_i & \Rightarrow o_i & \{v_i\} \\
 & [\,]\,I_1 \wedge B_2 & \Rightarrow o_2 & \{v_1\} \\
 & [\,]\,I_1 \wedge H_2 & \Rightarrow o_1\,[\,]\,cr_2 & \{v_1\hat{}\,p_2\} \\
 & [\,]\,I_2 \wedge B_1 & \Rightarrow o_1 & \{v_2\} \\
 & [\,]\,I_2 \wedge H_1 & \Rightarrow o_2\,[\,]\,cr_1 \quad] & \{v_2\hat{}\,p_1\}
\end{array}$$

In this expression of RH all the branches contain guarded expressions. We have recorded in brackets the events hidden at each branch.

d) Reduction to normal form: Applying now the rules given in step (d) of section 6, we get:

$$\begin{array}{lll}
RH(s)\ \Delta\ [& B_1 \vee (B_1 \wedge H_2) \vee (B_1 \wedge I_2) \vee (B_2 \wedge I_1) \vee I_1 \vee (I_1 \wedge H_2) & \Rightarrow o_1 \\
 & [\,]\,B_2 \vee (B_2 \wedge H_1) \vee (B_2 \wedge I_1) \vee (B_1 \wedge I_2) \vee I_2 \vee (I_2 \wedge H_1) & \Rightarrow o_2 \\
 & [\,]\,C_1 \vee (B_2 \wedge H_1) \vee H_1 \vee (I_2 \wedge H_1) & \Rightarrow cr_1 \\
 & [\,]\,C_2 \vee (B_1 \wedge H_2) \vee H_2 \vee (I_1 \wedge H_2) & \Rightarrow cr_2 \\
 & i\,[\,]\,D_i & \Rightarrow fr_i \\
 & [\,]\,H_1 \wedge H_2 & \Rightarrow cr_1 \sqcap cr_2 \quad]
\end{array}$$

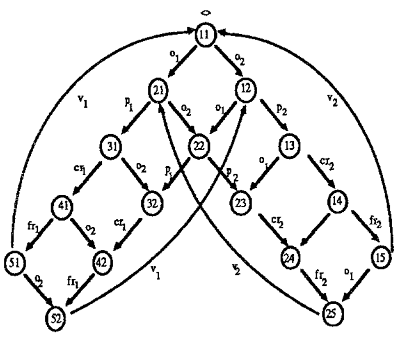
by simplifying the guards we obtain the final process and the definition of the _&_ operation:

$$\begin{array}{lll}
RH(s)\ \Delta\ [& B_1 \vee I_1 & \Rightarrow o_1 \\
 & [\,]\,B_2 \vee I_2 & \Rightarrow o_2 \\
 & [\,]\,C_1 \vee H_1 & \Rightarrow cr_1 \\
 & [\,]\,C_2 \vee H_2 & \Rightarrow cr_2 \\
 & [\,]\,H_1 \wedge H_2 & \Rightarrow cr_1 \sqcap cr_2 \\
 & i[\,]\,D_i & \Rightarrow fr_i \quad]
\end{array}$$

$$\begin{array}{ll}
I_1(s) \wedge H_2(s) = \mathit{true} & \Rightarrow s\&o_1 = s\hat{}\,v_1\hat{}\,p_2\hat{}\,o_1 \\
I_1(s) \wedge \neg H_2(s) = \mathit{true} & \Rightarrow s\&o_1 = s\hat{}\,v_1\hat{}\,o_1 \\
B_1(s) \wedge H_2(s) = \mathit{true} & \Rightarrow s\&o_1 = s\hat{}\,p_2\hat{}\,o_1 \\
B_1(s) \wedge I_2(s) = \mathit{true} & \Rightarrow s\&o_1 = s\hat{}\,v_2\hat{}\,o_1 \\
B_1(s) \wedge \neg H_2(s) \wedge \neg I_2(s) = \mathit{true} & \Rightarrow s\&o_1 = s\hat{}\,o_1 \\
H_1(s) \wedge I_2(s) = \mathit{true} & \Rightarrow s\&cr_1 = s\hat{}\,v_2\hat{}\,p_1\hat{}\,cr_1 \\
H_1(s) \wedge \neg I_2(s) = \mathit{true} & \Rightarrow s\&cr_1 = s\hat{}\,p_1\hat{}\,cr_1 \\
C_1(s) \wedge B_1(s) = \mathit{true} & \Rightarrow s\&cr_1 = s\hat{}\,cr_1 \\
D_1(s) = \mathit{true} & \Rightarrow s\&fr_1 = s\hat{}\,fr_1
\end{array}$$

The definitions for $s\&o_2$, $s\&cr_2$ and $s\&fr_2$ are simetrical to these. The implementation relation is completed with the following definitions for the boolean guards (left names are those of *example 1*, and right names are those of process R at the beginning of the appendix): $A_i = B_i \vee I_i$, $B = C_1 \vee H_1$, $C = C_2 \vee H_2$, $D = H_1 \wedge H_2$, $E_i = D_i$

e) Implementation relation: Obviously, *M=M'*. If SP$^+$ is a correct implementation of SP, then we have proved that S = RH.
In *example 3*, the final model of SP was depicted. In the next figure, the final model of SP$^+$ and the implementation relation is presented. Let us notice that the two typical situations of the implementation relation arise : "junk" values (e.g. 31, 32, 23 and 13 do not implement any value of the model of SP) and "confused" values (e.g. 11, 51 and 15 implement the value tt of the final model of SP):



◇
11
o1
o2
21
12
v1
p1
o2
o1
p2
v2
31
22
13
cr1
o2
p1
p2
o1
cr2
41
32
23
14
fr1
o2
cr1
cr2
fr2
51
42
v1
v2
24
15
o2
fr1
fr2
o1
52
25