Permutable Agents in Process Algebras

François Michel*
Pierre Azéma*
François Vernadat*

ABSTRACT
Within the framework of symmetrical systems, an extension of CCS [6], so-called PCCS, is described. PCCS equips CCS with the concept of pool of agents by means of the explicit structure of Pool expressions. The symmetries whithin a Pools of agents may then be used to simplify the validation process of concurrent systems.
An equivalence relation, so-called Permutability, is formally introduced : two PCCS expressions are permutable iff they can be obtained from each other by a permutation of expressions within a pool. Permutability can be decided in a polynomial time w.r.t. the length of expressions. The Permutability notion allows the definition of symbolic Processes, which describe the system behaviour when inside a pool the agent identities are removed . A transitional semantics is defined and behavioral verifications may be conducted over symbolic Processes.
Key words : Process Algebras, Symmetries, Concurrent Systems, Verification

1 Introduction

In concurrent systems, a single behaviour may be shared by several processes, in other words, all this processes are instances of a same process class. Furthermore, this kind of systems often exhibits architectural symmetries, e.g. two processes of the same class may be substituted one for each other without modifying the system behavior.

In order to study these symmetrical systems, some procedures have ever been designed. Their main advantage lies in the simplification of the validation step by elimination of symmetrical cases in behavioral studies. This approach, initiated by [8] with HL-Trees, formally consists of three steps (cf. Figure 1) :

1. Extraction of a group of symmetries from the formal description. Two states are symmetrical iff they satisfy a symmetry relationship. This

*LAAS/CNRS 7,Avenue du Colonel Roche - 31077 Toulouse Cedex - France

group of symmetries defines the equivalence relation "is symmetrical to" among states.

2. As a consequence, a quotient graph may be considered. This quotient graph may be directly built from the formal description, that is on line. Such a construction algorithm is easily derived from any standard enumeration algorithm, by replacing the syntactical equality among states by the relation "is symmetrical to".
3. The symmetrical properties, i.e. which hold for all the symmetrical states in a class, may be verified over the quotient graph. Study of structural properties (deadlocks, connectivity,....) [3, 5, 9], and temporal logic model checking [2, 1] have been adapted to these quotient graphs.

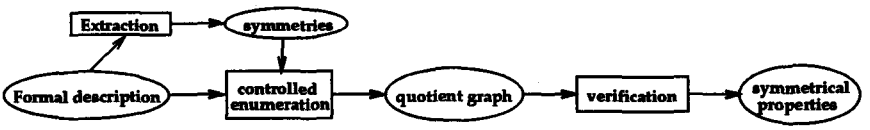

FIGURE 1. The symmetrical approach

The symmetry approach faces two difficulties : the symmetry derivation from the formal description, on one hand, and the computation efficiency for deciding whether two states are symmetrical, on the other hand.

A main contribution of this paper is to propose an efficient application of symmetries to Process Algebras for each step of the former symmetry approach. The motivation is twofold. On one hand, the validation of symmetrical systems described by process algebras will be conducted over reduced graph modulo system symmetries. Consequently, when the reduced graph derivation cost is low, space saving and time speed-up are expected. From a design point of view, on the other hand, the process composition is performed by means of algebraic operators, these operators have to preserve the symmetries of each components. This is the reason why Pool expressions will be introduced.

The purpose is not to promote a new Process Algebra but to present the ingredients which may be added to a Process Algebra, here CCS, in order to use symmetries. These ingredients concern the description formalism and specific types of symmetry :

Description Formalism : The detection of symmetries in the case of usual formalisms is a complex problem, e.g. Symmetries of Petri Nets [9]. Specific formalisms have been designed in order to *explicitly state* symmetries rather than to detect them : the Well-Formed Nets [3, 5], the language $Mur\varphi$ extended by scalarsets [4]. Like in $Mur\varphi$, an already existing formalism, CCS, will be extended by symmetrical structures, Pools of concurrent agents, resulting in PCCS.

The second section briefly introduces the syntax, semantics and behav-

ioral equivalence in CCS, the third section extends these three issues to PCCS.

Symmetries : [2] has shown that, in the general case, to determine whether two states are symmetrical is as hard as the graph isomorphism problem for which no polynomial solution is known. For this reason, PCCS only implements a particular kind of symmetries : all agents in a pool are pairwise permutable. In this context, the relation "is symmetrical to" is called Permutability. Permutability is defined syntactically : two PCCS expressions are permutable iff they can be obtained from each other by an agent permutation within the Pool expressions. The behavioural interpretation of Permutability is made precise by means of the P-Equivalence, which is a specific strong equivalence.

The Permutability definition, its interpretation and its computation, polynomial w.r.t. length of terms, are presented in forth section.

Quotient Graphs : Permutability allows symbolic Processes to be defined as an equivalence class. From each symbolic Process, a symbolic L.T.S. (the quotient graph in the symmetry approach) can be derived by a transitional semantics. An efficient construction algorithm manages to built the symbolic L.T.S. for any PCCS expression.

Symmetrical Properties : The behavior verification is conducted on the symbolic L.T.S.. That is, properties of states, which are symbolic Processes, are derived from the Permutability interpretation by means of the P-Equivalence.

The symbolic Processes and their properties are developed in the fifth section.

PCCS seems to offer a good balance between a low (polynomial) computation cost and a high reduction rate. This question is discussed in a deeper way in the sixth section.

2 CCS Algebra

CCS is a Process Algebra whose basic elements are a set of names $\mathcal{A}$, the associated co-names $\overline{\mathcal{A}}$ and a set $\mathcal{K}$ of agent constants. $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ denotes the set of labels and $Act = \mathcal{L} \cup \{\tau\}$ denotes the set of actions. Its syntax is described in the first part, its transitional semantics in second part. In the third part, three notions of equivalence among processes are given: strong equivalence, observation equivalence and observation congruence.

2.1 Syntax of CCS

The agent expressions are : $0 \quad a.E \quad E+F \quad E|F \quad E \setminus L \quad E[f]$ where E, F are agent expressions, a $\in \mathcal{L}$, $L \subset \mathcal{L}$ and f is a relabeling function, that is a function from $\mathcal{L}$ to $\mathcal{L}$ such that $\overline{f(l)} = f(\overline{l})$; f is extended to Act by $f(\tau) = \tau$.

Every constant A in $\mathcal{K}$ is assumed to have a definition :

$$A \stackrel{def}{=} E \text{ where E is an agent expression.}$$

Remark: the operator $\Sigma_{i \in I} E_i$, where I is any set of indexes, is not included in our version of CCS. Indeed, it will be assumed in the sequel of the article, that *the length of expressions is finite and so is the set of constant names.*

2.2 Transitional semantics of CCS

The semantics of CCS is defined in terms of labelled transition systems (L.T.S.). Their states are agent expressions, and transition labelled by actions are ruled by :

$$\frac{}{a.E \xrightarrow{a} E} \qquad \frac{E_1 \xrightarrow{a} E_1'}{E_1 + E_2 \xrightarrow{a} E_1'} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 + E_2 \xrightarrow{a} E_2'}$$

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1|E_2 \xrightarrow{a} E_1'|E_2} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1|E_2 \xrightarrow{a} E_1|E_2'} \qquad \frac{E_1 \xrightarrow{a} E_1' \; E_2 \xrightarrow{\bar{a}} E_2'}{E_1|E_2 \xrightarrow{\tau} E_1'|E_2'}$$

$$\frac{E \xrightarrow{a} E'}{E \setminus L \xrightarrow{a} E' \setminus L}(a, \bar{a}, \notin L) \qquad \frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E[f]} \qquad \frac{E \xrightarrow{a} E'}{A \xrightarrow{a} E'}(A \stackrel{def}{=} E)$$

Consequently, with each agent expression E, a L.T.S. can be associated.

2.3 Behavioral Equivalence of processes

In [7], three kind of equivalences are introduced; from the strongest to the weakest: strong equivalence, observation congruence, and observation equivalence. The first two relations are also congruence for the operators of CCS. They are all defined according to the same idea : *If two processes are equivalent then they can perform the same action and then become equivalent processes.* Only the semantics of "perform an action" changes.

Strong equivalence is defined as the union of all strong bisimulation :

Definition 2.1 (Strong bisimulation, strong equivalence) *a binary relation among agent expressions, B, is a strong bisimulation iff it verifies :*

$$P \; B \; Q \Rightarrow \quad \forall a, \; \forall P', \; (P \xrightarrow{a} P' \Rightarrow \exists Q', Q \xrightarrow{a} Q' \text{ and } P' \; B \; Q')$$
$$\text{and} \quad \forall Q', (Q \xrightarrow{a} Q' \Rightarrow \exists P', P \xrightarrow{a} P' \text{ and } P' \; B \; Q')$$

the relation $\sim = \cup_{B \text{strong bisimulation}} B$ *is an equivalence relation called strong equivalence.*

To define observation equivalence, the particular action τ, representing internal computation, is considered.Transition relations $\Rightarrow$ and $\Rightarrow_{ref}$ are defined by abstracting internal computation :

- $\forall$ P,Q agent expressions, $\forall a \in Act \; : \; P \stackrel{a}{\Longrightarrow} Q \Leftrightarrow P \xrightarrow{\tau^* a \tau^*} Q$
- $\Rightarrow_{ref} = \Rightarrow \cup \{ \; (P, \tau, P) \; / P \;$ agent expression $\}$

The definition of weak bisimulation and observation equivalence is obtained by substituting $\Rightarrow_{ref}$ for $\rightarrow$ in Definition 2.1. Unfortunately, observation

equivalence is not a congruence for the operator "+", that is why the stronger observation congruence is introduced :

Definition 2.2 (observation congruence, equality)
P and Q are observation equivalent iff
$\forall a \in Act,\ \forall P',\ (P \xrightarrow{a} P' \Rightarrow \exists Q', Q \stackrel{a}{\Longrightarrow} Q' \text{ and } P' \approx Q')$
$\text{and}\ \ \forall Q', (Q \xrightarrow{a} Q' \Rightarrow \exists P', P \stackrel{a}{\Longrightarrow} P' \text{ and } P' \approx Q')$
Observation congruence is also called equality and denoted by the usual symbol "=".

In the sequel, the symbol $\cong$ denotes $\sim$, $\approx$ or $=$ in properties verified by three equivalences.

3 PCCS presentation

In order to explicitly declare symmetries, new concepts are introduced into CCS, resulting in an extended CCS, so-called Pool CCS (PCCS). PCCS describes in a simple way pools of concurrent processes sharing a class of possible behaviours. For each pool of agents, an unique name `Pool_id` is introduced. | `Pool_id` | is the number of agents in `Pool_id`. | `Pool_id` |. | `Pool_id` | is an integer constant, that is, the number of agents for a given pool is constant. agents in the pool are assumed to be numbered from 1 to | `Pool_id` |.

Moreover, the set of labels is partitioned into :

- the set of common labels $\mathcal{L}_{com}$, which enables communication between distinct pools (with distinct pool identifiers),
  $Act_{com} = \mathcal{L}_{com} \cup \{\tau\}$ is the set of common actions,
- the sets of symbolic labels $\mathcal{L}$(`Pool_id`) attached to each pool `Pool_id`. Label a of $\mathcal{L}$(`Pool_id`) represents label a_i of any agent i of pool `Pool_id`.

Example : The Jobshop example described in [7] is extended. Let us consider that n jobbers share p hammers and q mallets to manufacture objects brought by a different conveyer belt for each jobbers. The whole system, depicted in Figure 2, may be represented by 3 pools `JOBBERS`, `HAMMERS` and `MALLETS` such that $| JOBBERS | = n$, $| HAMMERS | = p$ and $| MALLETS | = q$. As each jobber is working on his own conveyer belt, entering and leaving of objects is expressed by symbolic labels `in`, `out` in `JOBBERS`. The other labels `geth`, `puth`, `getm`, `putm` are common labels. In Figure 2, agents J_i, the i^{th} jobbers, H_k, the k^{th} hammers, and M_l, the l^{th} mallets are represented.

The communications between distinct pools are restricted to labels of $\mathcal{L}_{com}$ in order to not break the pool agent symmetry by composition. For instance, suppose that pool `One_Object` is inserted in the Jobshop system, its symbolic labels are $\{in\}$ (cf. Figure 3). The i^{th} agent in `One_Object`

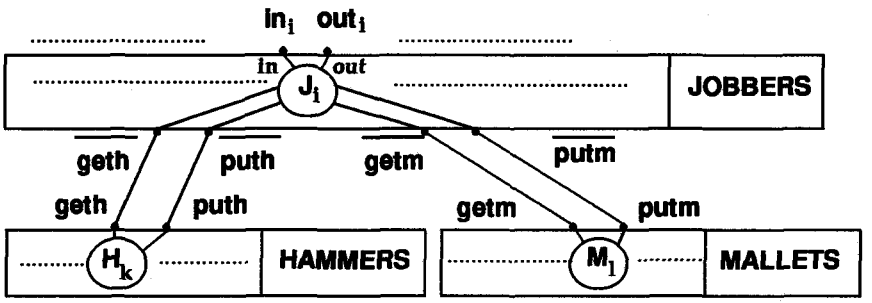

FIGURE 2. Jobshop system

prevents the i^{th} jobber to manufacture more than one object. Now, assume that $\mid One\_Object \mid = 1$, then only the first jobber cannot manufacture two objects, i.e., symmetry among jobbers is broken.

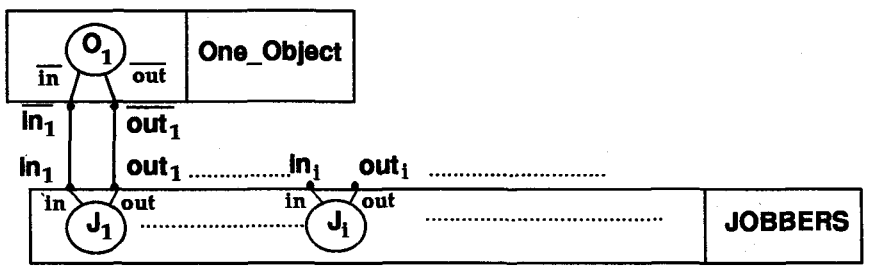

FIGURE 3. Breaking symmetries by communication on symbolic labels

The syntax and the transitional semantics are detailed in the next two parts. The relations between PCCS and behavioral equivalences are explained in the third part. The last part compares the expressive power of CCS and PCCS.

3.1 Syntax of PCCS

The new syntactical elements are :

- pool declaration, to define new pools,
- Pool expression, to define a pool behaviour,

The Pool expression addition to standard CCS expressions results in PCCS expressions.

Declaration of Pools

A pool is declared by : `POOL Pool_id n { symbolic_labels }`
`POOL` is a key word introducing the declaration. `Pool_id` is the identifier of

the declared pool, n is an integer constant defining the number of agents in the pool, | `Pool_id` |. `symbolic_labels` is a sequence of distinct identifiers defining symbolic labels of `Pool_id`, $\mathcal{L}$(`Pool_id`).

Pool expressions

There are three imbricated syntactical levels in PCCS :

Symbolic agent expression describes the behaviour of agents within a Pool. For a given pool, `Pool_id`, it consists of :

- a CCS expression using $\mathcal{L}_{com} \cup \mathcal{L}$(`Pool_id`) as labels,
- a constant A such that $A \stackrel{def}{=} E$ and E is a symbolic agent expression of Pool, `Pool_id`.

The set of symbolic agent expressions attached to a pool, `Pool_id`, defines the class of possible behaviours of agents within `Pool_id`.

Pool expression describes the behaviour of the whole pool. Syntactically, it consists of the name of the pool, `Pool_id`, following by an ordered list of n symbolic agent expressions delimited by parenthesis :

$$\texttt{Pool\_id}(E_1, E_2, \ldots, E_i, \ldots, E_{n-1}, E_n)$$

n must be equal to | `Pool_id` |, and the i^{th} expression in the list, E_i, specifies behaviour of agent i.

PCCS expression describes a system where several agent pools are working and consists of usual CCS operators and Pool expressions.

In the sequel, $E_i, F_i, \ldots$ will range over the symbolic agent expressions, $P, Q, \ldots$ will range over PCCS expressions, and `Pool_id` is a pool of n agents.

Remark : as symbolic agent expressions are usual CCS terms, the imbrication of pools is forbidden in PCCS (this choice is explained in part 6).

Example : In the Jobshop system, the three pools are declared by :

`POOL JOBBERS n {in,out}` `POOL HAMMERS p {}` `POOL MALLETS q {}`

The behaviour of agents are expressed by the symbolic agent expression :

$$\texttt{J} \stackrel{def}{=} in.(\overline{geth}.\overline{puth}.\overline{out}.J + \overline{getm}.\overline{putm}.\overline{out}.\texttt{J})$$

$$\texttt{H} \stackrel{def}{=} \texttt{geth.puth.H} \qquad \texttt{M} \stackrel{def}{=} \texttt{getm.putm.M}$$

Behaviours of pools are expressed by Pool expressions :

$$\texttt{JOBBERS}(\overbrace{J,\ldots,J}^{n\ times}) \qquad \texttt{HAMMERS}(\overbrace{H,\ldots,H}^{p\ times}) \qquad \texttt{MALLETS}(\overbrace{M,\ldots,M}^{q\ times})$$

The whole system is specified by the PCCS expression :

`(JOBBERS(`$J,\ldots,J$`) | HAMMERS(`$H,\ldots,H$`) | MALLETS(`$M,\ldots,M$`)) \ L`

where `L =` $\{geth, getm, putm, puth\}$

3.2 Transitional semantics of PCCS

The PCCS semantics is defined in terms of labelled transition systems like in CCS. The specification rules of the behaviour of Pool expressions are defined in the first part. The PCCS rules are the standard CCS rules plus specific rules for Pool expressions. Some minor modifications are introduced for restriction and relabeling in the case of symbolic labels.

Transitional semantics of Pool expressions

The transitional semantics of a symbolic agent expression is given by the usual CCS rules. The following extra rules define the semantics of Pool expressions :

$$\textbf{Com}\ \frac{E_i \xrightarrow{a} E_i'}{\texttt{Pool\_id}(...E_{i-1}, E_i, E_{i+1}...) \xrightarrow{a} \texttt{Pool\_id}(...E_{i-1}, E_i', E_{i+1}...)}(a \in Act_{com})$$

$$\textbf{Symb}\ \frac{E_i \xrightarrow{a} E_i'}{\texttt{Pool\_id}(..., E_i, ...) \xrightarrow{a_i} \texttt{Pool\_id}(..., E_i', ...)}(a \in \mathcal{L}(\texttt{Pool\_id}))$$

Com is the production of common action a by agent i.

Symb shows how a symbolic label catches the identity of the issuing agent, i.e. the symbolic agent expression is instantiated.

Intra-pool Communications obey to ($sync$ is a distinguished action in $\mathcal{L}_{com}$) :

$$\textbf{Rdv}_2\ \frac{E_i \xrightarrow{a} E_i' \qquad E_j \xrightarrow{\overline{a}} E_j'}{\texttt{Pool\_id}(..., E_i, ..., E_j, ...) \xrightarrow{\tau} \texttt{Pool\_id}(..., E_i', ..., E_j', ...)}\ \left(\begin{array}{l} a, \overline{a} \in \mathcal{L}_{com} \\ a, \overline{a} \neq sync \end{array}\right)$$

$$\textbf{Rdv}_{all}\ \frac{E_i \xrightarrow{\overline{sync}} E_i' \text{ and } \forall j \neq i, E_j \xrightarrow{sync} E_j'}{\texttt{Pool\_id}(E_1, \ldots, E_n) \xrightarrow{\tau} \texttt{Pool\_id}(E_1', \ldots, E_n')}$$

$\textbf{Rdv}_2$ represents the communication between agents i and j in the pool on a common label. Only agents i and j are moving.

$\textbf{Rdv}_{all}$ is a natural rule of atomic multicast within a pool. This kind of multi-synchronization is not in the spirit of CCS but it is very convenient to build high-level specifications; furthermore, it is restricted within a pool. Nevertheless, this rule is not essential and can be deleted from PCCS without altering the results of the next sections.

Example : in the Jobshop system, a jobber which detects a default on its input object, must stop the production in the Jobshop. Thus, the other jobbers must stop working after having manufactured their current object. The description of the new system is obtained by replacing the symbolic agent expression J by J_{watch} in the previous description :

$$J_{watch} \stackrel{def}{=} in.(\ sync.0 + \overline{geth}.\overline{puth}.O_{watch} + \overline{getm}.\overline{putm}.O_{watch}\ )$$
$$O_{watch} \stackrel{def}{=} \overline{sync}.0 + \overline{out}.J_{watch}$$

Pool expressions can be used in usual CCS expressions. Transitional semantics of "+" and "|" and "." remains the same, only restriction and relabeling have a different semantics when they are applied with labels of $\mathcal{L}(\texttt{Pool\_id})$:

$$\frac{E \xrightarrow{a_i} E'}{E \setminus L \xrightarrow{a_i} E \setminus L}(a \in \mathcal{L}(\texttt{Pool\_id}) \text{ and } a, \overline{a}, \notin L) \quad \frac{E \xrightarrow{a_i} E'}{E[f] \xrightarrow{f(a)_i} E'[f]}(a \in \mathcal{L}(\texttt{Pool\_id}))$$

Moreover, a relabeling function must respect the domains :

$$f(\mathcal{L}(\texttt{Pool\_id})) \subseteq \mathcal{L}(\texttt{Pool\_id})$$

Remarks : $\mathcal{L}(\texttt{Pool\_id}) \cap \mathcal{L}(\texttt{Pool\_id}') = \emptyset$ prevents different pools from communicating on symbolic labels. However, two expressions of a single pool may communicate on symbolic labels.

3.3 PCCS behavioral equivalences

The former three equivalence definitions are still valid in the case of PCCS. Moreover, they are congruences for operator `Pool_id` :

Property 3.1 (Pool expressions and behavioural equivalences)
$E_i \cong F_i \Rightarrow \texttt{Pool\_id}(E_1, \ldots, E_i, \ldots, E_n) \cong \texttt{Pool\_id}(E_1, \ldots, F_i, \ldots, E_n)$

Proof (sketch of) :
$B = \{(\texttt{Pool\_id}(E_1, \ldots, E_i, \ldots, E_n), \texttt{Pool\_id}(E_1, \ldots, F_i, \ldots, E_n)/E_i \cong F_i\}$
is a strong (resp. weak) bisimulation if $\cong$ is $\sim$ (resp. $\cong$ is $\approx$) as a consequence of the transitional semantics of Pool expressions.
($\cong$ *is* $=$) as a consequence of ($\cong$ *is* $\approx$). □.

3.4 PCCS power of expression

As any CCS expression is a PCCS expression, PCCS expressive power is greater than CCS. Conversely :

Property 3.2 *If the rule* $\mathbf{Rdv}_{all}$ *is not used then there exists a set of rewriting rules changing each PCCS expression, in a CCS expression having the same derived L.T.S..*

Sketch of proof : For each `Pool_id`, a family $(f_i)_{1 \leq i \leq n}$ of relabeling functions is defined :

$$f_i(a) = \begin{cases} a & if a \in \mathcal{L}_{com} \\ a_i & if a \in \mathcal{L}(\texttt{Pool\_id}) \end{cases}$$

Then, a CCS expression is obtained from each PCCS expression by the rewriting rules :

$$\texttt{Pool\_id}(E_1, \ldots, E_n) \rightarrow E_1[f_1] \mid \ldots \mid E_i[f_i] \mid \ldots \mid E_n[f_n]$$

Finally, verify that the obtained CCS expression share the same derived L.T.S. than P. □

From this property, one could claim that the concept of pool is useless since it can be replaced by usual CCS expressions. But, in this case, the compositional issue of PCCS is lost : nothing ensure that symmetries in CCS expressions will be preserved by composition (see for instance, the example of `One_Object` in the jobshop). Hence, symmetries should be re-computed for each CCS expression.

4 PCCS Symmetries

PCCS symmetries are defined for Pool expressions of the same pool. They involve the notion of Permutability. Two expressions are permutable iff it exists an agent permutation within the pool which make them equal. A deductive system, introduced in the next part, makes this definition precise. The behaviour interpretation of permutability is explained in part two. An efficient permutability computation is presented in part three.

4.1 Permutable expressions

Permutability is defined as the union of Φ-equalities. Two expressions are related by a Φ-equality iff they are equal up to permutation Φ of symbolic agents in pool expressions. Φ-equality and Permutability of pool expressions are first defined. The extension to any PCCS expression is then carried out by means of a deductive system. The recursion problem needs a particular treatment. Φ-equality is decidable over PCCS and so is Permutability. Permutability is finally defined, as an equivalence relation.

Permutable Pool expressions

Symmetries are derived from agent permutations :

Definition 4.1 (Permutable Pool expressions)
Let φ be permutation over $\{1,..,n\}$.
$\texttt{Pool\_id}(E_1, ..., E_n)$ *and* $\texttt{Pool\_id}(E'_1, ..., E'_n)$ *are equal up to φ or φ-equal iff* $\forall i \in \{1, \ldots, n\}$, E_i *and* $E'_{\varphi(i)}$ *are syntactically equal. Two Pool expressions are permutable iff it exists φ such that they are equal up to φ.*

Remark : a more simple definition of permutable pool expression can be : two pool expressions are permutable iff they are syntactically equal up to associativity and commutativity among processes within a pool. However, the extension of permutability to any PCCS expressions requires to identify the permutation which makes two Pool expressions equal. This is the reason why φ-equality is introduced.

PCCS Φ-equality

The extension to any PCCS expression is performed by first, defining a mapping Φ over Pool identifiers such that :

$\forall$`Pool_id`, Φ(`Pool_id`) is a permutation over $\{1, \ldots, |$ `Pool_id` $|\}$.

Equality up to Φ, or Φ-equality, denoted $\stackrel{\Phi}{=}$, is then defined by a deductive system. Obviously, the notion of Φ-equality over Pool expressions remains the same that in 4.1 :

$$(A_1)\ \frac{\texttt{Pool\_id}(E_1,...) \text{ and } \texttt{Pool\_id}(E'_1,...) \text{ are } \Phi(\texttt{Pool\_id})\text{-equal}}{\texttt{Pool\_id}(E_1,...) \stackrel{\Phi}{=} \texttt{Pool\_id}(E'_1,...)}$$

As our motivation is to preserve symmetries by composition, Φ-equality is naturally extended as a congruence :

$$(A_2)\frac{}{0 \stackrel{\Phi}{=} 0} \qquad (R_1)\frac{P \stackrel{\Phi}{=} Q}{P[f] \stackrel{\Phi}{=} Q[f]} \qquad (R_2)\ \frac{P \stackrel{\Phi}{=} Q}{P \setminus L \stackrel{\Phi}{=} Q \setminus L} \qquad (R_3)\frac{P \stackrel{\Phi}{=} Q}{a.P \stackrel{\Phi}{=} a.Q}$$

$$(R_4)\ \frac{P_1 \stackrel{\Phi}{=} Q_1 \textit{ and } P_2 \stackrel{\Phi}{=} Q_2}{(P_1 + P_2) \stackrel{\Phi}{=} (Q_1 + Q_2)} \qquad (R_5)\ \frac{P_1 \stackrel{\Phi}{=} Q_1 \textit{ and } P_2 \stackrel{\Phi}{=} Q_2}{(P_1 \mid P_2) \stackrel{\Phi}{=} (Q_1 \mid Q_2)}$$

In rules R_4, R_5 commutativity of "+" and "|" are not used. Indeed, commutativity would render the computation very complex. For instance, the comparison of $A_1 + \cdots + A_n$ and $B_1 + \cdots + B_n$, would require the examination of all the possible orderings of $A_1, \ldots, A_n$ and $B_1, \ldots B_n$. $n!$ operations are to be performed, and the computation is no more polynomial with the length of the expressions.

Recursion

Recursion requires a specific treatment. A naive idea would be to write the rule :

$$\frac{E \stackrel{\Phi}{=} Q}{A \stackrel{\Phi}{=} Q}(A \stackrel{def}{=} E)$$

But, the query, $\vdash\ A \stackrel{\Phi}{=} 0 + A$ where $A \stackrel{def}{=} 0 + A$, will lead to infinite computation.

So, a new construction, which preserves during the computation the substitution induced by $\stackrel{def}{=}$ is defined : $\Delta \rightsquigarrow P \stackrel{\Phi}{=} Q$ where Δ is a set of terms $P_i \stackrel{\Phi}{=} Q_i$. It means that $P \stackrel{\Phi}{=} Q$ under the assumption that for all i, $P_i \stackrel{\Phi}{=} Q_i$.

The following axiom is then a direct consequence :

$$(A_3)\ \frac{}{\Delta \cup \{P \stackrel{\Phi}{=} Q\} \rightsquigarrow P \stackrel{\Phi}{=} Q}$$

This construction is related to the former ones by the relation :

$$(R_6)\ \frac{P \stackrel{\Phi}{=} Q}{\Delta \rightsquigarrow P \stackrel{\Phi}{=} Q}(\text{for any } \Delta \text{ set of } \Phi\text{-equalities})$$

The recursion rules can finally be completed :

$$(R_7)\ \frac{\Delta \cup \{A \stackrel{\Phi}{=} Q\} \rightsquigarrow E \stackrel{\Phi}{=} Q}{\Delta \rightsquigarrow A \stackrel{\Phi}{=} Q}(A \stackrel{def}{=} E) \quad (R_8)\ \frac{\Delta \cup \{Q \stackrel{\Phi}{=} A\} \rightsquigarrow Q \stackrel{\Phi}{=} E}{\Delta \rightsquigarrow Q \stackrel{\Phi}{=} A}(A \stackrel{def}{=} E)$$

Φ-equality Decidability

By taking into account the former set of axioms and inferences rules, $\stackrel{\Phi}{=}$ is decidable :

Property 4.1 *it can be decided by finite applications of* $A_1, \ldots, R_8$ *whether* $\vdash P \stackrel{\Phi}{=} Q$

Sketch of proof: The derivation tree whose root is $\vdash P \stackrel{\Phi}{=} Q$ is shown to be finite :
(a) as the length of expressions is assumed to be finite, the number of sons of each node in the derivation tree is finite. (b) as the number of constant names is assumed to be finite, the number of possible sets Δ is finite, hence the depth of the tree is finite. (a) and (b) $\Rightarrow$ the derivation tree is finite. □

Permutability

P and Q are said to be permutable, denoted by P $\stackrel{p}{=}$ Q, iff it exists Φ such that $\vdash$ P $\stackrel{\Phi}{=}$ Q.

Property 4.2 *Permutability is an equivalence relation*

Sketch of proof: the set of all mappings Φ is a group from laws :

- $\Phi \circ \Phi'(\texttt{Pool\_id}) = \Phi(Pool_i d) \circ \Phi'(\texttt{Pool\_id})$
- and $\Phi^{-1}(\texttt{Pool\_id}) = (\Phi(\texttt{Pool\_id}))^{-1}$

This group structure implies an equivalence relation, more particularly, neutral element $\Rightarrow$ reflexivity, inverse $\Rightarrow$ symmetry, internal composition law $\Rightarrow$ transitivity. □

4.2 Interpretation of Permutability

Interpretation of Φ-equality

The interpretation of Φ-equality is similar to the other behavioral equivalences : *two processes are equal up to* Φ *iff if they can perform the same action up to* Φ *and then become* Φ*-equal processes* . The meaning of equal actions up to Φ is made precise by :

$$\begin{cases} \textit{if } a \in \mathcal{L}(\texttt{Pool\_id}) \textit{ then } \Phi(a_i) = a_j \textit{ with } j = \Phi(\texttt{Pool\_id})(i), \\ \textit{if } a \in \mathcal{L}com \textit{ then } \Phi(a) = a \end{cases}$$

The bisimulation and equivalence up to Φ may then be defined :

Definition 4.2 *A binary relation B_Φ is a bisimulation up to Φ iff*

$$P B_\Phi Q \Rightarrow \quad \forall a,\ \forall P',\ (P \xrightarrow{a} P' \ \Rightarrow\ \exists Q',\ Q \xrightarrow{\Phi(a)} Q' \text{ and } P' B_\Phi Q')$$
$$\text{and}\quad \forall Q',\ (Q \xrightarrow{\Phi(a)} Q' \ \Rightarrow\ \exists P',\ P \xrightarrow{a} P' \text{ and } P' B_\Phi Q')$$

The relation $\overset{\Phi}{\sim} = \cup\, B_\Phi$ is called Φ-equivalence.

The Φ-equality is sound for the Φ-equivalence :

Property 4.3 (Soundness of Φ-equality) $\vdash P \overset{\Phi}{=} Q \Rightarrow P \overset{\Phi}{\sim} Q$

Sketch of proof: $\{(P,Q)/P \overset{\Phi}{=} Q\}$ is a bisimulation up to Φ (use $A_1, \ldots, R_8$). □

Interpretation of Permutability

Permutability is defined as the union of all Φ-equalities, so permutability interpretation is defined as the union of all Φ-equivalences, or P-Equivalence, denoted $\overset{p}{\sim}$.

The Φ-equalities properties entail the following property:

Property 4.4 (Interpretation of $\overset{p}{=}$) $P \overset{p}{=} Q \Rightarrow P \overset{p}{\sim} Q$

4.3 Efficient Computation of Permutability

Processing Pool expression

Let $\mid X \mid$ be the cardinal of the finite set X. To decide whether two Pool expressions are permutable does not require the exhaustive enumeration of all the $\mid$ `Pool_id` $\mid$! possible permutations because of the following fundamental property :

Property 4.5 *Let $\{P_1, \ldots, P_m\}$ and $\{Q_1, \ldots, Q_m\}$ partitions of $\{1, ..., n\}$,* $(\forall 1 \leq i \leq m :\mid P_i \mid = \mid Q_i \mid) \Leftrightarrow (\exists\ \Phi$ *permutation of* $\{1, ..., n\} : \Phi(P_i) = Q_i)$

In the case of Pool expressions, partitions are induced by the notion of P-state : The P-state of a Pool expression $\texttt{Pool\_id}(E_1, \ldots, E_n)$, denoted $s(\texttt{Pool\_id}(E_1, \ldots, E_n))$ is the set $\{(P_k, E^k)/1 \leq k \leq m\}$ such that $P_k \subset \{1, ..., n\}$, is the set of agents which share the same symbolic expression E^k. $\{P_k/1 \leq k \leq m\}$ constitutes a partition of {1,...,n}.

$\mid s \mid$ is the set $\{(\mid P_k \mid, E^k)/1 \leq k \leq m\}$ which gives for each k the number of agents which share E^k.

Example : in a Jobshop system with 3 hammers, it holds that:

$$s(\ HAMMERS(H, puth.H, H)\ ) = \{\ (\{1,3\}, H),\ (\{2\}, puth.H)\ \}$$
$$\mid s(\ HAMMERS(H, puth.H, H)\ ) \mid = \{\ (2, H),\ (1, puth.H)\ \}$$

If s (resp. s') denotes the P-state of a Pool expression P (resp. P') then from property 4.5 :

Property 4.6 $\mid s \mid = \mid s' \mid \Leftrightarrow P \overset{p}{=} P'$

Example : $HAMMERS(H, puth.H, H) \stackrel{p}{=} HAMMERS(H, H, puth.H)$ since:
$|\ s(\ HAMMERS(H, puth.H, H)\ )\ | = |\ s(\ HAMMERS(H, H, puth.H)\ )\ | = \{(2, H), (1, puth.H)\}$
Intuitively, in both cases, there are two free hammers.

Extension to PCCS

The extension of P-state to PCCS is similar to the extension of Φ-equality. The whole process is rather technical. Therefore, it is detailed in appendix 1. Basically, it uses the fact that Φ-equality is defined as a congruence and P-state is computed by intersection of partitions.

5 Validation in PCCS

PCCS symmetries may be applied for an efficient reduction of the state space. This reduction is performed by associating symbolic process $\overline{P}$ with each PCCS process P, as defined in the next section. The transitional semantics of symbolic Processes, derived from the PCCS transitional semantics, is then used to build a L.T.S.. The algorithm, described in section 2, takes, as input, a PCCS expression P and constructs the L.T.S. derived from the symbolic Process $\overline{P}$. The symbolic Process properties are presented in section 3.

5.1 Symbolic Processes

A symbolic Process is an equivalence class of PCCS processes with respect to the equivalence relation $\stackrel{p}{=}$. The equivalence class of process P is denoted $\overline{P}$. It represents a process up to a permutation among agents in Pools. This abstraction can be extended to actions by the mapping **abst**:

$$\begin{cases} \textit{if } a \in \mathcal{L}_{com} \textit{ then} & abst(a) = a \\ \textit{if } a = b_i \textit{ with } b \in \mathcal{L}(\texttt{Pool\_id}) \textit{ then} & abst(a) = b \\ \textit{for all co-action } \overline{a}, & abst(\overline{a}) = \overline{abst(a)} \end{cases}$$

The transitional semantics of symbolic Processes is defined by the rule:

$$\frac{P \xrightarrow{a} Q}{\overline{P} \xrightarrow{abst(a)} \overline{Q}}$$

That is, the identity of agents is abstracted. This rule is not ambiguous since two processes, attached to the same symbolic Process gives the same symbolic transitions by the previous rule. In other words :

Property 5.1 $(\overline{P} = \overline{R}$ *and* $P \xrightarrow{a} Q) \Rightarrow$
$(\exists b,\ \exists T,\ abst(a) = abst(b)$ *and* $\overline{Q} = \overline{T}$ *and* $R \xrightarrow{b} T)$

Sketch of proof: from the interpretation of Permutability by P-Equivalence. □

The symbolic L.T.S. of PCCS process P is the L.T.S. which can be derived from $\overline{P}$ using the transitional semantics of symbolic Processes.

5.2 The symbolic L.T.S.

The algorithm, depicted in Table .1, build the symbolic L.T.S. of P. The algorithm consists of a partial exploration of the L.T.S. which can be derived from P. An equivalence class of processes is represented by a class member, the first reached one (line (*) in the algorithm).

In the algorithm, $\rightarrow$ denotes the set of the symbolic L.T.S. transition relation, the reached states are putted in set **Reached**, the states whose transitions are to be fired are stored in stack **To_Explore**.

```
PROCEDURE SYMBOLIC_LTS( P : IN PCCS expression) IS
   Reached := {P}; Push(To_Explore,P) ; →= ∅;
   WHILE To_Explore ≠ empty_stack LOOP
         Pop(To_Explore,Q);
         FOR ALL (Q,a,R) PCCS transitions
             IF ∃ R' ∈ Reached such that R ≝ R'                    (*)
                THEN →=→ ∪{(Q, abst(a), R')};
                ELSE Reached := Reached ∪ {R};
                     →:=→ ∪{(Q, abst(a), R)};
             END IF;
         END FOR ALL;
   END WHILE;
END PROCEDURE;
```

TABLE .1. Symbolic L.T.S. derivation algorithm

Property 5.2 (Soundness) *The algorithm of Table .1 builds the symbolic L.T.S. of P*

Sketch of proof: consequence of Property 5.1. □

5.3 Properties of symbolic Processes

The property of symbolic Processes can be deduced from the interpretation of Permutability in terms of P-Equivalence.

The Φ-equivalence notion is similar to the usual behavioral equivalences :

Property 5.3 $Q \overset{\Phi}{\sim} P$ *and* $Q \cong R$ *and* $R \overset{\Phi}{\sim} T \Rightarrow P \cong T$

The following property may then be deduced :

Property 5.4 $P \cong Q \Rightarrow \overline{P} \cong \overline{Q}$

This property is interesting in its negative form : $\overline{P} \ncong \overline{Q} \Rightarrow P \ncong Q$
That is it can be deduced over symbolic processes that two PCCS expressions are not linked by a behavioral equivalence.

Unfortunately, the converse property is false as shown by Figure 4.

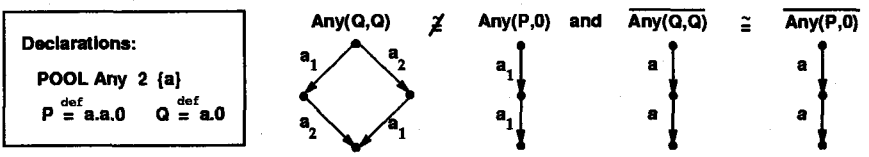

FIGURE 4. Counter example

Some weaker properties can nevertheless be decided :

1. $\mathcal{L}(\texttt{Pool\_id}) = \emptyset \Rightarrow \texttt{Pool\_id}(E_1, \ldots, E_n) \sim \overline{\texttt{Pool\_id}(E_1, \ldots, E_n)}$,
2. P is a deadlock $\Leftrightarrow \overline{P}$ is a deadlock,
3. P is divergent $\Leftrightarrow \overline{P}$ is divergent
   (P is divergent iff it can perform an infinite sequence of τ)

Sketch of proof: (1) consequence of property 5.1, and by noticing that a $= abst(a)$ if a $\in Act_{com}$. (2)+(3) from Property 5.1. □
Property (1) is very interesting since it allows a partial compositional analysis by substituting $\overline{\texttt{Pool\_id}(E_1, \ldots, E_n)}$ for $\texttt{Pool\_id}(E_1, \ldots, E_n)$ in the system description. Moreover, it expresses a simple intuitive idea as shows the following example:

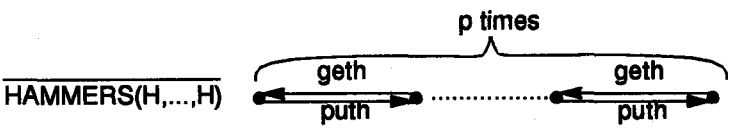

FIGURE 5. Symbolic L.T.S.

Example : in the Jobshop system, the symbolic L.T.S. of pool expression $HAMMERS(H, \ldots, H)$ is described in Figure 5. In CCS, it can be expressed by the agent ham_p such that :

$$\forall 1 \leq j \leq p-1, ham_j \stackrel{def}{=} geth.ham_{j-1} + puth.ham_{j+1}$$
$$ham_0 \stackrel{def}{=} puth.ham_1 \qquad ham_p \stackrel{def}{=} geth.ham_{p-1}$$

The same process can be applied to pool $MALLETS$. And the system described by :

$(JOBBERS(J, \ldots, J) \mid ham_p \mid mallet_q) \setminus \{geth, puth, getm, putm\}$
is strongly equivalent to the initial description of section 3 by Property (1). Hence, a conceptually simple expression can be automatically replaced by an expression easier to analyze.

6 PCCS enhancement

Two criteria are essential for a symmetrical formalism :

- the number of states of the quotient graph with respect to the initial graph. The reduction rate must be as high as possible.
- the quotient graph construction complexity must be low in order to preserve in time the save in space.

To formalize the reduction rate, let us consider a set of n agents (parameter n represents the complexity of the system). Each agent behaviour has m states. The agent composition is assumed to not restrict their reachable states, that is, the number of states of the whole systems is m^n : the number of state grows exponentially with the complexity.

The quotient graph state number of this system is now computed for several groups of agent permutations. This defines the theoretical reduction power of each permutation group.

The procedure complexity for determining whether two states are symmetrical is directly related to the computation complexity.

Table .2 shows the results in the case of four symmetries:

- The group of rotation, the agents are located on a ring,
- The dihedral group, the agents are located on a polygon,
- PCCS, i.e the group of all permutations, the agents are in a set (or a complete net),
- 2D-PCCS, an extension of PCCS where a pool of agents can contain other pools of agents. For instance, the agents are located on a tree.

	Size of quotient	Complexity w.r.t n
Rotations	$\geq m^n/n$	polynomial
Dihedral group	$\geq m^n/2n$	polynomial
PCCS	$C^n_{n+m-1} \leq (n+m)^m$	polynomial
2D-PCCS	$\leq ((n/2)+m)^{2m}$	equivalent to the graph isomorphism problem

TABLE .2. Results for different kinds of symmetries

It seems not interesting to use rotations or dihedral groups in order to fight against combinatorial explosion, because their reduction rate is low.

2D-PCCS could be a good candidate but the complexity of computation constitutes a handicap. Finally, PCCS seems to be a good trade-off. Besides, the list is not exhaustive, and other kind of symmetries may combine good figures.

7 Conclusion

PCCS supplies a suitable formalism for Pools of agents and their symmetries. The agent Pools are described by a new structure: Pool expressions. The symmetries are expressed by means of Permutability concept. As Permutability is an equivalence relation, symbolic Processes can be defined as an equivalence class. A symbolic transitional semantics is brought to symbolic processes. Furthermore, an algorithm is available for computing the symbolic L.T.S. of a PCCS expression. Permutability can be decided in polynomial time, consequently the computation cost of the symbolic L.T.S. is low. The Permutability interpretation in terms of P-Equivalence, makes possible verifications over symbolic L.T.S. : divergence, showing that two processes are not related by behaviour equivalences ...

Now, we are working on solutions to verify behavioral equivalence on symbolic processes. We think that it can be performed by introducing new synchronization operators $||$ such that behavioral equivalence between P and Q can be verified upon the symbolic process $\overline{P \,||\, Q}$.

8 References

[1] A. Sistla E. Emerson. Symmetry and model checking. In *Computer Aided Verification*, pages 463–478. Lecture Notes in Computer Science 697, June-July 1993.

[2] T. Filkorn E.M. Clarke and S. Jha. Exploiting symmetry in temporal logic model checking. In *Computer Aided Verification*, pages 451–462. Lecture Notes in Computer Science 697, June-July 1993.

[3] G. Franceschinis G. Chiola, C. Dutheillet and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In *High-Level Petri Nets*, pages 373–396. Springer-Verlag, 1991.

[4] C. Ip and D. Dill. Better verification through symmetry. In *Int. Symp. on Computer Hardware Description language and their Application*, 1993.

[5] Claude Dutheillet Lamonthezie. *Symétrie dans les Réseau Colorés.* PhD thesis, Université Paris 6, 1991.

[6] Robin Milner. *A Calculus of Communication Systems*, volume 92. Springer-Verlag, lncs edition, 1980.

[7] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[8] L.O. Jepsen P. Huber, A.M. Jensen and K. Jensen. Reachability trees for high-level petri nets. In *High-Level Petri Nets*, pages 319–350. Springer-Verlag, 1991.

[9] K. Schmidt. Symmetries of petri nets. Informatik-Berichte 33, Humbolt-Universität zu Berlin, 1994.

1 Efficient Computation of Permutabilty

1.1 P-states of Pool expression sets

The notion of P-state is generalized to sets of Pool expressions by :
Let $(\texttt{Pool\_id}(E_1^i, \ldots, E_n^i))_{1 \leq i \leq l}$ be a family of l Pool expressions of the same Pool, `Pool_id`.
The P-state of this family is the set: s = { (P_k, V_k) / $1 \leq k \leq m$ } where

- $(V_k)_{1 \leq k \leq m}$ is a family of vectors of length l :
  $V_k = (V_k^1, \ldots, V_k^l)$ where V_k^h is a symbolic agent expression.
- j and g in P_k means that, for each $1 \leq h \leq l$, agents j and g share the same agent expression V_k^h in the h^{th} Pool expression:

$$\text{l Pool expressions} \begin{cases} \texttt{Pool\_id}(\ldots, \overset{j}{V_k^1}, \ldots, \overset{g}{V_k^1}, \ldots) \\ \vdots \\ \texttt{Pool\_id}(\ldots, V_k^l, \ldots, V_k^l, \ldots) \end{cases}$$

Moreover, $| s | = \{ (| P_i |, V_i) / 1 \leq i \leq m \}$.

The P-state of a singleton $\{\texttt{Pool\_id}(E_1, \ldots, E_n)\}$ defined in section 4.3 is denoted $s(\texttt{Pool\_id}(E_1, \ldots, E_n))$. The P-state of the union of two families can be computed from the P-states of these two families, by the operation $\sqcap$:

Property 1.1 *Let s (resp. s') be the P-state of family $\mathcal{F}_1$ (resp. $\mathcal{F}_2$),*
$s \sqcap s'$ is the smallest set obtained by :
If $(P_k, V_k) \in s$ and $(P'_h, V'_h) \in s'$ and $P_k \cap P'_h \neq \emptyset$ then $(P_k \cap P'_h, V_k.V'_h) \in s \sqcap s'$
($V_k.V'_h$ is the concatenation of vectors V_k and V'_h).
$s \sqcap s'$ is the P-state of $\mathcal{F}_1 \cup \mathcal{F}_2$.

Indication of proof: from the property, if $\mathcal{P}$ and $\mathcal{Q}$ are partition of $\{1, \ldots, n\}$ then so are

$$\mathcal{P} \sqcap \mathcal{Q} = \{P \cap Q / \ P \in \mathcal{P}, \ Q \in \mathcal{Q}, P \cap Q \neq \emptyset\} \square$$

Moreover, the indetermined P-state denoted ω is introduced. It is the P-state associated to an empty set of Pool expressions. It is assumed that: $\omega \sqcap s = s \sqcap \omega = s$ and $| \omega | = \emptyset$.

1.2 Extension to PCCS expressions

First, an extended P-state S is defined as a mapping such that :
$\forall$`Pool_id`, S(`Pool_id`) is a P-state of a Pool expression family of `Pool_id`.
The operations over P-states are naturally extended :

$$| S | : \texttt{Pool\_id} \mapsto | S(\texttt{Pool\_id}) |$$

$$S \sqcap S' : \texttt{Pool\_id} \mapsto S(\texttt{Pool\_id}) \sqcap S'(\texttt{Pool\_id})$$

The indetermined extended P-state denoted Ω is defined by:

$$\forall \texttt{Pool\_id}, \ \Omega(\texttt{Pool\_id}) = \omega.$$

Now, the extended P-state S(P) associated with each P can be computed by :

$$S(0) = \Omega \qquad S(a.P) = S(P) \qquad S(P[f]) = S(P) \qquad S(P \setminus L) = S(P)$$

$$S(P + Q) = S(P \mid Q) = S(P) \sqcap S(Q)$$

$$S(\texttt{Pool\_id}(E_1, \ldots)) = \Omega \sqcap s(\texttt{Pool\_id}(E_1, \ldots))$$

Recursion is treated like for Φ-equality, i.e., a new construction is defined :
$\Delta \rightsquigarrow S(P) = S$, where Δ is a set of equalities $S(P_i) = S_i$.
It means that S(P)=S under the assumption that for all i, $S(P_i) = S_i$.
Hence, the axiom and the rule :

$$\Delta \cup \{S(P) = S\} \rightsquigarrow S(P) = S \qquad \frac{S(P) = S}{\Delta \rightsquigarrow S(P) = S}$$

The Recursion is processed by :

$$\frac{\Delta \cup \{S(A) = \Omega\} \rightsquigarrow S(E) = S}{S(A) = S}(A \stackrel{def}{=} E)$$

For the same reasons that Φ-equality, the computation of S(P) is always finite. And the Property 4.6 about P-states and Permutability can be extended :

Property 1.2 $P \stackrel{p}{=} Q \Leftrightarrow \mid S(P) \mid = \mid S(Q) \mid$

Sketch of proof: by structural induction over P + Property 4.6 □.
Moreover, since the computation of $\sqcap$ is polynomial with the arity of each pools of agents, the computation of the P-state is polynomial with the length of the system PCCS description, i.e., the sum of the length of the PCCS expressions which represents the whole behaviour and the length of all needed expressions $A \stackrel{def}{=} E$.