Synthesis of Impartial Deadlock-free Programs for Concurrent Systems

J. EZPELETA(*) **S. HADDAD(+)**

(*) Centro Politecnico Superior. Univ. Zaragoza

C/ Maria de Luna 3. 50015-Zaragoza. Spain
Tel: ++34 76 517274 Fax: ++34 76 512932
e-mail: ezpeleta@etsii.unizar.es

(+) Université P. & M. Curie

Laboratoire MASI -C.N.R.S.
4, Place Jussieu
Paris 75252 CEDEX 05. France
Tel: ++33 1 44277104 Fax: ++33 1 44276286
e-mail: haddad@masi.ibp.fr

Abstract

This paper provides an algorithm for parallel program synthesis. We study the particular but frequent case of sequential processes cooperation via shared resources. Given such a set of these processes (which may be non deterministic ones) , the algorithm outputs a parallel program which ensures safeness, liveness and fairness. The algorithm is divided in three parts: first it computes for each process some sets of unavoidable local states, thus if possible it selects one of these sets for each process - the selected sets must fulfill some constraints of consistency - at last it builds the program by adding new conditions on the local evolution of each process. The advantages of this algorithm are numerous: there is no centralized mechanism, the communication is organized through a virtual ring, and at last the algorithm is incremental (i.e. some parts of the algorithm do not need to be computed again if another process is added).

This work has been done while J. Ezpeleta was a researcher at the laboratory MASI under grant of the "Programa Nacional de becas de folmacion de Personal investigador en el Extranjero" of the "Ministerio de Educacion y Ciencia" of Spain.It was partially supported by the DEMON Esprit Basic Research Action 3148 and the PRONTIC-O358/89

1 INTRODUCTION

It is well-known that the different verification technics for parallel programs all suffer serious drawbacks (e.g. time and space complexity for methods based on reachability graph or incomplete validation for inference methods). So an interesting alternative to these methods is the synthesis of a program starting from a specification. Such a method would simultaneously solve two problems: the conception and the verification of the program.

Developing a theory of program synthesis involves four steps: the choice of the specification and the program models, the choice of a common semantics for these two models, the design of a transformation algorithm and the proof of soundness (and sometimes completeness) of this algorithm.

The choice of the specification model is closely related to the properties one should express about programs; for parallel programs a good candidate could be some class of temporal logics formulae (LTL, CTL, CTL*,...) [Sis85] since they enable to state safety, liveness and fairness properties. The program model should offer the parallel mechanisms that the real program will include (e.g. message passing and/or shared variables and/or semaphores, ...). The common semantics of these models must at least include reachable states and transition sequences in order to give sense to the formulae of the specifications. In order to present the objectives that the transformation algorithms should attain, let us discuss two existing methods.

In the first method [Old85], the specification model is a class of formulae with a variable for the sequence which leads to the current state and another variable for the set of actions enabled in this state. Some attributes and operators enable for instance to project sequences, to compare sequences, etc. The program model is T.C.S.P. [Br084] a theoretical version of C.S.P. [Hoa78] where the main parallel operator is the synchronization between processes for some subset of actions. The semantics of this model is a set of tuples composed by a sequence and a set of actions (with the same meaning as above). In case of the specification, it is the set of behaviors which fulfill the formula whilst in case of the program, it is the set of all possible behaviors of this program. A program satisfies a formula if its set of behaviors is included in the one of the formula. The transformation algorithm is based on a model including the specification and the program model with the same semantics and a set of rules which transform some part of that formula into a piece of program.

The method works at the syntactical level, so the rules are very efficient. Moreover modularity of the method is ensured by a meta-rule which establishes that every transformation can be inserted in any context. However this method suffers some drawbacks: some rules can not be automatized, moreover some transformations do not obey any rule but are only justified and even when some program satisfies the formula, the rules may not find any program (incompleteness).

The second method [Eme83] is based on the algorithm of satisfiability for the class of temporal logic formulae CTL. The specification model enables to state local properties of a process (mainly its possible behavior) and global properties (critical section, fairness, ...). Then if this specification is satisfiable, the algorithm produces a finite state graph which is the reachability graph of the program. Thus in order to obtain the program, this state graph is "folded" by adding global variables and defining conditions on the evolution of each process. Consequently the program model is based on processes which share global variables.

This algorithm always finds a program if such one exists. But the drawbacks of this method are fourfold : the algorithm works at the semantic level (state graph of the model) so the complexity of the method can prevent its use, the method is not modular as the evolution of some process may explicitly depend of the state of another one, the method is not incremental: adding a new process even with the same specification as an old one requires to redo the whole algorithm; moreover the new program may be very different from the old one and at last the use of shared variables prohibits its application in distributed systems. There is also a method for linear temporal logic [Man82] which presents similar features as the preceeding one.

As the main problem with parallel programs is the sharing of resources, we show here how to obtain a good algorithm of program synthesis in this case. The inputs of our algorithm are a set of processes defined by their automata. To each state of an automata is associated a multiset of resources required to execute the internal steps of the process during this state. Thus we give the semantics of a parallel program for these processes (a tree of the possible computations) and we define the three requirements for the program: absence of deadlock, impartiality of the processes execution and independance of the processes local choices.

In order to obtain an impartial program, we compute for each automata its set of control points subsets (a minimal set with some cut property). We show then how to build *an impartial program with a very simple mechanism based on the control points.* However this program is not deadlock free, so we restrict the states transitions of a process by adding "a resources test and set instruction" to every transition. More precisely we avoid the deadlock by ensuring that some particular scheduling of processes is always possible. This mechanism is similar to the banker algorithm [Hab69] *but without banker and without any knowledge given by a process to another one!* Moreover the parallel program that we build preserves of the independance of local choices.

The balance of the paper is as follows. In the second paragraph, we define the specification of the processes, the semantics of a parallel program composed by these processes and the properties required for the parallel programs. In the third paragraph, we give the impartial program. In the forth paragraph, we give our final program, we prove that it fulfills the requirements. In the last paragraph, we discuss the advantages and the drawbacks of our method and we give some perspectives to this work.

2 SPECIFICATION AND PROPERTIES OF A PARALLEL PROGRAM

Definitions and notations

N denotes the set of positive integers.

Let A be a finite set , then Bag(A) is the set of multisets on A defined by :
$\mathrm{Bag}(A) = \{v \mid v \text{ is a mapping } A \to \mathbf{N}\}$
Let $v \in \mathrm{Bag}(A)$, then one denotes $v = \sum_{a \in A} v(a).a$
Let $v, v' \in \mathrm{Bag}(A)$, then

- $v \geq v' \Leftrightarrow \forall a \in A, v(a) \geq v'(a)$
- $v + v' = \sum_{a \in A} (v(a)+v'(a)).a$
- $\mathrm{Sup}(v,v') = \sum_{a \in A} \mathrm{Sup}(v(a),v'(a)).a$
- if $v \geq v'$ then $v-v' = \sum_{a \in A} (v(a) - v'(a)).a$

2.1 SPECIFICATION OF A SYSTEM OF PROCESSES

First of all we need to specify the features of every process appearing in the parallel program. Let us suppose a system composed by n processes P_i ($i \in \mathbf{I}=\{1,...,n\}$) sharing a set R of ressources, every one of them belonging to the set ot types of resources $RS=\{R_1,...,R_m\}$. Since it may have multiple copies of the same resource, R will be a multiset over the set RS. In the following, R is the multi-set of system resources:

$$R = \sum_{j=1}^{m} r_j.R_j$$

Id est, there are r_j copies of the resource R_j

Every process is specified by means of its automata; the automata is a graph for which every node is a pair $(s,R(s))$, where s represents a possible state of the process and $R(s)$ is the multiset of resources used in this state. Given a process P_i, we will denote by S_i the set of its states. If, in this automata, there exists an arc from a state s to another s' we will say that s' is a successor of s. If E is a set of states of a process P, $R(E) = \sup\{ R(s) \mid s \in E\}$(i.e. the minimal multiset of resources such that $\forall\ s \in E\ R(E) \geq R(s)$). In the present paper, we will suppose that every automata is strongly connected.

Example Let us suppose a system composed of three processes sharing three copies of the same resource R0. In this case RS={R0} , R = 3.R0 and the processes, named P_1, P_2 and P_3 are represented by the following automatas.

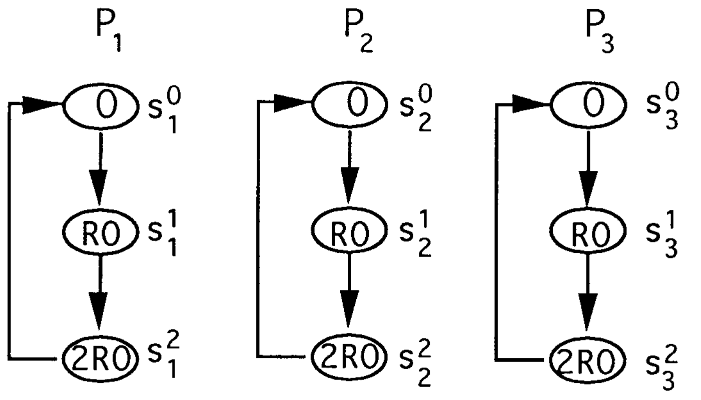


Figure 1: A system sharing copies of a unique kind of resource

In the following, we will denote $s_{i,j}$ the state i of process j. In this example, for instance, $\mathcal{R}(s_{1,1})$ = R0. If E={$s_{1,1}, s_{1,2}$} then $\mathcal{R}$(E)=2.R0. We conclude this paragraph by summarizing in a definition what preceeds.

Definition 2.1 A system of processes *SP* = <RS , R , P , $\mathcal{R}$ > is defined by :
-RS ={R_1,...,R_m} a set of resources
-R a multiset on RS
-P = {P_1,...,P_n } a set of strongly connected automatas (or processes)
where P_i = { S_i ,$\rightarrow_i$ } and S_i is the set of nodes of P_i with | S_i | > 1
and $\rightarrow_i$ the successor relation.
- $\mathcal{R}$ is a function from $\cup$ S_i to Bag(RS)

2.2 PARALLEL PROGRAMS FOR A SYSTEM OF PROCESSES

In this section we define the possible parallel programs for a given system. Whatever the definition of the parallel program may be, a state of this program must include a local state and a multiset of resources for each process. Moreover if this program is consistent with the specification, the multiset of resources must be greater or equal than the one associated to the local set and the sum of the busied resources may not exceed the resources of the system. This leads to the next definition.

Definition 2.2 (Sound state) Let *SP* = <RS , R , P , $\mathcal{R}$ > be a system of processes. A (global) sound state of this system is a tuple (s_1,...,s_n,r_1,...,r_n) , $s_i \in$ Pi , $r_i \in$ Bag(RS) verifying:

$$\forall\, i\, ,\, r_i \geq \mathcal{R}(s_i) \text{ and } \sum_{j=1}^{n} r_j \leq R.$$

In the following we will denote by SS the set of Sound States. Given two sound states (s,r) and (s',r') of a system *SP,* we will say that (s',r') is a (global) successor of (s,r) if and only if $\exists$ k $\in$ **I** such that if i$\neq$k then $s_i = s_i'$ else s_k' is a successor of s_k in the process P_k. By analogy with the successor relation of the processes, this fact will be denoted by means of (s,r) $\rightarrow_k$ (s',r').

Example In the previous example, $(s_{1,0},s_{2,1},s_{3,1},0,R0,R0)$ and $(s_{1,1},s_{2,1},s_{3,1},R0,R0,R0)$ are sound states and $(s_{1,0},s_{2,1},s_{3,1},0,R0,R0) \rightarrow_1 (s_{1,1},s_{2,1},s_{3,1},R0,R0,R0)$ while $(s_{1,1},s_{2,1},s_{3,1},R0,R0,2.R0)$ is not a sound state.

We can now associate to a system of processes, a set of "programs". A program can be seen as a control structure for the system. As we want to consider any kind of program, we just define the domain of the program as the set of possible states (not necessarily reachable) with an initial one and a successor relation. We associate to each state of the program a sound state of the system and we require that the successor relations are compatible.

Definition 2.3 (Program) Let $SP = <RS , R , P , \mathcal{R} >$ be a system. A (parallel) program of this system is a 4-tuple $P = <D,d_o,F,SR>$ where:
- D is a set, called the domain of the program
- $d_o \in D$ is the initial state
- F is a function from V to SS
- $SR \subset V \times V$ is the successor relation and verifies that, if d and d' are elements of V such that d SR d' then $\exists\ i \in \mathbf{I}$ such that $F(d) \rightarrow_i F(d')$

A simple example is a program with no more control than the resource management of the system.

Example A non-controlled program for a system S is $P= <D,d_o,F,SR>$ where:

- $D = \{ (s_l,\ldots,s_n,R(s_l),\ldots,R(s_n)) \in S_l \times \ldots \times S_n \times Bag(Rs)^n \mid \Sigma_{i=1,\ldots,n} \mathcal{R}(si) \leq R\}$
- $d_o \in D$
- F is the identity function
- d SR d' if and only if d' is a global successor of d

Remark If D (defined in the example) is empty then no parallel program can be defined.

2.3 PROPERTIES OF PARALLEL PROGRAMS

We now express the semantics of a program by means of a computation tree.

Definition 2.4 (Computation Tree) Let SP be a system and let $P= <D,d_o,F,SR>$ be a program for this system. A computation tree (CT) of this program is a tree such that d_o is the root of the tree and d' is a son of d if and only if d SR d'.

If d' is a son of d , then $\exists\ i \in \mathbf{I}$ such that $F(d) \rightarrow_i F(d')$, what will be denoted by $d \rightarrow_i d'$.

What is relevant in the computation tree is the tree structure , the states of processes and the resources they own. So we can substitute to the nodes their image by F. Doing this, only the way the control is done is lost in the new tree. But this is irrelevant for the properties of the parallel program.

Definition 2.5 (Observational Computation Tree) Let $P= <D,d_o,F,SR>$ be a program for a system SP. An observational computation tree for this program is the mapping by F of the computation tree of P

Notation A *(finite, infinite) computation* of a program is a (finite, infinite) path in the computation tree of this program.

We introduce the properties that we want obtain for our program. These behavioral properties are defined over a computational tree of a given program. In the following, we will suppose that *SP* is a system, *P* is a program for this system and that CT is a computation tree for the program.

Definition 2.6 (Impartiality) A program *P* is impartial if for every infinite computation $(d_0,...,d_k,...)$ $\forall$ i $\in$ **I** $\forall$ j $\in$ **N**, $\exists$ k $\geq$ j such that $d_k \rightarrow_i d_{k+1}$

In other words, every process is infinitely executed in every infinite computation.

Definition 2.7 (Liveness) A program *P* is live if every finite computation is the prefix of an infinite computation.

In other words, every node in the computation tree has a successor

Definition 2.8 Let d be a node of the observational computation tree, then the i-choice of d, denoted by *ID*(d,i) is defined by : $ID(d,i) = \{ s_i \mid d \rightarrow_i (s_1,...,s_n, r_1,.. ,r_n) \}$

In other words, the i-choice of d is the possible successors states of P_i by an action of the process P_i.

Definition 2.9 (Independance) A program *P* ensures independance of processes of *SP* if for every i $\in$ I and every node d of the observational computation tree one has :
If there is a computation $d \rightarrow_{j1} d_1 \rightarrow_{j2} . . d_{k-1} \rightarrow_{jk} d_k$ with $\forall\ 1 \leq t \leq k\ \ j_t \neq i$
then there is a computation $d_k \rightarrow_{jk+1} ..\ d_{u-1} \rightarrow_{ju} d'$ with $\forall\ k < t \leq u\ j_t \neq i$
and with $ID(d',i) \supset ID(d,i)$

In other words if a process has, in some state, a possible choice of successors states then whatever the other processes can do, it is always possible for the process to have again this choice (without any move of it).

3 BUILDING OF AN IMPARTIAL PROGRAM

In order to impose impartiality to a program, it is necessary to avoid that a subset of processes take the monopoly of a computation at some stage of some computation

Example If we take $d_0 =(s_{1,0}, s_{2,0}, s_{3,0}, 0, 0, 0)$ for the non controlled program of the previous example we obtain a non-impartial program, because in its CT there are computations that never execute some processes, as shown in the figure 2. Indeed one can execute infinitely many times the process P_1 from point A, and so, this program is not impartial.

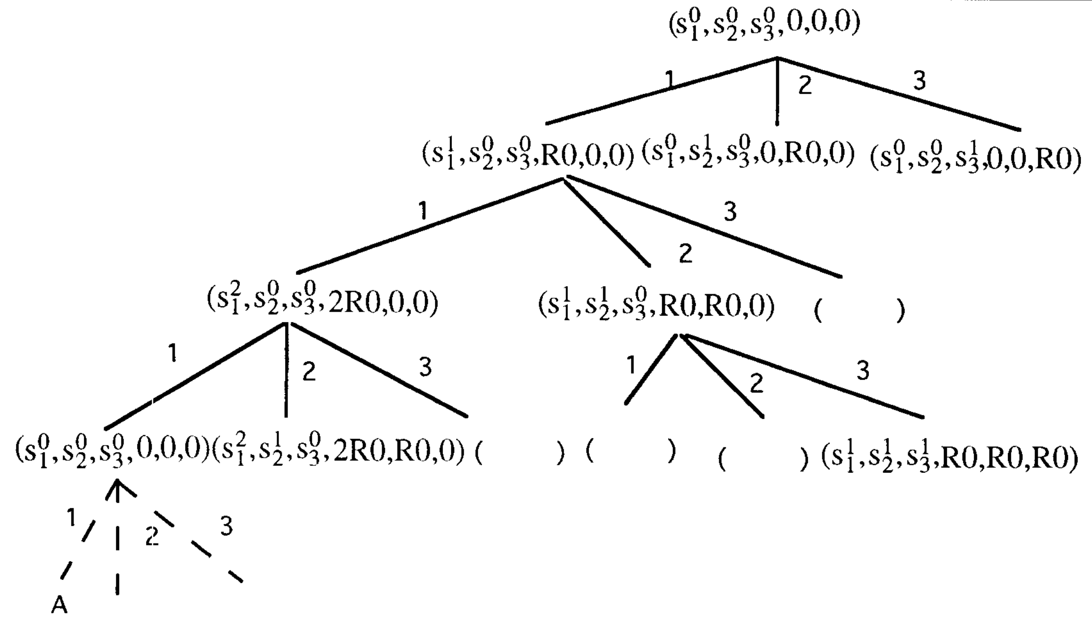

$(s_1^0,s_2^0,s_3^0,0,0,0)$
1
2
3
$(s_1^1,s_2^0,s_3^0,R0,0,0)$
$(s_1^0,s_2^1,s_3^0,0,R0,0)$
$(s_1^0,s_2^0,s_3^1,0,0,R0)$
1
2
3
$(s_1^2,s_2^0,s_3^0,2R0,0,0)$
$(s_1^1,s_2^1,s_3^0,R0,R0,0)$
( )
1
2
3
1
2
3
$(s_1^0,s_2^0,s_3^0,0,0,0)$
$(s_1^2,s_2^1,s_3^0,2R0,R0,0)$
( ) ( ) ( )
$(s_1^1,s_2^1,s_3^1,R0,R0,R0)$
1
2
3
A

Figure 2: A CT for the example

So, we need to limit the number of successive actions of the same process in a computation. It is clear that if in a computation σ a process acts infinitely many times, there exists at least a cycle in its automata such that σ passes infinitely many times over it. We can choose a set of states such that it contains at least a state of every cycle.

Definition 3.1 A set of control points of a process P is a minimal set of states such that it contains one state of every cycle of the process.

Remark Finding the set of control points subsets is a NP-complete problem [Kar72]. However for some simple kinds of automatas (which is often the case for those defined by a process) , one can show that the complexity of this search is polynomial.

We are ready to build an impartial program. In the sequel of the paper, we will suppose that E_i is a set of control points of the process P_i. At first, one organizes the processes on a virtual ring following their numbering and one associates a local variable for each process initialized to any number greater than zero. Each time a process leaves a control point it decreases its own variable and it increases the variable of its successor on the ring. A process can not leave a control point if its variable is equal to zero

Notations
Let $(v_1,...,v_n)$ be a vector (of states, resources or integers) indexed by $\mathbf{I}$, then one denotes $(v_1,...,v_n)$ by v.
Let $i \in \mathbf{I}$, then $i \oplus 1$ is defined by : $i \oplus 1$ = if $i<n$ then $i+1$ else 1

Definition 3.2 Let *SP* be a system of processes and $\{E_i\}_{i \in I}$ be a family of control points then *PI*(*SP*) = $<D, d_o, F, SR>$ is the program defined by :

- $D = \{ (s_1,...,s_n,R(s_1),...,R(s_n),e_1,...,e_n) \in S_1 \times ... \times S_n \times Bag(Rs)^n \times \mathbf{N}^n \mid \sum_{i=1,...,n} R(s_i) \leq R\}$

- $F(s_1,...,s_n,r_1,...,r_n,e_1,...,e_n) = (s_1,...,s_n,r_1,...,r_n)$
- $d_o = (s_0, r_0, e_0)$ such that $\forall\, i \in \mathbf{I}, e_{i,0} \neq 0\ \forall\, i \in \mathbf{I}, s_{i,0} \in Ei$
- Let d=(s,r,e) and d'=(s',r',e') be two items of *D* then d SR d' if and only if:

(i) $\exists\, i \in \mathbf{I}, (s,r) \rightarrow_i (s',r')$
(ii) $s_i \in E_i$, $e_i > 0$, $e_i' = e_i - 1$, $e'_{i\oplus 1} = e'_{i\oplus 1} + 1$ and $\forall j \neq i$ and $j \neq i\oplus 1$, $e_j' = e_j$
OR
$s_i \notin E_i$, $\forall\, j \in \mathbf{I}$, $e_j' = e_j$

Remark The incrementation of the variable of the successor can be replaced by the sending of a message (in the case of a safe medium). On reception of the message the process increases its local variable. The behaviour of the new program will be the same as the old one but with more intermediate states (when the messages are in transit). Doing this way, the control is distributed.

Proposition 3.1 *PI*(*SP*) is an impartial program.

Proof
Let $\sigma = (d_0,...,d_m,...)$ be an infinite computation of *PI*(*SP*). Let **J** be the subset of processes infinitely executed in σ. Let $i_0 \in \mathbf{I} \backslash \mathbf{J}$, and d_m be a state of the computation from where the process P_{i0} is no more executed. Thus the variable $e_{i0\oplus 1}$ is no more incremented from d_m.

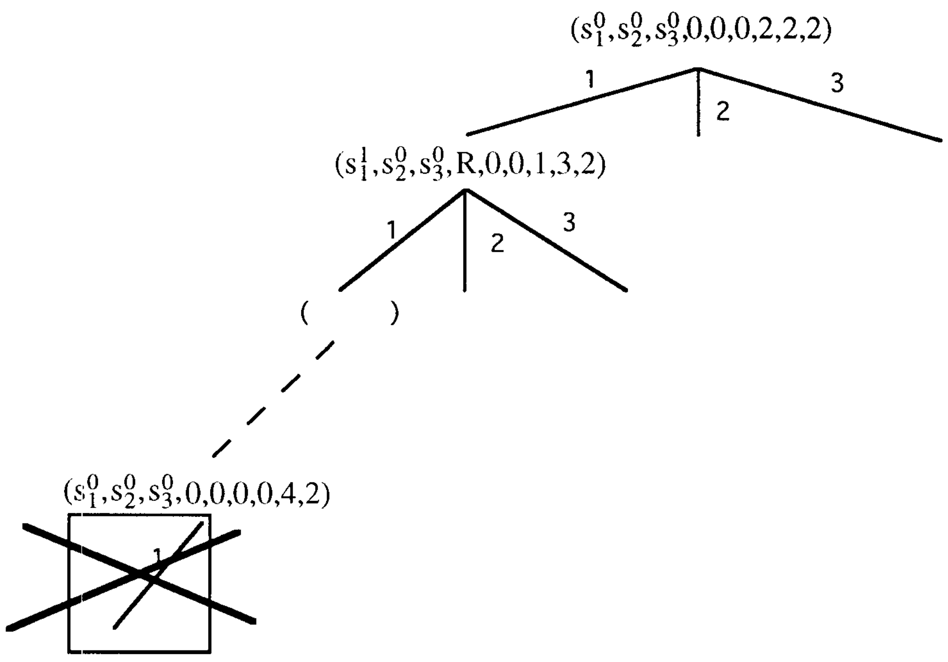
If $i0\oplus 1 \in \mathbf{J}$ then the computation must leave an infinite number of times a control point of the process $P_{i0\oplus 1}$ (because of the cut property of control points). Hence the variable $e_{i0\oplus 1}$ is infinitely decremented. It is contradictory with the control associated to *PI*(*SP*). *So* $i_0\oplus 1 \in \mathbf{I} \backslash \mathbf{J}$.

Thus either $\mathbf{J} = \mathbf{I}$ or $\mathbf{J} = \emptyset$ but as σ is an infinite computation $\mathbf{J} = \mathbf{I}$ ◊◊◊

Example In the impartial program, for our example starting from the initial element

$d_0=(s_{1,0},s_{2,0},s_{3,0},0,0,0,2,2,2)$ and taking $E_i= \{s_{i,0}\}$ $i \in \{1,2,3\}$, figure 3 shows where the unfair

computation of figure 2 is cut.



$(s_1^0,s_2^0,s_3^0,0,0,0,2,2,2)$
1
2
3
$(s_1^1,s_2^0,s_3^0,R,0,0,1,3,2)$
1
2
3
( )
$(s_1^0,s_2^0,s_3^0,0,0,0,0,4,2)$
1

Figure 3: A CT for the Program *PI(SP)* of the example

4. BUILDING OF A DEADLOCK-FREE PROGRAM

Introducing impartiality in the preceeding section does not solve the deadlock problem as shown in the example below. However the program that we are going to build adds control to the impartial program. Hence no new infinite computations are created and then impartiality is preserved.

Example In the previous impartial program we can reach the node $(s_{1,1},s_{2,1},s_{3,1},R0,R0,R0,l,l,l)$ which is a deadlock (i.e. the program has no successor from this node).

Our next program is based on two main ideas :

-Let us consider control points of the process as equivalent to "idle" states. We require prior to the building of our program that when all the processes except one are idle, the "active" process can move freely from one state to another one.

-Under this assumption, a trivial way to avoid deadlock should be that a process, leaving a control point, takes all the resources needed to reach any of the closest control points. However this solution would be a hidden sequentialization of the processes. A sufficient and much less restrictive condition is that "***in order to do its next move the process requires the resources needed to reach the closest control points to be free but doing this move, it just takes the resources needed for its next state***".

We now introduce our basic hypothesis:

Basic Hypothesis

$\forall$ i $\in$ **I**, $\forall$ $s_i \in E_i$, $R(S_i) + \sum_{j\neq i} R(E_j) \leq R$

Discussion It is easy to show that this condition is fulfilled by almost all the typical systems of processes. Indeed for a repetitive process, the initial state is a cut and moreover in this state the process does not own any resource. Hence the inequality becomes $R(S_i) \leq R$ which must be fulfilled otherwise a state of S_i requires more resources of some kind than the system owns and thus P_i is an inconsistent specification.

Definition 4.1 Let s_i be a state of S_i , then the set of extended neighours of s_i (relatively to E_i), denoted s_i^+ , is defined by :

$s_i^+ = \{ s \mid \exists\ v_0 .v_1 \ldots v_n$ a (possibly empty) path of P_i

with $v_0 = s_i$, $v_n = s$ and $\forall\ 0 < k < n\ v_k \notin Ej \}$

Informally, s_i^+ is the set of states reachable from s_i without crossing a state of E_i .

Notation We will denote by $R^+(s)$ the function defined by $R(s^+)$

Fact 4.1 Let s_i' be a successor of s_i in P_i with $s_i' \notin E_i$ then $R^+(s_i') \leq R^+(s_i)$. Obvious since $s_i'^{\,+} \subset s_i^+$.

In our next program a state needs a little bit more resources than the ones in the specification, i.e. we require that a process always keeps the resources needed at its control points. As already stated, this condition collapses in "standard" processes systems.

Definition 4.2 Let R be the resource function of a systeme process, then $\mathbb{R}$ is the "extra-resource" function (relatively to E_i) defined by :

$\forall\ s \in E_i\ \mathbb{R}(s)=\mathrm{Sup}(R(s),R(E_i))$

We are ready to present our deadlock-free program.

Definition 4.3 Let *SP* be a system of processes and $\{E_i\}_{i \in \mathbf{I}}$ be a family of control points, then *Pdf*(*SP*) = <*D*,d_0,F,SR> is the program defined by :

- *D* = $\{(s_1,\dots,s_n,\mathbb{R}(s_1),\dots,\mathbb{R}(s_n),e_1,\dots,e_n) \in S_1 \times \dots \times S_n \times \mathrm{Bag}(Rs)^n \times \mathbf{N}^n \mid \sum_{i=1,\dots,n} \mathbb{R}(si) \leq R\}$
- $F(s_1,\dots,s_n,r_1,\dots,r_n,e_1,\dots,e_n) = (s_1,\dots,s_n,r_1,\dots,r_n)$
- $d_o = (s_0, r_0, e_0)$ such that $\forall\ i \in \mathbf{I}, e_{i,0} \neq 0\ \forall\ i \in \mathbf{I}, s_{i,0} \in Ei$
- Let d=(s,r,e) and d'=(s',r',e') be two items of *D* then d SR d' if and only if:

(i) $\exists\ i \in \mathbf{I}$, $(s,r) \rightarrow_i (s',r')$

(ii) $s_i \in E_i$, $e_i > 0$, $e_i' = e_i - 1$, $e'_{i\oplus 1} = e'_{i\oplus 1} + 1$ and $\forall j \neq i$ and $j \neq i\oplus 1$, $e_j' = e_j$
OR
$s_i \notin E_i$, $\forall\ j \in \mathbf{I}$, $e_j' = e_j$

(iii) If $s_i' \notin E_i$ then $\sum_{j\neq i} r_j + \mathbb{R}^+(s_i') \leq R$

It is important to notice here that the policy imposed by the program *Pdf*(*SP*) in condition (iii) can be implemented just with the local knowledge and the knowledge of the multiset of non-engaged resources. Indeed if we denote R_s this multiset then:
(iii) $\Leftrightarrow \mathbb{R}^+(s_i') \leq \mathrm{Sup}(R_s, \mathbb{R}(s_i))$

At first we characterize two inductive invariants of *Pdf*(*SP*).

Lemma 4.1 Let *SP* be a system of processes and *Pdf*(*SP*) its associated program, let (s,r,e) be a reachable state of *Pdf*(*SP*), then: $\sum_{i \in \mathbf{I}} e_i = \sum_{i \in \mathbf{I}} e_{i,0}$ (> 0 by definition)

Proof
By induction: initially it is a tautology and in a state transition either all the e_i's are unchanged or one of them is decremented and another one is incremented. ◊◊◊

In order to prove our next invariant, we need some new definitions.

Definition 4.4 Let *SP* be a system of processes and *Pdf*(*SP*) its associated program,
let $\sigma = d_0 \rightarrow_{i1} d_1 \rightarrow_{i2} \dots \rightarrow_{ik} d_k$ be a computation sequence, with $d_k = (s_k,r_k,e_k)$ then we partition the set **I** in two sets :

-$\mathbf{I}_{1,k} = \{j \mid s_{k,j} \in E_j\}$, $\mathbf{I}_{2,k} = \mathbf{I} \setminus \mathbf{I}_{1,k}$

Moreover we order $\mathbf{I}_{2,k}$ by "the more recent move" in σ :

- Let $l,m \in \mathbf{I}_{2,k}$ then $l <_k m \Leftrightarrow \max\{j \mid i_j = l\} < \max\{j \mid i_j = m\}$.

Now we present the key lemma for deadlock freeness of the program *Pdf*(*SP*).

<u>Lemma 4.2</u> Let *SP* be a system of processes and *Pdf*(*SP*) its associated program,
let $\sigma = d_0 \rightarrow_{i1} d_1 \rightarrow_{i2} \ldots \rightarrow_{ik} d_k$ be a computation sequence, with $d_k = (s_k, r_k, e_k)$ then:

(x) $\forall\, l \in \mathbf{I}_{2,k}\ \ \sum_{j \in \mathbf{I}1,k} \mathbb{R}(E_j) + \sum_{j \in \mathbf{I}2,k\ \&\ j <_k l} r_{j,k} + \mathbb{R}^+(s_{l,k}) + \sum_{j \in \mathbf{I}2,k\ \&\ l <_k j} \mathbb{R}(E_j) \ \leq R$

Proof By induction, in the initial state all processes are "in" E_i, so $\mathbf{I}_{2,k} = \emptyset$ and there is nothing to prove.
Now let us suppose that this invariant relation is verified for a path of length k, and that $d_k \rightarrow_i d_{k+l}$.
We will distinguish different cases. Recall that:
$d_k = (s_{1,k}, \ldots, s_{n,k}, r_{1,k}, \ldots, r_{n,k}, e_{1,k}, \ldots, e_{n,k})$
and $d_{k+l} = (s_{1,k+1}, \ldots, s_{n,k+1}, r_{1,k+1}, \ldots, r_{n,k+1}, e_{1,k+1}, \ldots, e_{n,k+1})$

Case 1 $i \in \mathbf{I}_{1,k}$, $s_{i,k+l} \in E_i$ (i.e. P_i leaves one of its control point for another one)
It implies that $\mathbf{I}_{1,k+1} = \mathbf{I}_{1,k}$ and $\mathbf{I}_{2,k+1} = \mathbf{I}_{2,k}$ and the order of $\mathbf{I}_{2,k+1}$ is like the one of $\mathbf{I}_{2,k}$ .Moreover all the inequalities *(x)* are unchanged.

Case 2 $i \in \mathbf{I}_{1,k}$, $s_{i,k+l} \notin E_i$ (i.e. P_i leaves one of its control point for a state which is not a control point)
It implies that $\mathbf{I}_{1,k+1} = \mathbf{I}_{1,k} \setminus \{i\}$ and $\mathbf{I}_{2,k+1} = \mathbf{I}_{2,k} \cup \{i\}$ and the order of $\mathbf{I}_{2,k+1}$ is extended from the one of $\mathbf{I}_{2,k}$ with $j <_{k+l} i$ for all $j \neq i$ belonging to $\mathbf{I}_{2,k+1}$. Let us have a look on the new inequalities:

-$\forall\, j \neq i \in \mathbf{I}_{2,k+1}$,we get the new inequality by deleting in the first sum the i^th^ term $\mathbb{R}(E_i)$ but adding this term to the last sum (as $j <_{k+l} i$). Then the new inequality is unchanged.
-For i , we have to establish the new inequality defined by :
(y) $\sum_{j \in \mathbf{I}1,k+1} \mathbb{R}(E_j) + \sum_{j \in \mathbf{I}2,k+1\ \&\ j \neq i} r_{j,k} + \mathbb{R}^+(s_{i,k+1}) \leq R$
But taking into account the SR relation and the domain *D* of the program *Pdf*(*SP*), we have:
$\forall\, j \in \mathbf{I}_{1,k+1}, \mathbb{R}(E_j) = r_{j,k}$ since $\mathbf{I}_{1,k+1} \subset \mathbf{I}_{1,k}$,
$\forall\, j \in \mathbf{I}_{2,k+1}$, , $j \neq i$: , $r_{j,k+1} = r_{j,k}$ since the process j has not moved during this step.
Thus (y) becomes :
$\sum_{j \neq i} r_{j,k} + \mathbb{R}^+(s_{i,k+1}) \leq R$
which is exactly the condition (iii) of *Pdf*(*SP*).

Case 3 $i \in \mathbf{I}_{2,k}$, $s_{i,k+l} \in E_i$ (i.e. P_i reaches one of its control point from a state which is not a control point)
It implies that $\mathbf{I}_{1,k+1} = \mathbf{I}_{1,k} \cup \{i\}$ and $\mathbf{I}_{2,k+1} = \mathbf{I}_{2,k} \setminus \{i\}$ and the order of $\mathbf{I}_{2,k+1}$ is the restriction of the one of $\mathbf{I}_{2,k}$.
Let us examine the new inequalities (x) :
$\forall\, j \in \mathbf{I}_{2,k+1}$, , $j <_k i$: the new inequality is obtained from the old one by substituting the term $\mathbb{R}(E_i)$ of the last sum by the same term in the first sum. Hence nothing is changed.

$\forall\, j \in \mathbf{I}_{2,k+1}$, , $i <_k j$: the new inequality is obtained from the old one by substituting the term $r_{i,k}$ of the second sum by the term $\mathbb{R}(E_i)$ in the first sum. But by definition of *D*:
$r_{i,k} = \mathbb{R}(s_{i,k}) = \mathrm{Sup}\,(R(s_{i,k}), R(E_i)) \geq R(E_i) = \mathbb{R}(E_i)$
Hence the left term of the inequality is decreased and the inequality remains true.

Case 4 $i \in \mathbf{I}_{2,k}$, $s_{i,k+1} \in E_i$ (i.e. P_i moves from one state to another one, neither are control points) It implies that $\mathbf{I}_{1,k+1} = \mathbf{I}_{1,k}$ and $\mathbf{I}_{2,k+1} = \mathbf{I}_{2,k}$ and the order of $\mathbf{I}_{2,k+1}$ is unchanged for items different from i and $j <_{k+1} i$ for all $j \neq i$ belonging to $\mathbf{I}_{2,k+1}$.Let us examine the new inequalities (x) :

- $\forall$ j E $\mathbf{I}_{2,k+1}$ and $j <_k i$ the new inequality is the same as the old one.
- $\forall$ j E $\mathbf{I}_{2,k+1}$ and $i <_k j$ the new inequality is obtained from the old one by substituting the term $r_{i,k}$ of the second sum by the term $\mathbb{R}(E_i)$ in the third sum. As in case 3, the left term of the inequality is decreased and the inequality remains true.
- Since $j <_{k+1} i$ for all $j \neq i$ belonging to $\mathbf{I}_{2,k+1}$, the new inequality (x) for i is defined by :

(y) $\sum_{j \in I1,k+1} \mathbb{R}(E_j) + \sum_{j \in I2,k+1 \,\&\, j\neq i} r_{j,k} + \mathbb{R}^+(s_{i,k+1}) \leq R$

As in case 2 , one can reduce this inequality to the condition (iii) of *Pdf*(*SP*).

◊◊◊

<u>Proposition 4.1</u> *Pdf*(*SP*) is an impartial and live program and ensures independance of processes of *SP.*

Proof

.Impartiality**:** As every computation of *Pdf*(*SP*) is a computation of *PI*(*SP*) , using proposition 3.1 , we can conclude.

.Liveness: Let $d_k = (s_k,r_k,e_k)$ be a reachable state of *Pdf*(*SP*). We have to prove that d_k has a successor for the SR relation of *Pdf*(*SP*) .Two cases happen:

- $\mathbf{I}_{2,k} \neq \emptyset$, let i be the maximal item of $\mathbf{I}_{2,k}$ for the $<_k$ relation. The inequality (x) for i is defined by :

$\sum_{j \in I1,k} \mathbb{R}(E_j) + \sum_{j \in I2,k \,\&\, j\neq i} r_{j,k} + \mathbb{R}^+(s_{i,k}) \leq R$

which is equivalent to

$\sum_{j\neq i} r_{j,k} + \mathbb{R}^+(s_{i,k}) \leq R$

Let s' be a successor of $s_{i,k}$ in P_i (there is at last one since P_i is strongly connected and $|S_i| > 1$).

<u>either</u> $s' \notin E_i$, thus from the fact 4.1 $\mathbb{R}^+(s') \leq \mathbb{R}^+(s_{i,k})$, implying $\sum_{j\neq i} r_{j,k} + \mathbb{R}^+(s') \leq R$ and hence $(s_{1,k},\ldots,s',\ldots,s_{n,k},r_{1,k},\ldots\mathbb{R}(s'),\ldots,r_{n,k},e_{1,k},\ldots,e_{n,k})$ is a successor of d_k.

<u>or</u> $s' \in E_i$, since $\mathbb{R}(s') = \mathbb{R}(E_i) \leq \mathbb{R}(s_{i,k})$ and again

$(s_{1,k},\ldots,s',\ldots,s_{n,k},r_{1,k},\ldots\mathbb{R}(s'),\ldots,r_{n,k},e_{1,k},\ldots,e_{n,k})$ is a successor of d_k.

$\mathbf{I}_{2,k} = \emptyset$, by lemma 4.1 there is some $e_{i,k} > 0$

Let s be any successor of $e_{i,k}$ in P_i

$R(s) \leq \mathbb{R}(s) \leq \mathbb{R}^+(s) \leq R(S_i)$. So using the basic hypothesis, s provides a successor of d_k for SR.

. Independance : left to the final paper.

◊◊◊

<u>5 CONCLUSION</u>

We have presented an algorithm which takes as input a set of automatas specifying processes with resources requirement and which produces a fair and live program for these processes. Let us point out some features of our method.

It just handles a particular but frequent case of specification. However the specification language is much expressive than the other ones for this problem (e.g. try to describe in a concise way a multiset of resources requirement with propositional temporal logic !). If the basic hypothesis (a very weak condition) is fulfilled then it always produces a program.

Moreover the program is parametrized, i.e. the initial values of the e_i's determine the degree of parallelism between processes one wants to allow. With the assumption of an existing resource allocation mechanism, the program can be easily distributed since the communication of a process is limited to sending messages to a "successor" process and asking resources to the resource management. This resource management does not need to include an algorithm for deadlock prevention since the local policy of processes is enough. At last if a new process wants to be added, the only thing to do is to test the basic hypothesis and to insert the new process when all the old processes come back to a control point.

Possible extensions of this work should be to relax the constraint about the strong connexion of automatas or to enlarge the specification with other problems than the resource management as for instance processes cooperation via messages.

REFERENCES

[Bro84] S.D. Brookes, C.A.R. Hoare, A.W. Roscoe "A theory of communicating sequential processes". JACM 31 560-599. 1984.

[Eme83] A. Emerson, E. Clarke "Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons", Science of Computer Programming 2 p.241-266. 1983.

[Hab69] A.N Haberman "Prevention of System Deadlocks". Communications of the ACM, Vol. 12 (7) p.373-377 and 385. July 1969.

[Hoa78] C. A. R. Hoare "Communicating Sequential Processes". Communications of the ACM, Vol. 21(8) : p. 666-677 and 385. August 1978

[Kar72] R. M. Karp "Reducibility among combinatorial problems" in Complexity of Computer Computations edited by R.E. Miller and J.W. Tatcher. p.85-104 Plenum Press, New York. 1972

[Man82] Z. Manna, P. Wolper " Verification of communication processes from temporal logic specifications" Proceedings of workshop on Logics of Programs, Yorktowns Heights, New-York, May 1981 Springer-Verlag L.N.C.S Vol. 31 1982.

[Old85] E. R. Olderog "Specification-oriented programming in TCSP". in Logics and Models of Concurrent Systems. edited by K. R. Apt NATO ASI Series. Series F , Vol. 13. Springer-Verlag 1985.

[Sis85] A. P. Sistla , E.M. Clarke "Complexity of propositional temporal logic" *JACM* 32(3) , 1985 pp 733-749