RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

IRÈNE GUESSARIAN
WAFÂA NIAR-DINEDANE

Fairness and regularity for SCCS processes

RAIRO. Informatique théorique et applications, tome 23, n° 1 (1989), p. 59-86

<http://www.numdam.org/item?id=ITA_1989__23_1_59_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 23, n° 1, 1989, p. 59 à 86)

FAIRNESS AND REGULARITY FOR SCCS PROCESSES (*)

by Irène GUESSARIAN (1) and Wafâa NIAR-DINEDANE (2)

Abstract. – *We describe various kinds of fairness (mainly weak and strong fairness) for finite state* SCCS *processes by providing an automaton-theoretic characterization of the classes of fair languages. To this end, we introduce a variant of Muller automata, the T-automata, which still recognize the class of ω-regular languages, and which characterize the classes of fair languages.*

Résumé. – *Nous décrivons divers types d'équités (principalement l'équité forte et l'équité faible) pour les processus SCCS ayant un nombre fini d'états. Nous caractérisons les classes de langages équitables au moyen d'automates finis. Pour ce faire, nous introduisons une variante des automates de Muller, les T-automates, qui reconnaissent aussi les langages ω-réguliers, et qui caractérisent les classes de langages équitables.*

1. INTRODUCTION

Fairness is both a very complex and widely investigated subject [11]. The present paper is a contribution to the theory of fairness for Synchronous CCS, or SCCS, with delay operators. In SCCS, the loose synchronization of CCS is replaced by the tight synchronization operator $\times$, requiring that all individual processes which are composed via $\times$ take a step together at all time units [18, 19]. Whence the need, if we wish to allow for more flexibility and avoid some deadlocks, to introduce a delay operator enabling some processes to wait for some time, until e. g. the environment allows them to proceed. This in turn creates fairness problems.

Roughly speaking, fairness ensures that no process shall wait forever. More precisely we will mainly be concerned with strong fairness [8], requiring that every process which is enabled, i. e. allowed to pursue its computations, infinitely often, shall perform effective actions infinitely often. Transition systems are now acknowledged to be one of the best models for parallelism

(*) Support from the PRC Mathématiques-Informatique is gratefully acknowledged.
(1) C.N.R.S.-L.I.T.P., Université Paris-7, 2, place Jussieu, 75251 Paris Cedex 05, France.
(2) L.I.T.P., Université Paris-7, 2, place Jussieu, 75251 Paris Cedex 05, France.

Informatique théorique et Applications/Theoretical Informatics and Applications
0296-1598 89/01 59 28/$4.80/

[2, 4, 21, 28, 32]. Transition systems can be considered as automata skeletons, thus it seems quite natural to try to characterize fairness in terms of successful computations of automata. Surprisingly, up to now and to our knowledge, very few people have been trying similar approaches [24, 28, 29].

We show how to characterize fair computations of some finite state SCCS processes via the successful computations of a variant of Muller automata, namely the Muller automata with infinitary transitions instead of infinitary states. Our proof is effective in the sense that, starting from an SCCS process, we construct effectively the automaton which recognizes the fair computations of that process. We show that Muller automata with infinitary transitions still recognize the class of ω-regular languages as the usual Muller automata. This implies that the class of fair computations of a finite state SCCS process is contained in the class of ω-regular languages; we show that it coincides with the class of ε-free ω-regular languages. Besides providing a nice operational characterization of fair languages, we believe that our approach sheds a new light and gives more insight into the phenomenon of fairness. Our approach differs from the one of [29] in the following respects: (i) they introduce a general notion of fairness for *all* automata with a special acceptance condition, whereas we consider only the automata corresponding to SCCS processes, with an acceptance condition which is equivalent to the usual one, and (ii) they require that all edges (or all edges with a given label) be taken infinitely often in the course of a fair computation, whereas we require that a set of specific edges together with specific labels, be taken infinitely often, and we do not require for all arbitrary edges to be taken infinitely often.

The results of the present paper generalize those which were presented at STACS 88, where we had considered only a subclass of finite state processes, namely the strictly regular processes.

The present paper contains 3 more sections: section 2 describes the language and processes that we will study, section 3 recalls the necessary prerequisites about automata and introduces T-automata, and finally section 4 explains our results about fairness.

2. SCCS AND ITS SEMANTICS

2.1. The syntax

We will work with the language SCCS of [18, 14]. Let $Act = \langle A, ., ^{-}, 1 \rangle$ be a non empty commutative group of actions, and Var a set of variables. The unit action 1 represents an internal action, for instance the result of a

synchronization $a\bar{a}$, or a delay of one time unit. The SCCS expressions **E**, ranged over by E, are defined by the BNF scheme:

$$E ::= x \mid NIL \mid a:E \mid E \uparrow B \mid E+F \mid E \times F \mid \mathrm{rec}\,\vec{x}.\vec{E}$$

where $x \in Var$, $a \in A$, $B \subseteq A$, E, $F \in \mathbf{E}$. We will omit in the sequel the vector notation and shorten $\mathrm{rec}\,\vec{x}.\vec{E}$ into $\mathrm{rec}\,x.E$.

An occurrence of a variable x in an expression E is said to be *free* if it is not in the scope of a $\mathrm{rec}\,x$, and it is said to be *guarded* [19] if it occurs within a subexpression of the form $a:F$. An *SCCS process* is an expression without free variables; the set **P** of SCCS processes is ranged over by p.

NIL represents the process which can do nothing, : represents sequential composition, + represents nondeterminism, × represents the synchronous product of two processes performing actions simultaneously. $E \uparrow B$ represents the restriction of E where only actions in B can be performed. $\mathrm{rec}\,\vec{x}.\vec{E}$ represents the solution of the set of mutually recursive equations $x_i = E_i$, $i = 1, \ldots, n$. The *delay* operator δ is definable in that framework via $\delta E = \mathrm{rec}\,x(1\,x + E)$. See [14] for more details. The sequential composition : will be elided whenever this causes no ambiguity.

2.2. Operational semantics

As usual, the derivation relation: $E \xrightarrow{a} F$, means that E becomes F after performing action a, and defines the operational semantics of processes. $\xrightarrow{a}$ is defined inductively on **E** as the least relation containing, for all $a \in A$:

$a:E \xrightarrow{a} E$;

if $E \xrightarrow{a} E'$ then $E \uparrow B \xrightarrow{a} E' \uparrow B$ for all $a \in B \subseteq A$;

if $E \xrightarrow{a} E'$ then $E+F \xrightarrow{a} E'$ and $F+E \xrightarrow{a} E'$;

if $E \xrightarrow{a} E'$ and $F \xrightarrow{b} F'$ then $E \times F \xrightarrow{a.b} E' \times F'$ where $a.b = 1$ if $b = \bar{a}$;

if $E[\mathrm{rec}\,x.E/x] \xrightarrow{a} E'$ then $\mathrm{rec}\,x.E \xrightarrow{a} E'$;

finally, $\delta E \xrightarrow{1} \delta E$ and if $E \xrightarrow{a} E'$ then $\delta E \xrightarrow{a} E'$.

The rules concerning δ are deducible from the previous ones and are given for convenience only. Since δ is definable using rec, it is not a primitive operator in SCCS, and all the constraints concerning rec will apply to δ.

The derivatives of E are the E''s such that there exists a derivation c:

$$c: \quad E \overset{a_1}{\to} E_1 \overset{a_2}{\to} \ldots \overset{a_n}{\to} E'.$$

The set of derivatives of E is denoted by $Der(E)$. A computation c of a process p is a maximal (finite or infinite) derivation; the sequence of actions $a_1 a_2 \ldots$ executed by p in the course of computation c is called a *trace* of p, and denoted by $trace(c)$.

A process E is said to be *finite state* iff it has a finite number of derivatives, namely $Der(E)$ is finite. Finite state processes will also be called *regular* processes; this terminology stems from the fact that the (fair) trace languages of finite state processes will be shown to be ω-regular languages (*see* sections 2 and 4). The set of finite state (or regular) processes will be denoted by **R**.

A process E is said to be *strictly regular* iff all its subprocesses of the form $\mathrm{rec}\, x.F$ satisfy the following two restrictions: (i) all occurrences of x in F are guarded, i. e. in a subexpression of the form $a: F'$, and (ii) F has no occurrence of the synchronous product $\times$. The set of strictly regular processes will be denoted by **RP**.

A strictly regular process has a finite number of derivatives, and we will associate with it finite automata which recognize its set of traces and fair traces; however, not all finite state processes are strictly regular, e. g. $\mathrm{rec}\, x.(ax + bNIL \times cNIL)$. Strict regularity "strictly" implies that the process is finite state and that there is no dynamic generation of subprocesses and derivatives, whence the terminology.

Since we are interested in fairness, we need more information than just the name of the action performed during one derivation step, as shown by the following example:

$$\mathrm{rec}\, x.\delta ax \times \mathrm{rec}\, x.\delta \bar{a}x \overset{1}{\to} \mathrm{rec}\, x.\delta ax \times \mathrm{rec}\, x.\delta \bar{a}x.$$

The action 1 can here proceed:

— either from the product of the two delays: $\mathrm{rec}\, x.\delta ax \overset{1}{\to} \mathrm{rec}\, x.\delta ax$ and $\mathrm{rec}\, x.\delta \bar{a}x \overset{1}{\to} \mathrm{rec}\, x.\delta \bar{a}x$;

— or from the product of the actions a and $\bar{a}$: $\mathrm{rec}\, x.\delta ax \overset{a}{\to} \mathrm{rec}\, x.\delta ax$ and $\mathrm{rec}\, x.\delta \bar{a}x \overset{\bar{a}}{\to} \mathrm{rec}\, x.\delta \bar{a}x$.

Hennessy [14], differing slightly from the approaches of [7, 9] to model fairness, defined the set R of *action-redexes* by the BNF-scheme:

$$r ::= a \mid \delta \mid \delta r \mid \uparrow r \mid +1\, r \mid +2\, r \mid \mathrm{rec}.\, r \mid \langle r_1, r_2 \rangle$$

where $a \in A$, $r, r_1, r_2 \in R$.

$\xrightarrow{r}$ is defined inductively on $\mathbf{E}$ as the least relation containing, for all $a \in A$ and $r \in R$:

$a: \mathrm{E} \xrightarrow{a} \mathrm{E}$;

if $E \xrightarrow{r} E'$ and $name(r) \in B$ [*see* below the definition of $name(r)$] then:

$E \uparrow B \xrightarrow{\uparrow r} E' \uparrow B$;

if $E \xrightarrow{r} E'$ then $E + F \xrightarrow{+1\, r} E'$ and $F + E \xrightarrow{+2\, r} E'$;

if $E \xrightarrow{r} E'$ and $F \xrightarrow{r'} F'$ then $E \times F \xrightarrow{\langle r, r' \rangle} E' \times F'$;

if $E[\mathrm{rec}\, x.\, E/x] \xrightarrow{r} E'$ then $\mathrm{rec}\, x.\, E \xrightarrow{\mathrm{rec}\, r} E'$,

finally, $\delta E \xrightarrow{\delta} \delta E$, and moreover, if $E \xrightarrow{r} E'$ then $\delta E \xrightarrow{\delta r} E'$.

Let *name*: $R \to A$ be the function defined by:

$$name(a) = 1, \text{ for all } a \in A,$$
$$name(\delta) = 1,$$
$$name(ur) = name(r), \text{ for all } u \in \{\delta, \uparrow, \mathrm{rec}, +1, +2\}^*,$$
$$name(\langle r, r' \rangle) = name(r).\, name(r').$$

PROPOSITION 2.1: *For any process $p \in \mathbf{P}$, and action-redex $r \in R$, if $p \xrightarrow{r} p'$ and $p \xrightarrow{r} p''$, then $p' = p''$.*

So, the introduction of action-redexes determinizes the behavior of processes, and will help in the study of fairness. However, the formalism being somehow heavy, we will omit the symbols δ, $\uparrow$, rec, $+1$, $+2$ whenever possible.

Finally, for $r \in R$, define π_i, $i = 1, 2$, by:

if $r = u \langle r_1, r_2 \rangle$ with $u \in \{\delta, \uparrow, \mathrm{rec}, +1, +2\}^*$, then: $\pi_i(r) = r_i$ for $i = 1, 2$, otherwise $\pi_i(r) = \perp$, i. e. is undefined for $i = 1, 2$.

2.3. Bisimulation

In the literature, various authors have defined equivalence relations which identify processes having the same observational behavior [3, 9, 14, 16, 19, 27]. See [5, 12, 31], for a survey and comparison between these various equivalences. Most of them, however, are too weak, i.e. identify too many processes for our purposes. We shall work here with the notion of *bisimulation*, introduced by Park [27, 28], *see* also [18, 20].

DEFINITION 2.1: A relation $B \subseteq \mathbf{P}^2$ is a *bisimulation* if, whenever $p\,B\,q$ and $a \in A$:

(i) if $p \xrightarrow{a} p'$ then, for some q', $q \xrightarrow{a} q'$ and $p'\,B\,q'$;

(ii) if $q \xrightarrow{a} q'$ then, for some p', $p \xrightarrow{a} p'$ and $p'\,B\,q'$.

PROPOSITION 2.2: [18] *There exists a maximum bisimulation, denoted by* $\approx$, *and such that:*

(i) $\approx$ *is an equivalence relation;*

(ii) $\approx$ *is preserved by all the* SCCS *operators, and*

(iii) $\approx$ *is consequently a congruence.*

3. TRANSITION SYSTEMS AND AUTOMATA

We will model the behavior of finite state processes by finite automata; to this end, we first describe more generally the behavior of all SCCS processes via transition systems [6, 21, 32].

3.1. The syntax

DEFINITION 3.1: Let A be an alphabet and *Var* a set of variables; a *transition system* S is a triple $S=(Q, s, D)$, where: Q is a nonempty set of nodes, $s \in Q$ is the start node, and $D \subseteq Q \times A \times Q$ is the set of derivations, or transitions.

A transition system S is finite iff Q and D are finite.

We will use the notations:

$\text{Out}\,(q) = \{\, a \in A / \exists\, q' \in Q,\ (q, a, q') \in D \,\}$,

$D\,(q) = \{\, (a, q') / (q, a, q') \in D \,\}$, the derivations of q.

3.2. Algebra of transition systems and behaviors

DEFINITION 3.2: Let p be an SCCS process over an alphabet of actions A, and let R be the corresponding set of action redexes. A transition system $S=(Q, s, D)$ over the alphabet A (resp. R) is said to model the behavior of p, and we write $S=\mathscr{C}(p)$ [resp. $S=\mathscr{B}(p)$] iff: (i) $Q=Der(p)$, (ii) $s=p$, and (iii) $(q_1, a, q_2)\in D$ iff $q_1=p_1$, $q_2=p_2$, and $p_1 \overset{a}{\rightarrow} p_2$.

The alphabet of S will be R if we are interested in fairness, and A if we are only interested in the trace language. We identify (a, b) in A^2 with $\langle a, b \rangle$ in R (resp. with $a.b\in A$).

NOTATION: In the sequel, and unless otherwise specified, we will write $\mathscr{C}(p)$ for both $\mathscr{C}(p)$ and $\mathscr{B}(p)$, because the results concerning these two kinds of transition systems are the same.

This notion of modelling is adequate, because the transition systems are naturally endowed with an algebra structure, *see* [21] for the definitions of $+$, **0** and prefixing on charts, which are generalized transition systems.

PROPOSITION 3.1: *Let $S=(Q, s, D)$ be a transition system over A, and p a finite state* SCCS *process such that $S=\mathscr{C}(p)$, then S is a finite automaton.*

Proof: If the process is finite state, then it has a finite number of derivatives, hence the corresponding transition system is finite, namely is a finite automaton.

Proposition 3.1 becomes false when $S=\mathscr{B}(p)$, because in this latter case, S need not necessarily be finite, even if p is finite state. Consider for instance $p=\mathrm{rec}\, x.(ax+x)$; the corresponding transition system S on R such that $S=\mathscr{B}(p)$ has a single state, but infinitely many transitions.

For a process p and a transition system C such that $C=\mathscr{C}(p)$, the states of the transition system are the derivatives of the process, and its transitions, or derivations, are:

– either the actions which can be taken by the process, if the alphabet of the transition system is A;

– or the action redexes, if this alphabet is R; in this latter case the transition system is deterministic, *cf.* proposition 2.1.

3.3. Muller transition automata

We modelled the behavior of an SCCS process via a transition system; before introducing fair computations for SCCS processes, we first need to

describe the successful computations of these transition systems. To this end, we will transform our transition systems into automata by imposing some recognition criteria. We will mainly consider Muller automata.

For an alphabet A, let A^* (resp. A^ω) denote the set of finite (resp. infinite) sequences over A, and $A^\infty = A^* \cup A^\omega$.

DEFINITION 3.3: Let $S=(Q, q_0, D)$ be a transition system over A; for each $w \in A^\infty$ and $q \in Q$ define the set of paths visited by w starting in state q by:

$$path(q, w) = \{ t_{i_1} t_{i_2} \ldots t_{i_n} \ldots \in D^\infty / \forall i_j,\ t_{i_j} = (q_{i_j}, a_j, q_{i_{j+1}}) \in D, \quad w = a_1 a_2 \ldots a_n \ldots \text{ and } q_{i_1} = q \}.$$

For all c in $path(q, w)$, let $st(c) = q_{i_1} q_{i_2} \ldots q_{i_n} \ldots \in Q^\infty$ be the sequence of states visited in the course of computation c, and define:

$$\mathrm{Inf}_s(c) = \{ q \in Q / |st(c)|_q = \infty \} \qquad \text{and} \qquad \mathrm{Inf}_t(c) = \{ t \in D / |c|_t = \infty \},$$

where $|w|_a$ denotes the number of occurrences of a in w. For each c in $path(q, w)$, w is called the *trace* of c, and this is denoted by: $w = trace(c)$.

A *Muller automaton* over A is a triple $\mathscr{A} = (S, Q_T, Q_{\inf})$, where S is a transition system over A, $Q_T \subseteq Q$ is the set of terminal states, and $Q_{\inf} \subseteq 2^Q$ is the set of infinitary states. A *Muller T-automaton* over A is a triple $\mathscr{TA} = (S, Q_T, T_{\inf})$, where S and Q_T are as above, and $T_{\inf} \subseteq 2^D$ is the set of infinitary derivations or transitions.

The language accepted by a Muller automaton $\mathscr{A}$ is $L(\mathscr{A}) = |\mathscr{A}| \cup \|\mathscr{A}\|$, where:

$$|\mathscr{A}| = \{ w \in \mathscr{A}^* / \exists c \in path(q_0, w),\ st(c) = q_{i_1} q_{i_2} \ldots q_{i_n} \text{ and } q_{i_n} \in Q_T \},$$

and

$$\|\mathscr{A}\| = \{ w \in \mathscr{A}^\omega / \exists c \in path(q_0, w),\ \mathrm{Inf}_s(c) \in Q_{\inf} \}.$$

The language accepted by a Muller T-automaton $\mathscr{TA}$ is $L(\mathscr{TA}) = |\mathscr{TA}| \cup \|\mathscr{TA}\|_t$, where:

$|\mathscr{TA}|$ is as above, and

$\|\mathscr{TA}\|_t = \{ w \in \mathscr{A}^\omega / \exists c \in path(q_0, w),\ c = t_{i_1} t_{i_2} \ldots t_{i_n} \ldots \text{ and } \mathrm{Inf}_t(c) \in T_{\inf} \}$.

Muller T-automata are somehow similar to the automata considered in [17]. The idea leading to the notion of Muller T-automaton is the following: in a Muller automaton, an infinite computation is successful iff it eventually cycles through a set of infinitary states, whereas in a Muller T-automaton, an infinite computation is successful iff it eventually cycles through a set of infinitary transitions, which is a somehow more precise information. However,

the two notions are equivalent as shown by the:

PROPOSITION 3.2: *The class of languages accepted by Muller automata and by Muller T-automata coincide, and are the* ω*-regular languages.*

Sketch of proof: It is clear that any language accepted by a Muller T-automaton (S, Q_T, T_{inf}) can be accepted by a Muller automaton $(S', Q'_T, Q_{\inf})$, where the states of S' are the pairs (t, q), such that $t=(q', a, q)$ is a transition of S, $Q_{\inf}$ is defined by:

$$Q' \in Q_{\inf} \Leftrightarrow \exists\, T' \in T_{\inf} \text{ such that } Q' = \{(t, q)/t=(q', a, q) \in T'\},$$

and $(t, q) \xrightarrow{a} (t', q') \Leftrightarrow q \xrightarrow{a} q'$. Conversely, if $\mathscr{A}$ is a Muller automaton, and c a successful computation thereof, such that $\mathrm{Inf}_s(c)=Q' \in Q_{\inf}$, then Q' defines cycles of transitions $T_1, \ldots, T_n$, one of which will be taken infinitely often in the course of computation c.

Muller automata were first introduced in [23], where it is shown that they recognize the class of ω-regular languages, *see* also [10, 20], and [33] for a nice survey on how to recognize infinitary languages. The following then can be proved easily:

PROPOSITION 3.3: *The classes of languages (over A or R) corresponding to the derivations (resp. computations) of regular* SCCS *processes are closed and prefix-closed* ω*-regular languages [resp. closed* ε*-free* ω*-regular languages, i.e.* ω*-regular languages L such that the empty word* $\varepsilon \notin L$*, and* $adh(L) \subseteq L$*].*

It is not true that an arbitrary (ε-free) ω-regular language is the set of derivations or computations of some process: for instance, a^* (or a^+) cannot be a set of derivations or computations, because any set of derivations or computations containing a^* (or a^+) should also contain a^ω, since such a set should be closed. We will see later on, in theorem 4.1, that the power of fairness is so great as to enable us to generate all ε-free ω-regular languages, instead of just the closed ones. Nonetheless, it is shown in [7] that strongly fair computations can be described as limits of Cauchy sequences with respect to some very special metric distance defined on finite derivations; hence sets of fair computations are closed with respect to that metric.

A variant of the notion of Muller T-automaton, useful in the study of fairness (*see* the proof of proposition 4.5), can be obtained by modifying the acceptance condition as follows: $L(\mathscr{T}\mathscr{A}) = |\mathscr{T}\mathscr{A}| \cup ||\mathscr{T}\mathscr{A}||'$, where $||\mathscr{T}\mathscr{A}||' = \{w \in A^\omega / \exists\, c \in path(q, w), c = t_{i_1} t_{i_2} \ldots t_{i_n} \ldots \text{ and } \mathrm{Inf}_t(c) \supseteq T' \in T_{\inf}\}$.

We can show by a subset construction (*cf.* [25]) on the set of transitions that the class of languages defined by this last acceptance condition is again the class of ω-regular languages. See [26] for the proof.

4. FAIRNESS

4.1. Strong fairness

Intuitively, a computation of a process p is *strongly fair* iff every subprocess which is enabled infinitely often is active infinitely often. A subprocess is any concurrent component in a synchronous product; a process is enabled if it has the possibility of performing an effective action, and it is active if it performs an effective action, different from a delay. In other words, in a strongly fair computation, a process which can perform an effective action shall not delay forever.

Example 4.1: Let p be:

$$p = [\delta d\,\mathrm{NIL} \times \mathrm{rec}\,x.\,\delta ax \times \mathrm{rec}\,x.\,\delta bx \times \mathrm{rec}\,x.\,(\bar{a}:\bar{b}x) \times \mathrm{rec}\,x.\,(ex+bx)]\uparrow\{e\}$$

where we used the associativity of $\times$ to delete useless parentheses, and where the : sign indicating sequential composition has been omitted whenever this creates no ambiguity; and let c (resp. c') be the derivations:

$$c:\quad p \xrightarrow{\langle \delta, a, \delta, \bar{a}, e\rangle} [\delta d\mathrm{NIL} \times \mathrm{rec}\,x.\,\delta ax \times \mathrm{rec}\,x.\,\delta bx \times \bar{b}:(\mathrm{rec}\,x.\,\bar{a}:\bar{b}x) \times \mathrm{rec}\,x.\,(ex+bx)]\uparrow\{e\} \xrightarrow{\langle \delta, \delta, \delta, \bar{b}, b\rangle} p$$

$$c':\quad p \xrightarrow{\langle \delta, a, \delta, \bar{a}, e\rangle} [\delta d\mathrm{NIL} \times \mathrm{rec}\,x.\,\delta ax \times \mathrm{rec}\,x.\,\delta bx \times \bar{b}:(\mathrm{rec}\,x.\,\bar{a}:\bar{b}x) \times \mathrm{rec}\,x.\,(ex+bx)]\uparrow\{e\} \xrightarrow{\langle \delta, \delta, b\,\bar{b}, e\rangle} p.$$

Then the computation c^{ω} is not strongly fair, because the third process is infinitely often enabled but never activated; we will see in the next section that this computation is nevertheless weakly fair. The computation c'^{ω}, on the other hand, is strongly fair.

Clearly, any finite computation is strongly fair. For an infinite computation, we will show that the study of fairness boils down to the study of the cycles in the computation. Note that, as in the rest of this section, our attention will be restricted to regular processes, unless explicitly stated otherwise.

We will define a variant of action redexes, the derivation redexes, tailored for fairness. Since fairness is concerned exclusively with the behavior with respect to the synchronous product $\times$, derivation redexes will model exclusively the behavior of the processes with respect to the factors of unguarded synchronous products. The set L of *derivation redexes* will consist of the finite lists of action names, to which we will add a new symbol σ, intended to keep track of internal synchronizations.

We will also define a relation $\xrightarrow{l}$ on $\mathbf{E}$, for $l \in L$. To this end we need first some notations.

NOTATION: Define the mapping π^* from action-redexes, to L, the set of derivation redexes, by:

$$\pi^*(r) = \begin{cases} \pi^*(r'), & \text{if } r = \uparrow r', \\ \pi^*(r_1) * \pi^*(r_2), & \text{if } r = \langle r_1, r_2 \rangle, \\ name_s(r), & \text{otherwise,} \end{cases}$$

where * is again the concatenation of lists, and $name_s$: $R \to A \cup \{\sigma\}$ is defined as follows:

1. If $name(r) \neq 1$, then $name_s(r) = name(r)$, *cf.* section 2.2 for the definition of $name(r)$.

2. If $name(r) = 1$, then $name_s(r)$ is defined inductively as name(r), the first 3 cases being identical to the definition of $name(r)$ given in section 2.2, the last case becoming:

$$name_s(\langle r, r' \rangle) = \begin{cases} \sigma, & \text{if } \langle r, r' \rangle = \langle a, \bar{a} \rangle \text{ for } a \in A, \text{ or } name_s(r) = \sigma, \text{ or } name_s(r') = \sigma, \\ 1, & \text{otherwise.} \end{cases}$$

So, $name_s(r)$ is identical to $name(r)$, except that it keeps track of synchronizations. We identify $name_s(r)$ with the list having the single element $\langle name_s(r) \rangle$.

Let $\pi_i^*(r)$ be the i-th component of $\pi^*(r)$, if it exists, namely if $\pi^*(r)$ has $n \geqq i$ elements.

Finally, extend $\pi^*(r)$ and $\pi_i^*(r)$ to sets C of action redexes, in the obvious way, e. g.:

$$\bar{\pi}_i^*(C) = \{\pi_i^*(r) / r \in C\}.$$

The relation $\xrightarrow{l}$ is then defined on E as follows: for each p, $p' \in \mathbf{E}$ such that there exists a derivation $d: p \xrightarrow{r} p'$, the relation $p \xrightarrow{l} p'$ holds, with $l = \pi^*(r) \in L$.

We might define directly the relation $\xrightarrow{l}$ on $\mathbf{E}$ as the least relation containing, for all $a \in A$ and $l \in L$:

$a: E \xrightarrow{a} E$;

if $E \xrightarrow{a} E'$ and $n(l) \in B \cup \{\sigma\}$ [*see* below the definition of $n(l)$] then:

$E \uparrow B \xrightarrow{l} E' \uparrow B$;

if $E \xrightarrow{l} E'$ then $E + F \xrightarrow{n(l)} E'$ and $F + E \xrightarrow{n(l)} E'$;

if $E \xrightarrow{l} E'$ and $F \xrightarrow{l'} F'$ then $E \times F \xrightarrow{l * l'} E' \times F'$;

if $E[\mathrm{rec}\, x.\, E/x] \xrightarrow{l} E'$ then $\mathrm{rec}\, x.\, E \xrightarrow{n(l)} E'$,

finally, $\delta E \xrightarrow{1} \delta E$, and moreover, if $E \xrightarrow{l} E'$ then $\delta E \xrightarrow{n(l)} E'$,

where $n(l)$ is defined by, φ being the alphabetic morphism $\varphi: A \cup \{\sigma\} \to A$ which erases σ:

$$n(\langle b_1, \ldots, b_n \rangle) = \begin{cases} \varphi(b_1 \ldots b_n), & \text{if } \varphi(b_1 \ldots b_n) \neq 1, \\ \sigma, & \text{if } \varphi(b_1 \ldots b_n) = 1 \text{ and } \exists i\, b_i = \sigma \neq 1, \\ 1, & \text{if } b_1 = \ldots = b_n = 1. \end{cases}$$

The π^* notation is tailored for expressing fairness, where we are concerned only with unguarded components of synchronous products. See example 4.2 (i) below. If $d: p \xrightarrow{r} p'$, then $\pi^*(r)$ will also be denoted by $\pi^*(d)$.

Example 4.2: (i) Considering $d: b\,\mathrm{NIL} \times c\,\mathrm{NIL} \xrightarrow{\langle b, c \rangle} NIL$, then $\pi^*(d) = \langle b, c \rangle$, and the relation $b\,NIL \times c\,NIL \xrightarrow{\langle b, c \rangle} NIL$ also holds when $\langle b, c \rangle$ is considered as a derivation redex label in L. However, letting $d': a + b\,NIL \times c\,NIL \xrightarrow{+2\langle b, c \rangle} NIL$, *we have* $\pi^*(d') = bc$, and the derivation relation becomes with labels in $L: a + b\,NIL \times c\,NIL \xrightarrow{bc} NIL$.

Replacing now c by $\bar{b}$ in this example, we would obtain: $\pi^*(\langle b, \bar{b} \rangle) = \langle b, \bar{b} \rangle$, but $\pi^*(+2\langle b, \bar{b} \rangle) = \sigma$.

(ii) Let

$$d: \quad a:p_1 \times \delta((b:p_2+p_3) \times \delta(p_4 \times p_5)) \xrightarrow{r=\langle a, \delta \langle +1\, b, \delta \rangle \rangle} p_1 \times (p_2 \times \delta(p_4 \times p_5)),$$

then: $\pi^*(r)=\langle a, b \rangle$, $\pi_2^*(r)=b$.

(iii) Let $p=[\mathrm{rec}\, x.(ax+\mathrm{rec}\, z.bz \times \mathrm{rec}\, z.cz) \times \mathrm{rec}\, y.\delta fy] \uparrow \{A-\{f\}\}$;

then, letting $d:p \xrightarrow{\langle \mathrm{rec}+1\, a, \mathrm{rec}\, \delta \rangle} p$, $\pi^*(d)=\langle a, 1 \rangle$. Letting now

$$d':p \xrightarrow{\langle \mathrm{rec}+2\langle b, c \rangle, \mathrm{rec}\, \delta \rangle} p'=[(\mathrm{rec}\, z.bz \times \mathrm{rec}\, z.cz) \times \mathrm{rec}\, y.\delta fy] \uparrow \{A-\{f\}\},$$

we have $\pi^*(d')=\langle bc, 1 \rangle$. And finally, letting $d'':p' \xrightarrow{\langle \langle \mathrm{rec}\, b, \mathrm{rec}\, c \rangle, \mathrm{rec}\, \delta \rangle} p'$, we obtain $\pi^*(d'')=\langle b, c, 1 \rangle$.

DEFINITION 4.1: (i) Let $S=(Q, q_0, D)$ be a transition system over the alphabet R which models the behavior of a regular process p, i.e. $S=\mathscr{B}(p)$. A *cycle* c in S is a path

$$c=t_{i_1} t_{i_2} \ldots t_{i_n} \in D^*$$

such that

$$\forall i_j, \quad t_{i_j}=(q_{i_j}, r_j, q_{i_{j+1}}) \in D, \quad \text{and} \quad q_{i_1}=q_{i_{n+1}}.$$

Let $\mathscr{T}(S)$ be the set of $T \subseteq D$ such that for some cycle c in S, $T=c$ (this last notation being a shorthand for $T=\{t / |c|_t \neq 0\}$).

(ii) To each $S=\mathscr{B}(p)$, we will associate a transition system S' over the alphabet L of derivation redexes by relabelling the transitions of S by their π^* image, i.e. $S'=(Q, q_0, D')$, where if $t:p \xrightarrow{r} p' \in D$, then

$$\pi^*(t):p \xrightarrow{\pi^*(r)} p' \in D'$$

is the corresponding relabeled transition of S'. We will denote this new transition system by $S'=\mathscr{D}(p)$. Cycles in S' are defined as in (i).

(iii) A *cycle* of a process p is a derivation c of the form:

$$c: \quad p \xrightarrow{r_1} p_1 \xrightarrow{r_2} p_2 \ldots \xrightarrow{r_n} p_n=p.$$

(iv) A process p is said to be *cyclic* iff it has a cycle.

In the rest of the section we will identify a process p with the transition system S such that $S=\mathscr{B}(p)$ [or S' such that $S'=\mathscr{D}(p)$], whence the part (iii) of the above definition.

PROPOSITION 4.1: *Let S' be a transition system such that $S'=\mathscr{D}(p)$ as in definition 4.1, and u a path of S', then u is infinite iff there exists a cycle c in S' such that:* $\mathrm{Inf}_t(u)=c$.

Sketch of proof: The only thing to prove is the "only if" direction. If u is an infinite path in a finite transition system, then the following condition is satisfied by $\mathrm{Inf}_t(u)$:

$$\forall\, t_{i_j}=(q_{i_j},\, a_j,\, q_{i_{j+1}})\in \mathrm{Inf}_t(u),$$

$$\exists\, t_{i_k}=(q_{i_k},\, a_k,\, q_{i_{k+1}})\in \mathrm{Inf}_t(u) \quad \text{such that } q_{i_{j+1}}=q_{i_k},$$

and

$$\exists\, t_{i_l}=(q_{i_l},\, a_l,\, q_{i_{l+1}})\in \mathrm{Inf}_t(u) \quad \text{such that } q_{i_{l+1}}=q_{i_j}.$$

If $\mathrm{Inf}_t(u)$ satisfies the above condition, then $\mathrm{Inf}_t(u)$ defines a set of (possibly unconnected) cycles in S'. u being a path though, implies that the set defined by $\mathrm{Inf}_t(u)$ must be connected. Hence $\mathrm{Inf}_t(u)$ defines a cycle of S'.

This proposition says that a path u is infinite iff the transitions in $\mathrm{Inf}_t(u)$ form a cycle, or equivalently, that the transitions of u are eventually all in a cycle c.

We can conclude that the cycles in S' characterize the infinite paths in S'.

Proposition 4.1 remains true if we replace S' by a transition system S such that $S=\mathscr{C}(p)$, but becomes false if it is assumed that $S=\mathscr{B}(p)$: take e.g. $p=\mathrm{rec}\,x.(ax+x)$. This is one of the reasons why we needed to introduce the derivation redexes, better suited to our notion of fairness than the actions redexes.

We now come to the study of fairness; we need to define the notions of enabled and active subprocesses. Intuitively, a subprocess of a process p is any factor, or component of an unguarded synchronous product, i.e. a product which is not within a subterm of the form $a:q$, or $p+q$, or $\mathrm{rec}\,x.E$. So for instance a process which has no unguarded occurrence of a product $\times$ will be its own sole subprocess; any subprocess will thus contribute a (possibly noneffective) action in the cycle c. The number of subprocesses may evolve in the course of a computation. For fairness though, we will be

interested only in subprocesses which occur in cycles. All the notions introduced in the sequel will be relative to a cycle c; this cycle though, being once and for all fixed, will not be explicited.

DEFINITION 4.2: Let p be a cyclic regular process in **R**; the list $SP(p)$ of subprocesses of p in a cycle c of p is defined by induction on the structure of p as follows:

$$SP(NIL)=\varnothing,$$

$$SP(P\uparrow B)=SP(P),$$

$$SP(P_1\times P_2)=SP(P_1)*SP(P_2),$$

$$SP(p)=\langle p\rangle \text{ otherwise,}$$

where $*$ denotes the concatenation of lists defined by:

$$\langle E_1,\ldots,E_n\rangle * \langle F_1,\ldots,F_m\rangle=\langle E_1,\ldots,E_n,F_1,\ldots,F_m\rangle.$$

Remark: $SP(p)$ is the set of subprocesses of p which may perform an action starting the cycle c. The facts that p is cyclic, and c is a cycle of p, are essential in the above definition of subprocesses, *cf.* lemmata 4.2 and 4.3 below. The cycle c is always implicitly understood in the definition of $SP(p)$, even though it is omitted for simplicity's sake.

Example 4.3: Let $p=\delta(\operatorname{rec}x.ax\times\operatorname{rec}y.by)\times\operatorname{rec}z.dz$. Let $c:p\xrightarrow{\langle\delta,d\rangle}p$ be a cycle of p. Then: $SP(p)=\langle\delta(\operatorname{rec}x.ax\times\operatorname{rec}y.by),\ \operatorname{rec}z.dz\rangle$. Note that p is *not* strictly regular.

Let $S=(Q,q_0,D)$ be a transition system over R; recall that a computation is a maximal path, having one of the forms:

– a finite path $c=t_{i_1}t_{i_2}\ldots t_{i_n}$, with $t_{i_j}=(q_{i_j},a_j,q_{i_{j+1}})$, for all $j=1,\ldots,n$, $q_{i_1}=q_0$ and Out $(q_{i_{n+1}})=\varnothing$;

– an infinite path $c=t_{i_1}t_{i_2}\ldots t_{i_n}\ldots$, with t_{i_j} as above and $q_{i_1}=q_0$.

LEMMA 4.1: *Let* $p\xrightarrow{r}p'\xrightarrow{r'}p''$ *be a derivation, then* $|\pi^*(r)|\leqq|\pi^*(r')|$.

Proof: Let $n=|\pi^*(r)|$; this implies that, up to an associative parenthesizing, $p=p_1\times\ldots\times p_n\xrightarrow{r}p'=p'_1\times\ldots\times p'_n$, and $r=\langle r_1,\ldots,r_n\rangle$. Hence, since p' can be derived, all of the p'_i's can also be derived, and $r'=\langle r'_1,\ldots,r'_n\rangle$, with $p'=p'_1\times\ldots\times p'_n\xrightarrow{r'}p''$. Whence $\pi^*(r')=\pi^*(r'_1)*\ldots*\pi^*(r'_n)$; since each $\pi^*(r'_i)$ is of length at least 1, $|\pi^*(r')|\geqq n=|\pi^*(r)|$.

LEMMA 4.2: *Let c be a cycle of a process p of the form:* $c: p=p_1 \xrightarrow{r_1} p_2 \xrightarrow{r_2} p_3 \ldots \xrightarrow{r_n} p_n=p$; *then* $|\pi^*(r_1)|=|\pi^*(r_2)|=\ldots=|\pi^*(r_n)|=k$, *and* $SP(p)=\{q_1, \ldots, q_k\}$.

Proof: If $p_i \xrightarrow{r_i} p_{i+1} \xrightarrow{r_{i+1}} p_{i+2}$, then, by Lemma 4.1, $|\pi^*(r_i)| \leqq |\pi^*(r_{i+1})|$. Whence: $|\pi^*(r_1)| \leqq |\pi^*(r_2)| \leqq \ldots \leqq |\pi^*(r_n)| \leqq |\pi^*(r_1)|$. Thus in the cycle c, the process p has exactly k synchronous components which are unguarded and perform an action. So, according to definition 4.2, p has k subprocesses in c.

LEMMA 4.3: *Let c, p, k be as in Lemma* 4.2, *and let* $t=p \xrightarrow{r} p'$, $t \in c$; *then* $\pi^*(t)=\langle e_1, \ldots, e_k \rangle$ *with* $q_i \xrightarrow{r_i} q_i'$ *and* $name(r_i)=e_i$, *for* $i=1, \ldots, k$, *i.e.* $r_1, \ldots, r_k$ *are the unguarded synchronous components of r.*

The intuitive meaning of lemmata 4.2 and 4.3 is the following: with respect to the cycle c, the process p has exactly k factors, namely unguarded components of a synchronous product. Moreover, all the processes p_i obtained in the cycle c have the same number of factors; and, finally, the factors of any given index j also form a cycle, which might be called the "projection" of cycle c on its j-th factor. Note that these lemmata are valid for any cycle of an arbitrary SCCS process.

DEFINITION 4.3: Let p, c, k and $r_1, \ldots, r_k$ be as in lemma 4.3, and define the list $ASP(p)$ [resp. $ESP(p)$] of subprocesses of p which are *active* (resp. *enabled*) in c by:

$$ASP(p)=\langle q_{i_1}, \ldots, q_{i_m} \rangle$$

where for

$$j=1, \ldots, m, \qquad \exists t \in c, \quad t: p' \xrightarrow{r} s, \quad \exists i \in \{1, \ldots, k\},$$

$$q_{i_j}=q_i \in SP(p) \qquad \text{with} \quad q_i \xrightarrow{r_i} q_i' \quad \text{and} \quad name(r_i)=\pi_i^*(r) \neq 1,$$

$$ESP(p)=\langle q_{i_1}, \ldots, q_{i_{m'}} \rangle$$

where for

$$j=1, \ldots, m', \qquad \exists t \in c, \quad \exists t', \quad t: p' \xrightarrow{r} s, \quad t': p' \xrightarrow{r'} s', \quad \exists r_i \text{ subterm of } r',$$

with $q_{i_j}=q_i\in SP(p)$, $q_i \xrightarrow{r_i} q_i'$ and for some j, l, $\pi_j^*(r_i)=\pi_l^*(r')\neq 1$, where t, t' are transitions of the set D of derivations of the automaton $\mathscr{B}(p)$ representing the behavior of p, and $c\subseteq D$.

Remark: SP, ASP and ESP must be lists and not sets in order to keep track of possible multiple occurrences of identical subprocesses. Note moreover that $ASP(p)$ is a sublist of $SP(p)$, so in the case of multiple occurrences of identical subprocesses we can remember the number of the subprocess which was active; this is still true for $ESP(p)$, though less important because identical subprocesses will always be enabled together.

Example 4.4: (i) Let $p=\delta\, a\, NIL\times \mathrm{rec}\, x.\, ax\times \delta\, a\, NIL$ and let c be the cycle: $c: p \xrightarrow{\langle \delta,\, a,\, \delta\rangle} p$. Then $SP(p)=\langle \delta\, a\, NIL,\ \mathrm{rec}\, x.\, ax,\ \delta\, a\, NIL\rangle = ESP(p)$, and $ASP(p)=\langle \mathrm{rec}\, x.\, ax\rangle$.

(ii) Let $p=\delta(\mathrm{rec}\, x.\, ax\times \mathrm{rec}\, y.\, \bar{a}y)\times \mathrm{rec}\, z.\, dz$, and let $c: p \xrightarrow{\langle \delta,\, d\rangle} p$. Then: $SP(p)=\langle \delta(\mathrm{rec}\, x.\, ax\times \mathrm{rec}\, y.\, \bar{a}y),\ \mathrm{rec}\, z.\, dz\rangle = ESP(p)$, and $ASP(p)=\langle \mathrm{rec}\, z.\, dz\rangle$. The subprocess $\delta(\mathrm{rec}\, x.\, ax\times \mathrm{rec}\, y.\, \bar{a}y)$ is enabled because of the possible derivation:

$$p \xrightarrow{\langle \delta\langle a,\, \bar{a}\rangle,\, d\rangle} \mathrm{rec}\, x.\, ax\times \mathrm{rec}\, y.\, \bar{a}y\times \mathrm{rec}\, z.\, dz.$$

(iii) Let $p=[\delta\, a\, \mathrm{NIL}\times \mathrm{rec}\, x.\, \bar{a}x\times \mathrm{rec}\, y.\, ay]\uparrow\{1\}$, and let $c: p \xrightarrow{\langle \delta,\, \bar{a},\, a\rangle} p$. Then:

$$SP(p)=\langle \delta\, a\, NIL,\ \mathrm{rec}\, x.\, \bar{a}x,\ \mathrm{rec}\, y.\, ay\rangle,$$

and

$$ASP(p)=\langle \mathrm{rec}\, x.\, \bar{a}x,\ \mathrm{rec}\, y.\, ay\rangle = ESP(p).$$

(iv) Let $p=[\mathrm{rec}\, x.\, (ax+\mathrm{rec}\, z.\, bz\times \mathrm{rec}\, z.\, cz)\times \mathrm{rec}\, y.\, \delta fy]\uparrow\{A-\{f\}\}$, and $p'=[\mathrm{rec}\, z.\, bz\times \mathrm{rec}\, z.\, cz\times \mathrm{rec}\, y.\, \delta fy]\uparrow\{A-\{f\}\}$, be as in example 4.2 (iii). Let $c: p \xrightarrow{\langle a,\, \delta\rangle} p$; then $SP(p)=\langle \mathrm{rec}\, x.\, (ax+\mathrm{rec}\, z.\, bz\times \mathrm{rec}\, z.\, cz),\ \mathrm{rec}\, y.\, \delta fy\rangle$, and $ASP(p)=\langle \mathrm{rec}\, x.\, (ax+\mathrm{rec}\, z.\, bz\times \mathrm{rec}\, z.\, cz)\rangle = ESP(p)$. Let now $c': p' \xrightarrow{\langle\langle b,\, c\rangle,\, \delta\rangle} p'$, then $SP(p')=\langle \mathrm{rec}\, z.\, bz,\ \mathrm{rec}\, z.\, cz,\ \mathrm{rec}\, y.\, \delta fy\rangle$, and $ASP(p')=\langle \mathrm{rec}\, z.\, bz,\ \mathrm{rec}\, z.\, cz\rangle = ESP(p')$.

DEFINITION 4.4: Let $c: p=p_1 \xrightarrow{r_1} p_2 \xrightarrow{r_2} p_3 \ldots \xrightarrow{r_n} p_n=p$ be a cycle of a process p; c is said to be *strongly fair* iff every subprocess of every process p_i

in c which is enabled is active, i.e. formally, iff $ESP(p_i)=ASP(p_i)$, for $i=1, \ldots, n$.

Example 4.1 (continued): The cycle c is not strongly fair because the subprocess $\mathrm{rec}\, x.\, \delta\, bx$ is enabled but not active in c.

Example 4.3 (continued): The cycle c is not strongly fair because the subprocess $\delta(\mathrm{rec}\, x.\, ax \times \mathrm{rec}\, y.\, by)$ is enabled but not active in c.

Example 4.4 (continued): Only the cycles c and c' given in parts (iii) and (iv) are strongly fair.

The following proposition now becomes immediate:

PROPOSITION 4.2: *Let $S=(Q, q_0, D)$ be a transition system over R modelling the behavior of a regular process p, i.e. $S=\mathscr{B}(p)$, then a computation c in S is strongly fair iff:*

- *either it is finite (and maximal);*
- *or it is infinite and satisfies:*

$$\forall i, \quad \begin{array}{l} \text{if } \exists p' \in \mathrm{Inf}_s(c) \quad \text{such that } \varnothing \neq \pi_i^*(\mathrm{Out}\,(p')) \neq \{1\}, \\ \text{then } \exists t \in \mathrm{Inf}_t(c),\ t: p'' \xrightarrow{r} s \quad \text{such that } \pi_i^*(r) \neq 1. \end{array} \tag{1}$$

Sketch of proof: Clearly, the cycle $\mathrm{Inf}_t(c)$ satisfies the strong fairness condition: whenever the i-th component of an action redex is enabled in $\mathrm{Inf}_s(c)$, then it eventually performs an effective action in $\mathrm{Inf}_t(c)$.

This condition models quite adequately, and in an intuitively operational way, the requirements of strong fairness.

We now will construct a Muller T-automaton which recognizes the set of strongly fair computations of an arbitrary regular SCCS process.

PROPOSITION 4.3: *Let $S=(Q, q_0, D)$ be a transition system over R modelling the behavior of a regular process p, i.e. $S=\mathscr{B}(p)$; define a Muller T-automaton as follows: $\mathscr{TA}=(S', Q_T, T_{\mathrm{inf}})$, where $S'=\mathscr{D}(p)$ is deduced from the above given transition system S over R, by relabeling the transitions of S by their π^* image, as in definition* 4.1 (ii), *$Q_T=\{q \in Q / \mathrm{Out}\,(q)=\varnothing$ in $S\}$, and*

$$T_{\mathrm{inf}}=\{\pi^*(T) / T=\{t_i / i \in I\} \in \mathscr{T}(S), \quad \forall i,\ t_i: q_i \xrightarrow{r_i} q_i', \text{ and}$$
$$\forall j=1, \ldots, k, \quad (\bigcup_{i \in I} \bar{\pi}_j^*(\mathrm{Out}\,(q_i)) \neq \{1\}) \Rightarrow (\bar{\pi}_j^*(\bigcup_{i \in I} r_i) \neq \{1\})\},$$

where T is the set of transitions of a cycle c in S, and k is the number of synchronous components (cf. *lemma* 4.2) *in c.*

Finally, define $\mathcal{TA}' = (Q, q_0, D'', Q_T, T'_{\text{inf}})$ *by relabeling the transitions of* $\mathcal{TA}$ *by their name in A, i.e. for each* $t' : p \xrightarrow{r} p' \in D'$, $r = \langle b_1, \ldots, b_n \rangle$, *let* $t'' : p \xrightarrow{\varphi(b_1 \ldots b_n)} p'$ *be in D'', where* φ *is the alphabetic morphism* $\varphi : A \cup \{\sigma\} \to A$ *which erases* σ. *Let* T''_{inf} *be the infinitary cycles deduced from* T_{inf} *by the same relabeling. Then:*

$L(\mathcal{TA}') = \{ w = trace(c)/c$ *is a strongly fair computation of S starting in state* $q_0 \}$.

Remark: We would obtain a slightly simpler formulation of proposition 4.3, and the similar ones propositions 4.8 and 4.10, by considering directly derivation redexes instead of action redexes to start with. This leads to a shortcut saving the use of the transition system S and of its π^* image. We chose the present approach to make more explicit the relationship with the usual action redex formalism.

Proof of Proposition 4.3: The idea is the following: the Muller T-automaton $\mathcal{TA}$ recognizes the strongly fair computations of p; then, the Muller T-automaton $\mathcal{TA}'$, deduced from $\mathcal{TA}$ by replacing action-redex labels by the corresponding action names, recognizes the traces of strongly fair computations of p. $\mathcal{TA}'$ accepts w if there exists some fair computation c of p with trace w.

Note first that the transition system S' is finite, even if S was infinite: this stems from the fact that p is regular, hence has a finite number of derivatives, which can be derived into one another via a finite number of actions; and the states and transitions of S' consist of respectively those derivatives and actions (up to decomposition of the actions into finitely many synchronous components). Hence $\mathcal{TA}$ is indeed a Muller T-automaton. Now, clearly, a finite successful computation of $\mathcal{TA}$ which is in $|\mathcal{TA}|$ is a maximal finite computation of S, and the set T_{inf} of infinitary transitions is defined so that the set $\|\mathcal{TA}\|_t$ of infinite successful computations of $\mathcal{TA}$ will fulfill condition (1) of proposition 4.2.

COROLLARY 4.4: *The set of strongly fair computations of a regular* SCCS *process either is an* ε*-free* ω*-regular language, i.e. an* ω*-regular language L such that* $\varepsilon \notin L$, *or is reduced to* ε, *i.e.* $L = \{\varepsilon\}$.

Proof: The fact that the set of strongly fair computations of a regular SCCS process is an ω-regular language follows from the previous proposition. The fact that this language is ε-free proceeds from the previous construction: the only case when ε might belong to the language is the case when $q_0 \in Q_T$, hence Out $(q_0) = \varnothing$; this means the SCCS process we started with is bisimilar

to *NIL* and has no derivation at all; hence its set of strongly fair computations is ε.

We now state a converse to corollary 4.4. Recall that A is the alphabet of actions.

LEMMA 4.4: (*i*) *Let* $K \subset A^+$ *be an* ε*-free regular language, i.e. such that* $\varepsilon \notin K$. *Then there exists an* SCCS *process p, which can be built using only the actions in Act,* + *and* rec, *and such that K is the set of finite computations of p.*

(ii) *Let* $K \subset A^+$ *be an* ε*-free regular language, then there exists a strictly regular* SCCS *process p′ such that K is the set of strongly fair computations of p′.*

Sketch of proof: (i) If K is ε-free, then it can be accepted by an automaton $\mathscr{A}$ in which all terminal states q are such that Out $(q) = \varnothing$, i.e. are final in the terminology of [24]; hence, if p is a process such that $\mathscr{A} = \mathscr{C}(p)$, $\mathscr{A}$ recognizes exactly the maximal derivations, i.e. the computations, of p.

The idea of accepting ε-free regular languages by automata where all terminal states are final first appears in [24].

(ii) The idea is to take a process a whose set of finite computations is K by (i), and, via a suitable product, to force all fair computations to be finite, hence obtaining p' whose fair computations are the finite computations of p. Let $c_1, \ldots, c_n$ be all the cycles of the automaton C modelling the behavior of p, and let $\overline{c_1}, \ldots, \overline{c_n}$ be new letters, one for each cycle; relabel each transition (q, a, q') of C exiting from cycles $c_{i_1}, \ldots, c_{i_k}$ by $(q, a.\overline{c_{i_1}} \ldots \overline{c_{i_k}}, q')$ and let p'' be the resulting process; then, define p' by:

$$p' = [p'' \times (\mathrm{rec}\, x.\, \delta\, c_1\, x) \times \ldots \times (\mathrm{rec}\, x.\, \delta\, c_n\, x)] \upharpoonright \mathrm{sort}(p)$$

where $\mathrm{sort}(p) \subset \{A - \{c_1, \ldots, c_n\}\}$ is the set of actions which can be performed by p or its derivatives; the infinite computations of p' and p'' follow the same paths; however, strongly fair computations of p' cannot loop forever in any cycle, because of the fairness constraint on the components of the synchronous product in p'. Moreover, once a fair computation has gone out of any cycle which is maximal for the underlying set inclusion, there is no way it can reenter that cycle, hence all fair computations are finite; so, the set of strongly fair computations of p' is K.

Note first that this proof can be improved by finding a process p' with exactly one synchronous × operator which satisfies the conditions of lemma 4.4 (ii) [13].

Note then that if K is an arbitrary prefix closed regular language, possibly containing ε, K always represents the set of finite derivations of an SCCS process, but may not represent a set of finite computations: for instance, a^* is not a set of finite computations because computations are maximal, and the words in a^* are not maximal in the present case. The first problem we encounter for regular languages containing ε is to make sure that the words are obtained as *maximal* derivation sequences. See the counter examples to proposition 3.3, and also the acceptance condition by final, i.e. maximal, path of [30], or the notion of final state of [24]. Similarly, not every ω-regular language will be obtainable as the language of strongly fair computations of some SCCS process: for instance, $a^\infty = a^* \cup a^\omega$ is not a set of fair computations.

Note finally that bisimilar processes need not have the same strongly fair computations, e.g. $p = \mathrm{rec}\, x.\, \delta ax \times \mathrm{rec}\, x.\, \delta bx$ and $q = \mathrm{rec}\, x.\, \delta(abx + ax + bx)$ are bisimilar, but e.g. a^ω is the trace of a strongly fair computation of q, whereas all the strongly fair computations of p have a trace containing infinitely many b's.

PROPOSITION 4.5: *Let* $\mathrm{L} = \sum_{i=1}^{n} L_i K_i^\omega$ [10] *be an ω-regular language contained in A^ω. Then there exists a strictly regular SCCS process p, such that L represents the set of traces of strongly fair computations of p.*

Sketch of proof: It suffices to prove that each $L_i K_i^\omega$ is the set of fair computations of some process p_i; then L will be the set of fair computations of $p = \sum_{i=1}^{n} p_i$. So, assume $L' = LK^\omega$; note that K is ε-free, and that, by writing $LK^\omega = (L - \varepsilon) K^\omega \cup KK^\omega$, we also may assume that L is ε-free. So let p (resp. q) be an SCCS process whose set of (traces of) strongly fair computations is L (resp. K), as given in lemma 4.4 (ii) . Assume the behavior of p (resp. q) is represented by the automaton B (resp. C). Let C' be deduced from C by replacing each transition $q \xrightarrow{b} q' \in Q_T$ of C by $q \xrightarrow{b} q_0$, where q_0 is the initial state of C and q' a terminal state of C. The infinitary transitions of C' correspond to the cycles of C' containing one such transition $q \xrightarrow{b} q_0$. The strongly fair computations of C' form the language K^ω. Let B' be deduced from B by hooking to each terminal state of B a copy of C'; hence B' has no more terminal states, and its infinitary transitions are those coming from C'; B' behaves like B and then like C'. B' still represents the behavior of an

SCCS process p'. Hence, L' represents the strongly fair computations of the process p'.

This proposition states that purely infinitary ω-regular languages can be obtained as sets of strongly fair computations.

THEOREM 4.1: *An ε-free language $K \subseteq A^+ \cup A^\omega$ is ω-regular if and only if it is the set of traces of strongly fair computations of some (strictly) regular* SCCS *process.*

Proof: The "if" part follows from corollary 4.4. For the "only if" part, take an ε-free ω-regular language K and decompose it as $K = K_1 + K_2$, where $K_1 \subseteq A^+$ is regular, and $K_2 \subseteq A^\omega$ is ω-regular, apply then lemma 4.4 (ii) and proposition 4.5.

The main difference between our approach and those of [24, 30] is that we characterize sets of strongly fair computations as ε-free ω-regular languages, whereas they obtain *all* ω-regular languages. However, this difference can be remedied if we identify the empty word ε with 1, the silent move or delay of one time unit. Then, the counter examples to proposition 3.3 and lemma 4.4 are taken care of as follows: a^* (resp. a^∞) is the set of finite (resp. fair) computations of $\mathrm{rec}\, x.(ax + 1\ NIL)$. This identification could be understandable since 1 represents an invisible move, and actually, in the asynchronous case, [19] takes the option of such an identification. We can state:

PROPOSITION 4.6: *If we identify $\varepsilon \in A^*$ with $1 \in Act$, then:*

(i) *a language $K \subseteq A^*$ is regular iff it is the set of finite computations of some* SCCS *process;*

(ii) *a language $K \subseteq A^\infty$ is ω-regular iff it is the set of fair computations of some* SCCS *process.*

4.2. Weak fairness

We can establish similar results for weak fairness. Weak fairness has been much more widely studied in the literature [8, 14, 24]. For brevity's sake, we will only give the definition and propositions corresponding to propositions 4.2, 4.3, and theorem 4.1, without any more details, for the case of weak fairness.

DEFINITION 4.5: A computation of an SCCS process p is said to be *weakly fair* iff every subprocess which is permanently enabled from some point on is infinitely often active; a cycle c of p is said to be *weakly fair* iff every subprocess which is always enabled is at least once active.

Example 4.1 (continued): The computation c^{ω} is weakly fair, even though the first and third subprocesses are never active, because the first subprocess is never enabled, and the second subprocess is only enabled every second derivation.

PROPOSITION 4.7: *Let $S=(Q, q_0, D)$ be a transition system over R modelling the behavior of a regular process p, i.e. $S=\mathscr{B}(p)$, then a computation c in S is weakly fair iff:*

- *either it is finite (and maximal);*
- *or it is infinite and satisfies:*

$$\forall i, \quad \begin{array}{l} \text{if } \forall p' \in \mathrm{Inf}_s(c), \quad \varnothing \neq \bar{\pi}_i^*(\mathrm{Out}(p')) \neq \{1\}, \\ \text{then } \exists t \in \mathrm{Inf}_t(c),\ t: p'' \xrightarrow{r} s \quad \text{such that } \pi_i^*(r) \neq 1. \end{array} \tag{2}$$

PROPOSITION 4.8: *Let $S=(Q, q_0, D)$ be a transition system over R modelling the behavior of a regular process p, i.e. $S=\mathscr{B}(p)$; define a Muller T-automaton as follows: $\mathscr{T}\mathscr{A}_w=(S', Q_T, T_{\mathrm{inf}})$, where $S'=\mathscr{D}(p)$ is the transition system over L deduced from S as in proposition 4.3 and definition 4.1 (ii), $Q_T=\{q \in Q/\mathrm{Out}(q)=\varnothing$ in $S\}$, and*

$$T_{\mathrm{inf}}=\{\pi^*(T)/T=\{t_i/i \in I\} \in \mathscr{T}(S),\ \forall i\, t_i: q_i \xrightarrow{r_i} q_i',\ \text{and } \forall j=1, \ldots, k,$$
$$(\forall i \in I\ \varnothing \neq \bar{\pi}_j^*(\mathrm{Out}(q_i)) \neq \{1\}) \Rightarrow (\bar{\pi}_j^*(\bigcup_{i \in I} r_i) \neq \{1\})\}.$$

Let $\mathscr{T}\mathscr{A}'_w$ be deduced from $\mathscr{T}\mathscr{A}_w$ by relabeling transitions by the corresponding action names, as in proposition 4.3; then:

$L(\mathscr{T}\mathscr{A}'_w)=\{w=\mathrm{trace}(c)/c$ *is a weakly fair computation of S starting in state* $q_0\}$.

We can adapt the proof of lemma 4.4 (ii) to cover the case of weak fairness. The construction is a little more tricky though: at each state in a loop, we have to add new suitably labelled transitions which will allow for the possibility to simulate the rest of the loop up to and including an exit from that loop. So, the possibility of exiting the loop will be permanently allowed, hence the weak fairness constraint will force an eventual exit from the loop. The proofs of proposition 4.5 and theorem 4.1 then hold without

any change, whence the:

THEOREM 4.2: *An ε-free language $K \subseteq A^+ \cup A^\omega$ is ω-regular if and only if it is the set of traces of weakly fair computations of some (strictly) regular* SCCS *process.*

A similar theorem was first proved in [24] for characterizing weakly fair computations of asynchronous digital networks.

4.3. Strict fairness

DEFINITION 4.6 [14]: A computation of an SCCS process p is said to be *strictly fair* iff no subprocess can delay forever.

The following implications are obvious: u strictly fair $\Rightarrow u$ strongly fair $\Rightarrow u$ weakly fair.

Example 4.1 (continued): The condition of strict fairness is a bit exacting in requiring that *all* subprocesses perform effective actions: for instance, the process p of example 4.1 can have no strictly fair computation, because its first subprocess is never enabled; hence this subprocess can perform no effective action. This notion of fairness has nevertheless been studied in the literature [14, 15, 22].

Our method can be applied quite straightforwardly to give an operational characterization of strict fairness in terms of Muller T-automata.

PROPOSITION 4.9: *Let $S=(Q, q_0, D)$ be a transition system over R modelling the behavior of a regular process p, i.e. $S=\mathscr{B}(p)$, then a computation c in S is stricly fair iff:*

- *either it is finite (and maximal);*
- *or it is infinite and satisfies:*

$$\forall i \quad \exists t \in \mathrm{Inf}_t(c),\ t = p'' \xrightarrow{r} s \qquad \text{such that} \quad \pi_i^*(r) \neq 1. \tag{3}$$

PROPOSITION 4.10: *Let $S=(Q, q_0, D)$ be a transition system over R modelling the behavior of a regular process p, i.e. $S=\mathscr{B}(p)$; define a Muller T-automaton as follows: $\mathscr{T}\mathscr{A}_s=(S', Q_T, T_{\mathrm{inf}})$, where S' is deduced from S as*

in proposition 4.3 *and definition* 4.1 (ii), $Q_T = \{ q \in Q / \text{Out}(q) = \varnothing \text{ in } S \}$, *and*

$$T_{\text{inf}} = \{ \pi^*(T) / T = \{ t_i / i \in I \} \in \mathscr{T}(S), \forall, t_i : q_i \xrightarrow{r_i} q_i', \text{ and } \forall j = 1, \ldots, k, \pi_j^*(\bigcup_{i \in I} r_i) \neq \{1\} \}.$$

Let $\mathscr{T}\mathscr{A}_s'$ *be deduced from* $\mathscr{T}\mathscr{A}_s$ *by relabeling transitions by the corresponding action names, as in proposition* 4.3; *then:*

$$L(\mathscr{T}\mathscr{A}_s') = \{ w = trace(c) / c \text{ is a strictly fair computation of } S \text{ starting in state } q_0 \}.$$

COROLLARY 4.11: *The set of strictly fair computations of a (strictly) regular* SCCS *process is an* ε*-free* ω*-regular language.*

4.4. Discussion

Our notions of fairness model the behaviors of SCCS processes, and are designed to this specific end; in particular, the notions of strong and weak fairness are context dependent: e. g., the strongly fair computations of $p \uparrow B$ *cannot* be obtained by restricting to B the strongly fair computations of p. On the other hand, the notion of strict fairness, which we considered mainly for historical reasons, is context independent.

Our notions of fairness meet the criteria suggested in [1].

Our notions of fairness are similar to those defined in [7, 8, 14, 15]. However, most of these papers consider mainly the case of weak fairness, and leave aside the case of strong fairness: the reason is that, in general, strong fairness is more cumbersome to study, because one needs to take into account the whole computation; hence the study of strong fairness cannot usually be "localized" [8]; in our framework though, and because we consider only finite state processes, we easily can finitely describe strong fairness via an automaton.

Historically, the first operational characterization of fairness via ω-regular languages appears in [24], where it is shown that the languages of weakly fair computations of asynchronous digital networks coincide with the class of ω-regular languages.

Our notion of strong fairness is related to the notions considered in [29, 30] but differs from them in several respects. The notions of fairness considered by [29, 30] are all relative to automata with a slightly different acceptance

condition, and are all context independent. Among the notions they introduce, those which are closest to ours are t-fairness, edge-fairness and letter-fairness, which all differ in some respects from our notions: for instance, t-fairness considers only labels of transitions, edge-fairness (resp. letter-fairness) is concerned with *all* transitions (resp. all transitions with a given label) originating in a given state, whereas we consider only some sets of labelled transitions occurring in specific cycles. However, the results are somehow related, in that [29, 30] show that the class of edge-fair computations coincides with the class of ω-regular languages, whereas we show that our classes of strongly fair languages coincide with the class of ε-free ω-regular languages. Some more work would be needed in order to make explicit the relationship between their and our fairnesses.

Note finally that the results of the present paper can be extended to cover other kinds of fairness: we considered here fairness relative to the synchronous $\times$ operator exclusively; we would obtain similar results considering also notions of fairness dealing with the $+$ operator [29, 30]. In that case, we would have to use action redexes instead of derivation redexes.

ACKNOWLEDGMENTS

We thank P. Darondeau and D. Muller, with whom we had fruitful discussions while elaborating this paper, and who influenced decisively its development, and A. Arnold, L. Priese, A. Saoudi, P. Schupp, and the anymous referees for their insightful comments.

REFERENCES

[1] K. Apt, N. Francez and S. Katz, *Appraising Fairness in Languages for Distributed Programming* in Journ. of Distributed Computing (to appear).

[2] A. Arnold and A. Dicky, *An Algebraic Characterization of Transition System Equivalences* (to appear).

[3] J. A. Bergstra and J. W. Klop, Process algebra: *Specification and Verification in Bisimulation Semantics*, in Math. and Comput. Sci. II, Hazewinkel, Lenstra and Meertens éd., CWI Monograph 4, North-Holland, Amsterdam, 1986, pp. 61-94.

[4] G. Boudol and I. Castellani, *On the Semantics of Concurrency: Partial Orders and Transition Systems*, Proc. CAAP 87, Lect. Notes in Comput. Sci, Vol. 249, Springer-Verlag, Berlin, 1987, pp. 123-147.

[5] S. D. Brookes and W. C. Rounds, *Behavioural Equivalence Relations induced by Programming Logics*, Proc. ICALP 83, Lect. Notes in Comput. Sci. Vol. 154, Springer-Verlag, Berlin, 1983, pp. 97-108.

[6] I. Castellani, *Bisimulations and Abstraction Homomorphisms*, Proc. CAAP 85, Lect. Notes in Comput. Sci., Vol. 185, Springer-Verlag, Berlin, 1985, pp. 223-238.

[7] G. COSTA, *A Metric Characterization of Fair Computations in CCS*, Proc. CAAP 85, Lect. Notes in Comput. Sci., Vol. 185, Springer-Verlag, Berlin, 1985, pp. 239-252.

[8] G. COSTA and C. STIRLING, *Weak and Strong Fairness in CCS*, Proc. MFCS 84, Lect. Notes in Comput. Sci. Vol. 176, Springer-Verlag, Berlin, 1984, pp. 245-254.

[9] P. DARONDEAU, *About Fair Asynchrony*, Theor. Comput. Sci., Vol. 37, 1985, pp. 305-336.

[10] S. EILENGERG, *Automata, Languages and Machines*, Academic Press, London, 1974.

[11] N. FRANCEZ, *Fairness*, Springer-Verlag, Berlin, 1986.

[12] R. J. VAN GLABBEEK, *Notes on the Methodology of CCS and CSP*, CWI Tec. Rep. CS-R 8624, Amsterdam, 1986.

[13] I. GUESSARIAN and L. PRIESE, *On the Minimal Number of $\times$ Operators to model regularity in fair SCCS*, in Information Processing Letters (to appear).

[14] M. HENNESSY, *Modelling Finite Delay Operators*, Tec. Rep. CSR-153-83, Edinburgh, 1983.

[15] M. HENNESSY, *Axiomatizing Finite Delay Operators*, Acta Inform., Vol. 21, 1984, pp. 61-88.

[16] M. HENNESSY and G. PLOTKIN, *Finite Conjunctive Non-determinism* (to appear).

[17] B. LESAEC, *Étude de la reconnaissabilité des languages rationnels de mots infinis*, Ph. D. Thesis, Univ. Bordeaux-I, 1986.

[18] R. MILNER, *Calculi for Synchrony and Asynchrony*, Theoret. Comput. Sci., Vol. 25, 1983, pp. 267-310.

[19] R. MILNER, *A Calculus for Communicating Systems*, Lect. Notes in Comput. Sci., Vol. 92, Springer-Verlag, Berlin, 1980.

[20] R. MILNER, *Lectures on a Calculus for Communicating Systems*, Lect. Notes in Comput. Sci., Vol. 197, Springer-Verlag, Berlin, 1982, pp. 197-220.

[21] R. MILNER, *A Complete Inference System for a Class of Regular Behaviors*, J. Comput. and Sys. Sci., Vol. 28, 1984, pp. 439-466.

[22] R. MILNER, *A Finite Delay Operator in Synchronous CCS*, Tec. Rep. CSR-116-82, Edinburgh, 1982.

[23] D. E. MULLER, *Infinite Sequences and Finite Machines*, Proc. 4th IEEE Symp. on switching circuit theory and logical design, New York, 1963, pp. 3-16.

[24] D. E. MULLER, *The General Synthesis Problem for Asynchronous Digital Networks*, Proc. SWAT Conf., 1967, pp. 71-82.

[25] D. E. MULLER, A. SAOUDI and P. SCHUPP, *Alternating Automata, the Weak Monadic Theory of the Tree and its Complexity*, Proc. ICALP 86, Lect. Notes in Comput. Sci., Vol. 226, Springer-Verlag, Berlin, 1986, pp. 275-283.

[26] W. NIAR, *Equités et Automates en CCS*, Thèse de 3^e cycle, Paris, 1988.

[27] D. PARK, *On the Semantics of Fair Parallelism*, Abstract Software Specifications, Lect. Notes in Comput. Sci., Vol. 86, Springer-Verlag, Berlin, 1980, pp. 504-526.

[28] D. PARK, *Concurrency and Automata on Infinite Sequences*, Proc. 5th GI Conf., Lect. Notes in Comput. Sci. Vol. 104, Springer-Verlag, Berlin, 1981, pp. 167-183.

[29] L. PRIESE, R. REHRMANN and U. WILLECKE-KLEMME, *An Introduction to the Regular Theory of Fairness*, Theor. Comput. Sci., Vol. 54, 1987, pp. 139-163.

[30] L. PRIESE, R. REHRMANN and U. WILLECKE-KLEMME, *Some Results on Fairness*, Report TI-1987-38, Univ. Paderborn, 1987.

[31] L. PRIESE and U. WILLECKE-KLEMME, *On State Equivalence Relations in Nondeterministic or Concurrent Systems*, Report TI-1986-34, Univ. Paderborn, 1986.

[32] J. QUEILLE and J. SIFAKIS, *Fairness and Related Properties in Transition Systems: A Time Logic to Deal with Fairness*, Acta Inform., Vol. 19, 1983, pp. 195-220.

[33] W. THOMAS, *Automata on Infinite Objects*, Handbook of Theoretical Computer Science (to appear).