AN AUTOMATON CHARACTERIZATION OF FAIRNESS IN SCCS

by Irène Guessarian * and Wafaa Niar-Dinedane**

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 $\omega$-regular languages, and which characterize the classes of fair languages.

1 - INTRODUCTION

Fairness is both a very complex and widely investigated subject [Francez]. 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 $\times$, requiring that all individual processes which are composed via $\times$ take a step together at all time units [Milner1, Milner2]. 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 [Costa-Stirling], 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 [Arnold-Dicky, Boudol-Castellani, Milner4, Park2, Queille-Sifakis]. 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 [Muller2, Park2, Priese-Rehrmann-Willecke-Klemmel].

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.

* CNRS-LITP, Université Paris 7, 2, Place Jussieu, 75251 Paris Cédex 05, FRANCE.
** LITP, Université Paris 7, 2, Place Jussieu, 75251 Paris Cédex 05, FRANCE.
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 $\omega$-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 $\omega$-regular languages; we show that it coincides with the class of $\varepsilon$-free $\omega$-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 [Priese-Rehrmann-Willecke-Klemmel] 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 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 [Milner1, Hennessy1]. Let $\text{Act} = \langle A, \cdot, ^-, 1 \rangle$ be a non empty commutative group of actions, and $\text{Var}$ a set of variables. The unit action $1$ represents an internal action, for instance the result of a synchronization $a \cdot a$, or a delay of one time unit. The SCCS expressions $E$, ranged over by $E$, are defined by the BNF scheme:

$$E ::= x | \text{NIL} | a : E | E \uparrow B | E + F | E \times F | \text{rec} x .E$$

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

An occurrence of a variable $x$ in an expression $E$ is said to be free if it is not in the scope of a $\text{rec} x$, and it is said to be guarded [Milner2] 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 $p$.

$\text{NIL}$ represents the process which can do nothing, $:$ represents sequential composition, $+$ represents nondeterminism, $\times$ 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. $\text{rec} x .E$ represents the solution of the set of mutually recursive equations $x_i = E_i, i = 1, \ldots, n$. The delay operator $\delta$ is definable in that framework via $\delta E = \text{rec} x .(1x + E)$. See [Hennessy1] for more details.

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 $a \in A$:

$\xrightarrow{a}$

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