MATHEMATICAL PROBLEMS IN COMPUTATION THEORY
BANACH CENTER PUBLICATIONS, VOLUME 21
PWN – POLISH SCIENTIFIC PUBLISHERS
WARSAW 1988

FAIRNESS FOR SYNCHRONOUS FORK-JOIN NETS

M. CHADILI
I. GUESSARIAN

Université Paris VII, France

Synchronous fork-join flow-diagrams are defined and their operational semantics is given using finite automata. Three notions of fairness are relevant: fairness regarding path-behavior, fairness regarding input-output behavior, and a combination of these two notions. Properties of nets and their fair computations are studied.

1. Introduction

In the present paper we address fairness problems for synchronous fork-join parallelism. Fork-join parallelism is the most basic and lowlevel form of parallelism, where a process can fork, i.e. split, into two subprocesses which then proceed independantly; two processes can also join into a single one at some point, namely the final process waits for the results of the two processes racing to the join, and proceeds with the results of either one, nondeterministically (cf. *fork* and *wait* statements in *C*, *PL*1, etc.). There is a major difference between our synchronous join and the asynchronous merge mostly studied in the literature (see e.g. [3]): data which do not find their way through the merge can wait for an arbitrary long time in infinite buffers, whereas data which do not pass through a join are irremediably lost, because we assume:

(1) that all processes are ruled by a global clock and take some action at every time unit;

(2) that buffers are finite (or length 1 to simplify). These assumptions, to our sense, correspond more closely to "real life", but make fairness more difficult to study.

D. Benson gave an algebraic semantics for fork-join parallelism [2]: fairness, however, being an intrinsically operational phenomenon, does not seem to be easily expressible in that formalism. So, we first introduce an operational semantics for fork-join parallelism, using the notion of oracle [3], which then enables us to study fairness.

Also, we allow explicitly for iteration in our formalism, whereas finite nets only are considered in [2], who leaves implicit the case of iterative nets, even though they might be studied by the same methods. We could also deal with recursive nets, but at the cost of heavier machinery, e.g. finite automata and transducers would not any more be sufficient, so we leave that case out of the present study.

Various notions of fairness naturally occur in the present formalism: but they all pertain to the following intuitive notion: an event infinitely often possible does occur infinitely often, cf. the strong fairness of [4]–[6].

We will study two main notions of fairness and the resulting properties for nets of processes.

(1) Fairness with respect to path behavior: every finite path in the net must be executed infinitely often. This is a syntactic and global notion of fairness, well-suited to modelling e.g. fairness of hardware circuits, ensuring that no process will ever starve or get blocked. However, composing fair nets does not result in a fair net.

(2) Fairness with respect to input-output behavior: every data item requesting infinitely often access through a join shall go through that join infinitely often. This is a semantic and local notion of fairness, which could model e.g. fairness in a message passing protocol. This second notion is stable by composition, but there exist no behavior globally fair with respect to all inputs.

Now, for the asynchronous case with infinite buffers, the first notion of fairness implies the second one. This is not true any more in the synchronous case, where the two notions are incomparable. Hence, we define a "strong" fairness by combining the two previous notions.

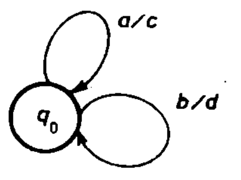
The paper is organized as follows: we first describe the semantics of fork-join nets and show it can be characterized by finite state transducers, which give a semantics in terms of infinitary languages. We then study various notions of fairness, their properties and relationships. Finally, we give in an appendix a few useful properties of infinitary languages. This paper has been written with a bias in favor of intuition in order to improve readability. Consequently, some formal definitions and proofs are skipped and replaced by examples. Proofs and technical details will be given in an extended version of the paper.

2. Fork-join nets and their semantics

Our nets will consist of three basic components, processes, forks and joins connected together in a finite flowdiagram organization.

A *process* is a one state transducer which transforms an infinite stream of data into an infinite stream of results: processes will be considered as unary functions in the sequel.

EXAMPLE 2.1.

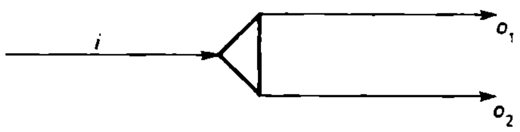


with input alphabet $\{a, b\}$ and output alphabet $\{c, d\}$ corresponds to the function

$$f(x) = \begin{cases} c & \text{if} \quad x = a, \\ d & \text{if} \quad x = b, \end{cases}$$

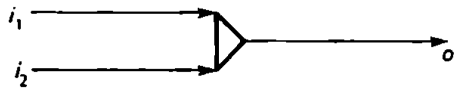
and will transform the input $(ab)^\omega$ into the output $(cd)^\omega$. ∎

A *fork* is an operator with one input channel and two output channels, which instantaneously duplicates the contents of its input channel into its output channels. A fork will be represented by the diagram:

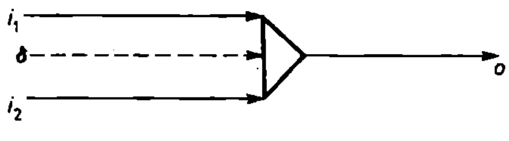


and will transform the input $(ab)^\omega$ into the outputs $((ab)^\omega, (ab)^\omega)$. However, since we will assume in the sequel that channels are length one buffers, both input and outputs can only appear at the rate of one per time unit.

A *join* is an operator with two input channels and one output channel, which, at each time unit, nondeterministically chooses one of its inputs and outputs it, while dropping out the input which was not chosen; this whole process takes one time unit, contrary to the fork which was supposed to be instantaneous. A join will be depicted as follows:



If we feed the previous join with inputs $i_1 = ab$, $i_2 = cd$, the possible output will be one among $\{ab, cd, ad, cb\}$. In order to make deterministic the behavior of the join, we add a third, fictive, input, the *oracle* δ, which tells which input is picked, so that, e.g.



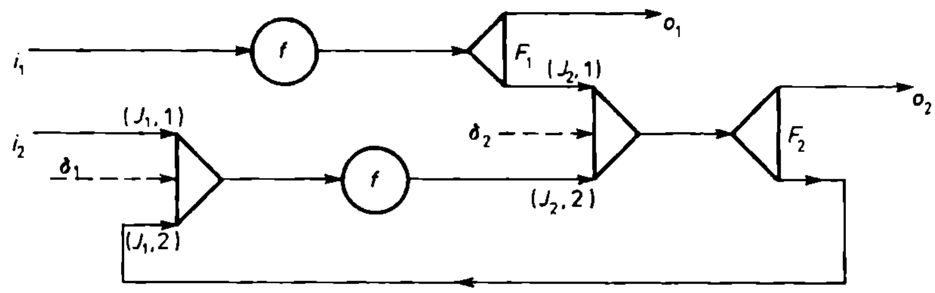
with $i_1 = (ab)^\omega$, $i_2 = (cd)^\omega$, $\delta = (01)^\omega$ will output $o = (ad)^\omega$.

DEFINITION 2.2. A *net* N is a finite flowdiagram built up from processes, forks and joins, which we represent as a quintuple:

$$N = \langle I, O, n, m, L\rangle,$$

where $I = (i_1, \ldots, i_k)$ is the vector of input channels, $O = (o_1, \ldots, o_l)$ is the vector of output channels, n is the number of joins, m is the number of forks, L is the set of all labels in the net. ∎

EXAMPLE 2.3.



$$I = (i_1, i_2), \qquad O = (o_1, o_2), \qquad n = m = 2,$$

$$L = \{f, (J_1, 1), (J_1, 2), (J_2, 1), (J_2, 2), F_1, F_2, i_1, i_2, o_1, o_2\}. \ ∎$$

Nets can thus be considered as finite state transducers with several input channels and several output channels, or as transition nets. An equivalent algebraic definition could be given, saying that nets are the least structure containing processes, fork, join, projections and closed under tupling, composition and iteration. The advantage of that second definition is that it can be more readily generalized to allow for recursion. However, the definition with transition nets presents the advantages of being more intuitive and more handy for describing the operational semantics of nets. Since our main interest is fairness, which is intrinsically operational, we will use Definition 2.2 only.

The operational semantics is based on Arnold's work [1] and the following assumptions:

(i) nets are synchronous and there is a global clock: i.e., at each time unit, all operations advance by one step;

(ii) a letter takes one unit of time to go through a join, or to be transformed by a process;

(iii) a letter takes no time to go through a fork.

Modulo the above assumptions, a net can be considered as a transducer based on a finite automaton: it performs a transduction, transforming each vector I of infinite input words into a vector O of infinite output words; this transformation is by definition the semantics of the net.

More precisely, each oracle δ in $(\{0, 1\}^n)^\omega$ determines a *computation* of the net for every input i in $(A^k)^\omega$: the result of this computation is an output o in $(T^l)^\omega$, where A is the input alphabet and T the output alphabet. The semantics of the net is the set of all pairs (i, o), for all possible inputs i and oracles δ.

PROPOSITION 2.4. *If the inputs and the oracle of a join are regular ω-languages, the output of the join is a regular ω-language.*

Proof. A suitable product of the automata recognizing the inputs and the oracle gives an extended Büchi automaton recognizing the output. ∎

For the next proposition, we need to assume that the input and output alphabets of all processes in a net are finite, so that the net has only a finite number of possible behaviors for arbitrary inputs of bounded length; then

PROPOSITION 2.5. *If all inputs and oracles of a net with finite alphabets are regular ω-languages, then the outputs of the net are regular ω-languages too.*

The proof proceeds on the same ideas as the previous proposition, except that one has to add one more component in the product automaton: this new component simulates the behavior of the net. ∎

3. Fairness

3.1. Fairness with respect to path behavior. The first notion of fairness which we will study is a global notion, ensuring that a computation will go through every channel infinitely often. In order to express it easily, we introduce the notion of path in a net. To this end, we label the input channels with pairwise different names; all other channels are implicitly distinguished by the labelling of the forks and joins (we could almost forget about the processes and keep only the fork-join structure of the net).

Equivalently, the first notion of fairness that we study is relative to streams of data of the form x_p^ω on input channel i_p for every p.

DEFINITION 3.1. A *path* in a net N is a word $w = w_1 \ldots w_n$ in L^+ such that $w_1 = i_j$, $w_n = o_i$, for some input channel i_j and output channel o_i and for each $p = 1, \ldots, n-1$, there is a channel linking directly w_p to w_{p+1}. P_N denotes the set of finite paths in the net N. ∎

EXAMPLE 3.2. Let N be the net of Example 2.3; then $i_1 f F_1 o_1$, $i_2(J_1, 1)$ $f(J_2, 2)$ $F_2(J_1, 2)$ $f(J_2, 2)$ $F_2 o_2$ are two paths in P_N. ∎

PROPOSITION 3.3. *The set P_N is a regular language.* ∎

We now state intuitively the goal we wish to achieve with fairness with respect to path behavior. A computation is fair if condition (i) is satisfied.

(i) Every input letter x_p, which could, after transformations in a path, arrive infinitely often to an output channel o_i, does arrive infinitely often to that output channel.

We will in the sequel give sufficient conditions for (i) to become true.

A path is *active* in a computation iff it is taken infinitely often during that computation. More formally, define for each net N the alphabetic morphism φ_N: $L \to P((\{0, 1\}^n)^*)$ by $\forall x \in L$

$$\varphi_N(x) = \begin{cases} \{\{b_1, \ldots, b_{i-1}, a, b_{i+1}, \ldots, b_n\}, b_i \in \{0, 1\}\} & \text{if} \quad x = (J_i, a), \\ \{0, 1\}^n \quad \text{if} \quad x = f, \text{ where } f \text{ is a process}, \\ \emptyset \quad \text{otherwise}, \end{cases}$$

φ_N is alphabetically extended to L^*.

Definition 3.4. (i) Let N be a net and δ an oracle in $(\{0, 1\}^n)^\omega$: δ is *path-fair* iff for every path p in P_N, $|\delta|_{\varphi_N}(p) = \omega$, where $|\delta|_B$ is the number of times words of B occur in δ.

(ii) a computation is path-fair iff the corresponding oracle is, or iff, equivalently, every path is active in that computation. ∎

Proposition 3.5. *Path-fairness implies condition* (i). ∎

When the output alphabets of all processes in the net are finite, we can replace the previous notion of fairness by a weaker one, which nevertheless implies condition (i). Namely, instead of demanding that every finite path be active, it is enough to require that all paths in a finite set of finite paths be active.

Definition 3.6. A path $p = w_1 \ldots w_n$ with $w_1 = i_j$ and $w_n = o_i$ is *useful* iff $w_2 \ldots w_{n-1} = w' w^q w''$ for some q and $w = w_k \ldots w_l$ such that

(i) for all x in L, if x is not a process $\sup\{|w|_x, |w'|_x, |w''|_x\} \leqslant 1$;

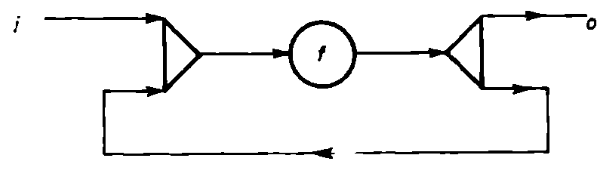
(ii) if $f_1, \ldots, f_r$ are the processes occurring in w, and n_i is the cardinal of the output alphabet of f_i, then $q \leqslant \inf\{n_i\colon i = 1, \ldots, r\}$. ∎

Intuitively, a useful path can go through a given loop at most q times, where q is the minimum of the cardinals of the output alphabets of the processes in that loop. Useful paths are sufficient to ensure that inputs will undergo all possible transformations before being output. Notice first that the set of useful paths is finite, hence this will lead to an effective notion of fairness.

Definition 3.7. A computation of a net N is *weakly path-fair* iff all usefull paths are active. ∎

Proposition 3.8. *If the output alphabets of all the processes in a net are finite, then weak path-fairness implies condition* (i). ∎

Examples 3.9. Let N be the following net

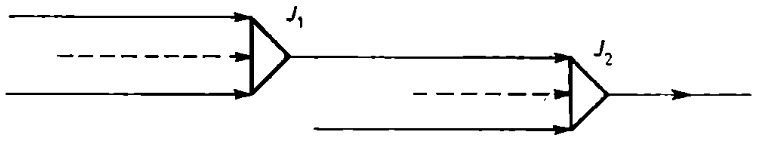


where f has input and output alphabets $A = \{a, b\}$ and is defined by $f(a) = b$, $f(b) = a$. Then, the set of useful paths in N is

$$P_u = \{i(J, 1)fFw^l o, \text{ where } l = 0, 1, 2 \text{ and } w = (J, 2)fF\}.$$

PROPOSITION 3.10. *Path-fairness (weak or not) is not closed under composition.*

This is shown by the following example: let N be

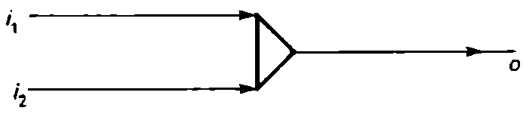


$\delta_1 = (01)^\omega$ and $\delta_2 = (10)^\omega$ are path-fair when we consider each join separately. But $\delta = ((0, 1)(1, 0))^\omega$ is not path-fair for the net N, because the path $(J_1, 2)$ $(J_2, 1)$ will never be taken. ∎

In the rest of this section we will assume that all input and output alphabets are finite.

3.2. Fairness with respect to input-output behavior. Because of the synchronous behaviors of our nets, and of the lack of infinite buffers, path-fairness does not imply fairness with respect to input-output behavior, as it does for the case of the asynchronous merge (cf. [3]). This is shown by the following:

EXAMPLE 3.11. N consists of a single join



Then $\delta = (01)^\omega$ is path-fair; however, if $x_1 = (ab)^\omega$, $x_2 = (cd)^\omega$, then the output is $(ad)^\omega$, and b and c will never go through the join. ∎

To remedy this drawback, we define a local notion of fairness with respect to input-output behavior, which ensures that, whenever a letter "knocks" infinitely often at the door of a join, coming from either channel, it will go through infinitely often.

DEFINITION 3.12. A computation of a net N is *IO-fair* for input I iff, for each join in the net, every letter which comes infinitely often on either one of the input channels of the join goes also infinitely often through the output channel of that join. ∎

Remarks. (i) In an *IO*-fair computation, for every channel c of the net and every letter x, if x could go infinitely often through c, then x must go infinitely often through c.

(ii) IO-fairness is always relative to some input I.

PROPOSITION 3.13. *Path-fairness and IO-fairness are incomparable.*

Proof. Example 3.11 shows that path fairness does not imply IO-fairness: the converse implication is also false as shown by the following example: let N be as in Example 3.11 and $i_1 = i_2 = a^\omega$, $\delta = 0^\omega$. Then δ defines an IO-fair computation which is not path-fair, since the path $i_2(J_2, 2)o$ is never taken. ∎

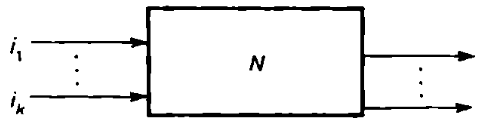
DEFINITION 3.14. Let $F_N(I)$ be the set of oracles which define IO-fair computations for input I. ∎

PROPOSITION 3.15. *For all I, $F_N(I) \neq \emptyset$, i.e., for each input there exists at least one IO-fair computation.*

Proof. We only have to ensure fairness at each join: this is done locally by endowing each join with an algorithm which defines the oracle at that join so that IO-fairness is guaranteed. The idea of the algorithm is as follows: if $(x_1, \ldots, x_q)$ are all the letters which could arrive at the join, we define a list $(n_1, \ldots, n_q)$ of integers, where n_i gives the "priority" rank of letter x_i for going through the join; n_i simply counts the number of times letter x_i tried unsuccessfully access to the join. The oracle is then defined in order to let through the letter with the higher priority, while resetting the priority of that letter to 0. ∎

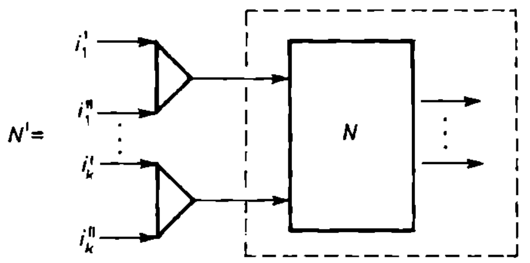
PROPOSITION 3.16. *For every net N and inputs I, I', $F_N(I) \cap F_N(I') \neq \emptyset$.*

Proof. Let N be represented as follows:



and let $I = (x_1, \ldots, x_k)$, $I' = (x'_1, \ldots, x'_k)$.

Define N' by



where the dotted square contains a copy of N.

Then, every computation of N' which is IO-fair for the input $(x_1, x'_1, \ldots, x_k, x'_k)$ defines, by restriction to the subnet N, a computation of N which is fair with respect to both inputs I and I'. ∎

COROLLARY 3.17. *For every finite set K of inputs, $K \subset A_1^\omega \times \ldots \times A_k^\omega$, where A_j is the input alphabet for channel i_j, $\bigcap_{I \in K} F_N(I) \neq \emptyset$.* ∎

The next proposition expresses there is no oracle which is globally *IO*-fair with respect to all inputs.

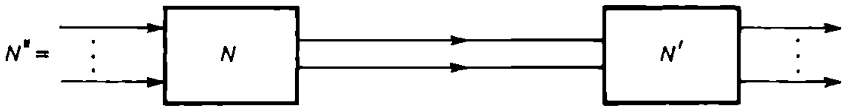
PROPOSITION 3.18. *If at least one of the joins in a net N has an output alphabet containing at least two elements, then $\bigcap_{I \in A} F_N(I) = \emptyset$, where $A = A_1^\omega \times \ldots \times A_k^\omega$, where A_j is the input alphabet for channel j.*

Proof. We can then extract from N a join, possibly linked to input channels via processes but no loops, whose output, and hence input, alphabets contain at least two letters, say $\{a, b\}$. Let $\delta = 0^{i_1} 1^{j_1} 0^{i_2} 1^{j_2} \ldots$ be an arbitrary oracle for that join J and consider the net N' consisting of the single join J. Then, letting $I = (x_1, x_2)$ with $x_1 = a^{i_1} b^{j_1} a^{i_2} b^{j_2} \ldots$ and

$$x_2 = \begin{cases} a^\omega & \text{if} \quad |x_1|_b = \omega, \\ a^q b^\omega & \text{if} \quad x_1 = a^{i_1} b^{j_1} \ldots a^{i_p} b^{j_p} a^\omega \text{ with } i_1 + j_1 + \ldots + i_p + j_p \leqslant q, \end{cases}$$

δ is not *IO*-fair for I since b will never be output. Hence, for subnet N', and also thus for net N, every oracle is *IO*-unfair for some input. ∎

PROPOSITION 3.19. *IO-fair computations are closed under composition.*
Proof.



Let N, N' be two nets, and N'' be the net obtained by feeding the outputs of N as inputs to N'. Then, if δ is in $F_N(I)$ and δ' is in $F_{N'}(I')$, where I' is the output of the computation of N defined by δ, $\delta'' = \delta \times \delta'$ is in $F_{N''}(I)$, where, if $\delta = a_1 a_2 \ldots \in (\{0, 1\}^n)^\omega$ and $\delta' = a'_1 a'_2 \ldots \in (\{0, 1\}^{n'})^\omega$, $\delta'' = (a_1 a'_1)(a_2 a'_2) \ldots \in (\{0, 1\}^{n+n'})^\omega$. ∎

Now, even though Proposition 3.18 showed us that there is no oracle globally *IO*-fair for all inputs, we can show that there are oracles *IO*-fair with respect to classes of inputs. One such class is the class of ultimately periodic inputs.

LEMMA 3.20. *Let N be a net, $w = (w_1, \ldots, w_k)$ a vector of finite input words such that $|w_1| = \ldots = |w_k| = l$, and let w^ω be the infinite input $(w_1^\omega, \ldots, w_k^\omega)$; let $(\sigma_1, \ldots, \sigma_k)$ in Σ^k be arbitrary circular permutations of $\{1, \ldots, l\}$ and let $\sigma(w)^\omega$ be the vector of inputs $(\sigma_1(w_1)^\omega, \ldots, \sigma_k(w_k)^\omega)$. Then $\bigcap_{\sigma \in \Sigma^k} F_N(\sigma(w)^\omega)$ is nonempty and contains an oracle δ of the form $\delta = \eta^\omega$.*

Proof. As in Proposition 3.16, we first construct a net N' with an input combining all the $\sigma(w)^\omega$, for σ in Σ^k; this is possible since there are at most l^k

such vectors $\sigma(w)$. We then use the algorithm in Proposition 3.16 to compute a fair oracle; everything being periodic, the oracle will also be periodic. ∎

This lemma asserts that there exists a single periodic oracle *IO*-fair for all inputs of the form $\sigma(w)^\omega$. The restriction that all the w_i have the same length is inessential; if the periods are different, it suffices to take as common period their least common multiple.

LEMMA 3.21. *Let* $I = (x_1 w_1^\omega, \ldots, x_k w_k^\omega)$, *where* $w = (w_1, \ldots, w_k)$, σ *and* δ *are as in Lemma* 3.20; *then* $\delta \in F_N(I)$.

Proof. Let $q = \max\{|x_i|: i = 1, \ldots, k\}$ and let m be the first multiple of $|\eta|$ greater than q.

There exist $u = (u_1, \ldots, u_k)$ and σ in Σ^k such that for $i = 1, \ldots, k$,

$$|u_i| = m, \qquad I = \left(u_1 \sigma_1 (w_1)^\omega, \ldots, u_k \sigma_k (w_k)^\omega\right) = u\sigma(w)^\omega.$$

Notice now that $\eta^\omega = \delta \in F_N(\sigma(w)^\omega)$; moreover, fairness being concerned with infinitary behavior, we can forget about the first m components of I and δ, hence all oracles δ' in $\Delta = (\{0, 1\}^n)^m \delta$ are in $F_N(I)$; in particular, δ itself being in Δ, is in $F_N(I)$. The following Fig. 1 representing the lengths of w, η and I might help. ∎

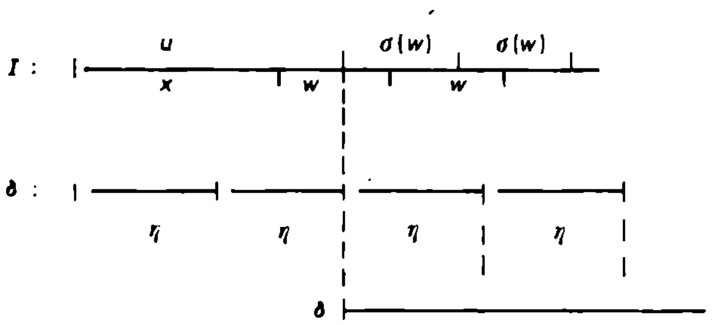


Fig. 1

THEOREM 3.22. *Let L be the set of inputs of the form* $I = uw^\omega$, *where* $u = (u_1, \ldots, u_k)$ *is an arbitrary k-tuple of finite words, and* $w = (w_1, \ldots, w_k)$ *is a k-tuple of words such that* $|w_1| = \ldots = |w_k| = l$; *then* $\bigcap_{I \in L} F_N(I) \neq \emptyset$.

Proof. Lemma 3.21 shows that δ is in $F_N(I)$ for each I in L. ∎

Note that

(i) we can relax the hypothesis that all w_i have the same length l,

(ii) we can, more generally, assume that w is an element of a finite set W of finite k-tuples $(w_1, \ldots, w_k)$, instead of w being always the same k-tuple.

The proof of Lemma 3.20, and hence of the theorem, is the same under those weaker hypotheses.

Note finally that the oracles are perfectly effective, and might be called schedulers.

3.3. Strong fairness. We noticed in Proposition 3.13 that both *IO*-fairness and path-fairness allow for some injustice (respectively for path-fairness and *IO*-fairness). In the present subsection, we will henceforth combine these two notions of fairness into a notion of strong fairness. Recall that all input and output alphabets are finite, so that path-fairness amounts to all useful paths being active. In order to define strong fairness we will keep track of the path a letter has gone through before arriving in a channel, as long as that path is a useful path or a prefix thereof.

Example 3.23. Let N be as in Example 3.9, with input $(ab)^\omega$, oracle 001^ω. Then, the pairs (p, x) of the letter x which is in the channel $[F, (J, 2)]$ together with the prefix p of useful path x has already gone through, are as follows for the successive time units: at time 1, the channel is empty, so $(p, x) = \emptyset$

at time unit 2: $(i(J, 1) fF, b)$,
at time unit 3: $(i(J, 1) fF, a)$,
at time unit 4: $(i(J, 1) fFw, a)$,
at time unit 5: $(i(J, 1) fFw, b)$,
at time unit 6: $(i(J, 1) fFw^2, b)$,
at time unit 7: $(i(J, 1) fFw^2, a)$,
where $w = (J, 2) fF$.

At the subsequent time units, we no longer keep track of the path, because it is no more a useful path, hence the history of that channel starting from time unit 8 and on will be $((*, a)(*, b)(*, b)(*, a))^\omega$. ∎

Definition 3.24. Let N be a net, $P_u(N)$ the set of prefixes of useful paths in N, I an input. A computation is *strongly fair* with respect to I iff: for every channel c, and every history $h = (p, x)$, where $p \in P_u(N) \cup \{*\}$ describes the portion of useful path already taken by letter x, if h could go infinitely often through channel c, then h must go infinitely often through channel c. Let $SF_N(I)$ be the set of oracles defining strongly fair computations for I. ∎

Example 3.23 (continued). 001^ω is *IO*-fair for the input $(ab)^\omega$, but it is neither path-fair, nor strongly fair. The corresponding output is $b(a^2b^2)^\omega$. Now the oracle $\delta = (0^2 1^4)^\omega$ is strongly fair, path-fair and *IO*-fair; the sequence of histories corresponding to channel $[F, (J, 2)]$ is $[(p, b), (p, a), (pw, a), (pw, b), (pw^2, b), (pw^2, a)]^\omega$, where $p = i(J, 1) fF$ and $w = (J, 2) fF$ as above; the corresponding output is $(baabba)^\omega$. ∎

Basically all the propositions and theorems we proved for *IO*-fairness also hold for strong fairness with similar proofs. We state some of them.

Proposition 3.25. *For every net N and input I, $SF_N(I) \neq \emptyset$.*

The proof is similar to the one for Proposition 3.15; the algorithm for finding fair oracles now has to keep track not only of the number of times a letter tried to go through a join, but also of the previous history of that letter. Hence, instead of having n different algorithms, one for each join in the net, we have a single global algorithm, which chooses all the oracles in the net, while keeping track of all useful histories in its priority list. This list stays bounded because once a useful path is completed, the corresponding history becomes $(*, a)$ and $*$ acts as zero for path composition, i.e., extending $*$ by any path again results in $*$. ∎

PROPOSITION 3.26. *For every finite set K of inputs* $\bigcap_{I\in K} SF_N(I) \neq \emptyset$. ∎

PROPOSITION 3.27. *A strongly fair computation is both IO-fair and weakly path-fair.*

Proof. Since there are only a finite number of pairs $h = (p, x)$, including the pairs $(*, x)$, then:

(i) if a letter x comes infinitely often to a join, then some $h = (p, x)$ (with possibly $p = *$), will also come infinitely often to that join, hence h will go through infinitely often because of strong fairness; whence IO-fairness;

(ii) assume now c is strongly fair but not weakly path-fair. Let p be a shortest useful path which is taken only finitely many times; p cannot be of length one, since inputs are infinite streams. Hence p is of the form $p = p'\varphi$, where p' is taken infinitely often; p' can only output a finite number of different letters, hence for some letter x, $(p'\varphi, x)$ is possible infinitely often, hence is taken infinitely often by the strong fairness assumption; p is thus also taken infinitely often, a contradiction. ∎

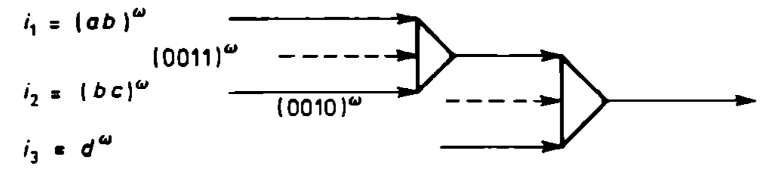
COROLLARY 3.28. (i) $\bigcap_{I\in A} SF_N(I) = \emptyset$,

(ii) *strongly fair computations are not closed under composition.*

Proof. (i) Holds for IO-fair computations.

(ii) See the counter example in Remark 3.29 (i). ∎

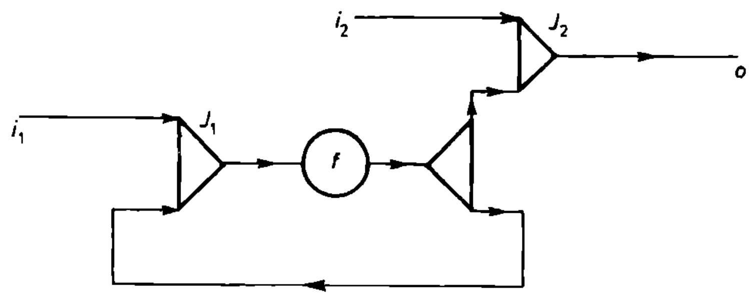
Remark 3.29. (i) The converse of Proposition 3.27 is false. Let N be the following net, with the indicated inputs and oracles



Then the output is $(abcd)^\omega$ but the computation is not strongly fair, because the history $((J_1, 2)(J_2, 1), b)$ is never taken. Moreover, the oracles define strongly fair computations if each join is considered as a separate net, but the composition is not strongly fair.

(ii) It is essential, in the definition of strong fairness, to demand that, if

$(*, x)$ could go infinitely often through a channel c, then it must go infinitely often in that channel. If we relax that requirement, the following counter-example shows that strong fairness does not any more imply IO-fairness. Let N be



where f is the identity (and just introduces a delay of one time unit), and the input and output alphabets are $\{a, b\}$. Let us assume that $I_1 = ab^{\omega}$ and $I_2 = b^{\omega}$, then, if we only require that histories (p, x), with p a prefix of a useful path, be taken infinitely often, the oracle:

$$\delta = (0, 0)[(0, 1)(1, 0)(1, 1)(1, 0)]^{\omega}$$

defines a "strongly fair" computation which is not IO-fair. The oracle corresponding to the join J_1 is $0(01^3)^{\omega}$, corresponding to the output $\varepsilon^2(ab)^{\omega}$ on channel $[F, (J_2, 2)]$. The oracle corresponding to join J_2 is $(01)^{\omega}$, corresponding to output b^{ω} on channel o, hence a will never pass join J_2; this is due to the fact that a arrives to that join with increasingly long histories of the form (pw^n, a), with $p = I_1(J_1, 1)fF$, $w = (J_1, 2)fF$ and $n = 0, 1, 2, \ldots$

(iii) A last remark is to note the essentiality of the synchronous behaviors of all operators: namely no process or join is ever blocked, and all operators take a step together at every time unit. Of course this presupposes the inputs are also infinite nonempty streams.

Appendix. Infinitary languages

We recall some definitions about regular ω-languages. Let A be a finite nonempty alphabet, we consider an infinite word w over A as map

$$w\colon N \dashrightarrow \Sigma$$

such that $w(i)$ is the ith letter of w:

A^*: the set of finite words on A,

A^{ω}: the set of infinite words on A,

$A^{\infty} = A^* \cup A^{\omega}$, $A^+ = AA^*$.

We denote $|w|_a$: the number of occurrences of a in w.

DEFINITION A.1. *A Büchi automaton* is a structure

$$A = (Q, A, q_0, \delta, Q_{\text{inf}}),$$

Q: the set of states (finite),
A: finite alphabet,
$q_0 \in Q$: initial state,
δ: transition function,

$$\delta\colon Q \times A \dashrightarrow 2^Q,$$

Q_{inf}: the set of infinitary states.

DEFINITION A.2. For $w \in A^\omega$, we call *computation* of A over w a map $c\colon N \dashrightarrow Q$ such that

(i) $c(0) = q_0$,
(ii) $c(i+1) \in \delta(c(i), w(i))$,

and a computation is successful iff $\inf(c) \cap Q_{\text{inf}} \neq \emptyset$, where $\inf(c)$ is the set of states such that: $q \in \inf(c)$ iff there exists an infinity of $i \in N$ such that $c(i) = q$.

For a Büchi automaton A, the set of all ω-words in A^ω such that there exists a successful computation of A over w, is denoted by $\|A\|$, and called the *ω-language* accepted by A. An ω-language is said to be *ω-regular* if it is accepted by some Büchi automaton.

DEFINITION A.3. An extended *Büchi automaton* is a structure $A = (Q, A, q_0, \delta, Q_1, \ldots, Q_n)$ such that Q, A, q_0 and δ are as in Definition A.1 and $Q_1, \ldots, Q_n$ are sets of infinitary states.

DEFINITION A.4. A computation of A is successful iff $\forall i = 1, \ldots, n$ $\inf(c) \cap Q_i \neq \emptyset$.

PROPERTY A.1. *Büchi automata and extended Büchi automata accept the same class of languages.*

References

[1] A. Arnold, *Sémantique des processus communicants,* RAIRO Informatique Théorique 15 (1981), 103–139.
[2] D. Benson, *Studies in Fork-join parallelism,* report CS-82, 101, WSU (1982).
[3] F. Boussinot, *Réseaux de processus avec mélange équitable: une approche du temps réel,* Thèse d'Etat Paris (1981).
[4] G. Costa, C. Stirling, *Weak and strong fairness in CCS,* MFCS 84, LNCS 176, Springer Verlag, Berlin (1984), 245–254.
[5] P. Darondeau, *About fair Asynchrony,* TCS 37 (1985), 305–336.
[6] M. Hennessy, *Modelling finite delay operators,* rep. CSR-153-83 (1983).

Presented to the semester
Mathematical Problems in Computation Theory
September 16–December 14, 1985