Fairness in LOTOS

Cheng Wu, Gregor v. Bochmann
Departement d'informatique et de recherche operationnelle
Universite de Montreal
Montreal, P.Q., Canada

Abstract

Fairness is an important concept related to specification languages which are based on concurrent and non-deterministic computation models; it is related to liveness. In this paper we formally introduce fairness to the LOTOS specification language by employing the standard LOTOS semantics together with a formalism which states restrictions on fair infinite execution sequences. We extend three fairness concepts of CSP, namely process, guard and channel fairness, to LOTOS. Certain features of LOTOS, such as the dynamic creation of processes, the dynamic relation between gates and processes, and related membership in multi-way rendezvous, not present in CSP, make the definition of fairness difficult. We introduce the concept of "transition groups", which leads to a general notion of fairness, and use LOTOS action indexes to define the concepts of process, alternative and channel for LOTOS. We explain how a fair execution model for LOTOS can be obtained, and demonstrate the use of these concepts by showing how fairness assumptions can be used to prove liveness properties for a given LOTOS specification.

1. Introduction

Specification languages such as LOTOS[LOTOS87], CSP[Hoar78], etc. are used to describe distributed systems which are based on concurrent and non-deterministic computation models. Fairness is an important property related to such models, which is related to liveness [Fran86]. Liveness properties are usually described as "good things will eventually happen" and fairness properties are usually described as "if something is always or infinitely often ready then it will eventually happen". So, if one can show that a "good thing" meets some conditions such as being always or infinitely often ready, then fairness assumptions will lead to the related liveness properties, that is, "it will eventually happen". For example, a fairness property of an unsafe channel could be "if a sender insist (always) putting messages into the channel, the receiver will receive infinitely many messages". It is obvious that the liveness property of "message *A* will eventually reach its receiver through the unsafe channel" holds if the sender repeatedly puts message *A* into the channel.

Besides to obtain proof systems, another motivation for studying fairness properties is to design execution models or algorithms which provide fair execution. This makes sure that properties, which are proved to hold at the specification level based on fairness assumptions, will also hold for the implementation.

A variety of fairness properties have been proposed in the context of different specification languages. For example, three kinds of fairness, namely process fairness, guard fairness and channel fairness, are defined for CSP [Kuip83]; process fairness has also been defined in CCS [Cost84, Cost84a]. So far, little work has been done about fairness in LOTOS.

LOTOS [LOTOS87][Bolo87] is an FDT (Formal Description Technique) which is standardized within ISO (International Standard Organization). It was developed for formally specifying communication protocols and services in the context of the OSI (Open System Interconnection) standards. However, it is suitable for many other applications in distributed systems. A LOTOS specification usually describes two aspects. The aspect of interaction parameters and associated data structures and operations is described using an abstract data type formalism. The aspect of the temporal ordering of the interactions is described using a formalism close to CCS [Miln 80]. In this paper, we are principally concerned with the latter aspect.

LOTOS prescribes infinite execution sequences without regard to fairness. For example, for the specification of an unsafe channel

```
process unsafe_channel[send, receive]:noexit:=
    send; (i; unsafe_channel[send,receive]
          []
          receive; unsafe_channel [send,receive])
endproc
```

the execution sequence **send, i, send, i, ...** is allowed by LOTOS semantics, which means the channel may loose all input messages.

Certain features of LOTOS make the definition of fairness difficult. Recursion in LOTOS can be used to define a loop (e.g. **P[g]:= g;P[g]**), as well as the dynamic creation of processes (e.g. **P[g]:= g; (g; stop) ||| P[g]**). Gates in LOTOS do not uniquely determine the process performing the action (e.g. **P[g]:= (g; stop) ||| (g;stop)**). In addition, multi-way rendezvous is allowed in LOTOS and the number of processes involved in a rendezvous may change dynamically from one occurrence of a rendezvous to another. Because of all of these aspects, it is not very clear what fairness means in the context of LOTOS specifications.

In this paper, we introduce fairness to LOTOS. We do not mean by this to change the current standard definition of LOTOS, but rather to introduce fairness restrictions. We follow the approach of the so-called 'two level semantics', that is, we first introduce an unfair model, and then define certain restrictions which must be satisfied for fair execution sequences. We employ the standard LOTOS semantics [LOTOS87] at the first level. We define process, alternative (guard) and channel fairness in LOTOS by defining suitable restrictions. They are extensions of the corresponding fairness concepts in CSP [Kuip83]. We also show the application of the fairness definitions, namely the proof liveness properties and the construction of fair execution models.

The paper is organized as follows. Section 2 is a short introduction of linear temporal logic and labelled transition systems, two formal systems which are used in this paper. In Section 3, we introduce the concept of "groupings transitions", which provides a framework for defining fairness. We describe the intuitive meanings of our fairness definitions for LOTOS in Section 4, and define these concepts formally in Section 5. Section 6 shows the proof liveness properties based on fairness assumptions. Section 7 discusses the construction of fair execution models. Finally, Section 8 contains some conclusions.

2. Theoretical framework

2.1. Linear Temporal Logic

Fairness are properties related to infinite execution histories. Usually so-called "weak fairness" and "strong fairness" properties are described informally as "if permanently (always) A then eventually B" and "if infinitely often A then eventually B" respectively. Temporal Logic can be used to describe temporal concepts such as *permanently(always)*, *infinite often* and *eventually* [Fran86][Ku[redacted]5][[redacted]arr85]. I[redacted]he following we will give a brief introduction to Linear Temporal Lo[redacted][P[redacted]e79] w[redacted] will [redacted] used to define fairness in the rest of this paper.

Besides ordinary logical operators ([redacted]), (and), ([redacted]t), and ⇒ (imply), Linear Temporal Logic uses two temporal operators ◊ and □. The expression □ P (read "henceforth P") means that P is true now and will always be true in the future, and ◊ P (read "eventually

P") means that P is true now or will be true sometimes in the future. They are usually interpreted based on a computation model of state sequences. Let $\sigma = <s_1, s_2, ... >$ be the sequence of states for a particular execution history of the system, and P be a boolean assertion on s_i (i= 1,2, ...). We have the following definitions:

P	iff P is true for s_1
$[]$ P	iff $\forall$ i (P is true for s_i)
$\Diamond$ P	iff $\exists$ i (P is true for s_i)

The temporal concepts mentioned above can be described by the two temporal operators and their combinations. For example, $\Diamond[]$ P means from a certain time onwards permanently P. Its formal interpretation is

$\Diamond[]$ P iff $\exists$ j $\forall$ i>j (P is true for s_i);

$[]\Diamond$ P means infinitely often P. Its formal interpretation is

$[]\Diamond$P iff $\forall$ j $\exists$ i>j (P is true for s_i);

And $\Diamond$ P means eventually P. Its formal interpretation is given above.

Now we are able to define the concepts of "weak fairness" and "strong fairness", mentioned above, in terms of temporal logic. A property of the forms $\Diamond[]\, A \Rightarrow \Diamond B$ is a weak fairness assertions, and a property of the form $[]\,\Diamond A \Rightarrow \Diamond B$ is a strong fairness assertions.

2.2. Labeled transition systems

In this section, we consider Labeled Transition Systems, the model in which the LOTOS semantics is defined[LOTOS87]. We will show how the concepts of temporal logic can be used in this context to define fairness properties.

Definition 2.1

A *labeled transition system* is a triple **LTS=(S,T,**{-**t**->}$_{t\in T}$) where,

- **S** is a countable set of states
- **T** = {**t1**, **t2**, ...} is a set of labeled transitions
- {-**t**->}$_{t\in T}$ is a set of binary relations on S ($S\times S \supseteq$ -**t**->) in bijection with the labeled transitions.

A sequence of transitions, starting from some initial state of the transition system, defines at the same time a sequence of states, which are reached after each of the transitions. This sequence of states may be used to defined temporal properties, using the formalism of temporal logic. One particular state predicate, we are interested in, indicates whether a transition of a given label is enabled in a given state, as defined in Definition 2.3. However, we are also interested in the question whether a given type of transition has been executed, as defined in Definition 2.4, however, this information is not directly visible from a given state. Therefore we introduce the concept of "transition state" which corresponds to the pair of a transition label and a state, where the label corresponds to the last executed transition which led to the state in question.

Definition 2.2
An *execution history* **h** is a sequence of transition states $\mathbf{s_0(\varepsilon),s_1(t_1),s_2(t_2),\ldots}$, where $\mathbf{s_i \in S}$, $\mathbf{t_i \in T}$ and $\mathbf{<s_{i-1},s_i> \in\ -t_i->}$, and $\mathbf{s_0(\varepsilon)}$ is an initial transition state.

Definition 2.3
For a given state $\mathbf{s \in S}$, a transition labelled **t** is said to be *enabled* iff $\exists$**s'** (**s -t -> s'**). For a given transition state **s(t')**, a transition labelled **t** is said to be *enabled*, written as ENABLED(**t**), iff **t** is enabled at **s**.

Definition 2.4
For a given transition state **s(t')** in an execution history **h**, a transition labelled **t** is said to be *executed*, which is denoted as EXECUTED(**t**), iff **t = t'**.

We can combine the two assertions ENABLED and EXECUTED with the two temporal operators [] and ◊, which are defined in Section 2.1. For instance, ◊[] ENABLED(**t**) means transition **t** is (from a certain time onwards) **always** enabled; []◊ ENABLED(**t**) means transition **t** is **infinitely often** enabled; []◊ EXECUTED(**t**) means transition **t** is **infinitely often** executed. For the example of Figure 1, for the execution history **h** = $\mathbf{s_1(\varepsilon),s_2(t_3),s_1(t_3),s_2(t_3),\ldots}$ (that is alternatively $\mathbf{s_1(t_3)}$ and $\mathbf{s_2(t_3)}$), ◊[] ENABLED($\mathbf{t_1}$)= **false**, []◊ENABLED($\mathbf{t_1}$)=**true**, and []◊ EXECUTED($\mathbf{t_3}$)=**true**.

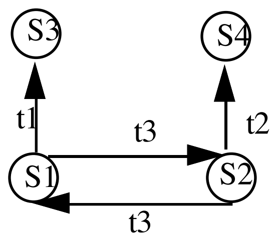


Figure 1: A labeled transition system

Therefore the following definitions are reasonable:

Definition 2.5 (Weak fairness for a labeled transition *t*)
An infinite execution history **h** respects *weak fairness* for a labeled transition **t** if it satisfies
◊[] ENABLED(t) ⇒ []◊ EXECUTED(t)

Definition 2.6 (Strong fairness for a labeled transition *t*)
An infinite execution history **h** respects *strong fairness* for a labeled transition **t** if it satisfies
[]◊ ENABLED(t) ⇒ []◊ EXECUTED(t)

2.3 Serialized models vs overlapping models

In some literature (e.g. [Fran86]), two types of models, namely *a serialized model* and *an overlapping model*, are considered to specify distributed systems. Distributed systems usually involve a collection of processes. Each process can do local actions as well as communicate with other processes in the system (here we consider that the communication mechanism is rendezvous). Each process in turn is in one of the following two kinds of states: a *communication state* where it is waiting to rendezvous with other processes, an *execution state* where it executes a rendezvous or local actions.

In the serialized model, all processes synchronize in a communication state, and then exactly one communication (rendezvous) is selected for execution, advancing the execution to the next state, where again all processes are in a communication state. Thus, only the processes executing the current communication are active, while all the others are idle until the next transition.

In the overlapping model, concurrent execution is captured by not requiring all processes to be synchronized in their respective communication states. A communication can be chosen for execution if the corresponding processes are in their communication states and willing to communicate. If a communication is executed, processes which involve in the communication enter their execution states and they are not available for another communication until they reach their next communication states.

The overlapping model is closer to the nature of distributed systems, while the serialized model is easier to reason about. The two models are different as far as fairness is concerned. Caused by a so-called *conspiracy* phenomenon in the overlapping model, a fair execution history for the overlapping model may not be fair for the serialized model (see

[Fran86]). In the rest of this paper, we only consider the serialized model, because LOTOS semantic is defined on a interleaving (serialized) model [LOTOS87].

3. Grouping labeled transitions

In certain papers (e.g.[Quei83]), fairness properties are related to the idea of "grouping interesting events". That is, properties are described in the following form: "if a group of events, which are of interest, become possible infinitely often, then events of this group will happen infinitely often". In this section, we introduce several concepts which are related to grouping labeled transitions. They will provide later the basis for defining fairness for LOTOS specifications.

3.1. Grouping labeled transitions

The following definitions, theorems and corollaries are always in respect to a labeled transition system **LTS=(S,T,{-t->}** $_{t\in T}$), as defined in the previous section.

Definition 3.1

We call a *transition group* **g** any non-empty sub-set of **T**. We call a *grouping* **G={g1, g2,...}** any set of groups such that $\mathbf{T} = \cup_{\mathbf{g}\in \mathbf{G}}\, \mathbf{g}$. (Note: groups need not be disjoint)

Definition 3.2

A transition group **g** of **T** is said to be enabled at state **s**∈**S**, denoted as ENABLED(**g**), if ∃**t**∈**g** such that ENABLED(**t**).

Definition 3.3

Let **h** be an execution history, a labeled transition group **g** of **T** is said to be executed in **h**, denoted as EXECUTED(**g**), if ∃**t**∈**g** such that EXECUTED(**t**).

Definition 3.4 (g-fairness)

The followings are fairness properties with respect to a transition group **g**

(*weak* ***g****-fairness*) ◊[] ENABLED(**g**) ⇒ []◊ EXECUTED(**g**)

(*strong* ***g****-fairness*) []◊ ENABLED(**g**) ⇒ []◊ EXECUTED(**g**)

Definition 3.5 (G-fairness)

The followings are fairness properties with respect to a grouping **G** of transitions

(*weak* ***G****-fairness*) ∀**g**∈**G** (◊[] ENABLED(**g**) ⇒ []◊ EXECUTED(**g**))

(*strong* ***G****-fairness*) ∀**g**∈**G** ([]◊ ENABLED(**g**) ⇒ []◊ EXECUTED(**g**))

Theorem 3.1
strong **g**-fairness ⇒ weak **g**-fairness
strong **G**-fairness ⇒ weak **G**-fairness

Proof (omitted).

Theorem 3.2
Given a finite number of transition groups $\mathbf{g_i}$ (i = 1,...,n) and $\mathbf{g}=\bigcup_{i=1}^{n}\mathbf{g_i}$. Any execution history **h** which is strongly $\mathbf{g_i}$-fair for all i=1,...,n is also strongly **g**-fair.

Proof (by contradiction)
Suppose there exist an execution history **h** such that
I) strong **g**-fairness is not satisfied, that is, there are infinitely many states **s**∈**h** for which there is **t**∈**g** such that **t** is enabled and there are only a finite numb[redacted] of occurrences of **s(t)** in **h** where **t** is an element of **g**;
II) strong $\mathbf{g_i}$-fairness is satisfied for i=1..n.
From II) we have either
(i) there is a $\mathbf{g_i}$ which is infinitely often enabled. Then there is an infinite number of occurrences **s(t)** in **h** where **t** is an element of $\mathbf{g_i}$. Because $\mathbf{g_i}$ **g**, [redacted]).
or (ii) there is not any $\mathbf{g_i}$ which is infinitely often enabled(for i=1..n). Then **g** is not infinitely often enabled, because $g=\bigcup_{i=1}^{n} g_i$. But this contradicts with I) .

Note the theorem above is not true for weak fairness, which is shown by the example in Figure 1. Let **g**={t1,t2}, $\mathbf{g_1}$={t1}and $\mathbf{g_2}$={t2}, the execution history **h** = $\mathbf{s_1(\varepsilon),s_2(t_3),s_1(t_3),s_2(t_3),...}$ (that is alternatively $\mathbf{s_1(t_3)}$ and $\mathbf{s_2(t_3)}$), respects weak $\mathbf{g_1}$-fairness as well as weak $\mathbf{g_2}$-fairness, but not weak **g**-fairness.

Definition 3.6 (Minimal Liveness)
Weak **T**-fairness is called *Minimal Liveness.*

The Minimal Liveness ensures that as long as there are transitions possible, the system will make a move, that is, execute a transition. In other words, the Minimal Liveness ensures that the system stops only if it enters a terminal or blocked state.

Corollary 3.1 (of Theorem 3.1 and Theorem 3.2)
Let **T** be finite and **G** be any grouping, then
strong **G**-fairness ⇒ Minimal Liveness.

Proof [redacted]
From [redacted]eorem 3.2 we have strong **G**-fairness ⇒ strong **T**-fairness, and from Theorem 3.1 [redacted] have strong **T**-fairness ⇒ Minimal Liveness.

Coroll[redacted]**y 3.2 (of Theorem 3.2)**
Let **T** be finite and let **G'** and **G"** be two groupings such that for all **g"** ∈ **G"** , there exists **G G'**[redacted]$_{G}$ **g**, then
strong **G'**-fairness ⇒ strong **G"**-fairness

3.2. Process fairness

In this section, we define process fairness for a distributed system based on the concepts of Section 3.1. Distributed systems usually involve a collection of processes. Process fairness is informally described as "if a process is always (or infinitely often) enabled, then it is infinitely often executed".

We use a Labeled Transition System to describe a distributed system. A transition is *local* if it is contributed by one process, that is, it specifies a local action of that process; a transition is *joint* if it is contributed by more than one process, that is, it corresponds to communication among several processes (rendezvous). Then we have the following definitions:

Definition 3.7 (Process)
Given a process **P**, we call **p** = {**t** | $\mathbf{f_P(t)}$} the *transition group of process* ***P*** , where $\mathbf{f_P}$**: T-> {true, false}** is the *process assignment function* for process **P** which is defined as

$\mathbf{f_P(t)}$ = **true** if **t** denotes either a local action of process **P** or a communication in which process **P** is involved, or
false otherwise.

Definition 3.8 (Fairness for process P)
1) *Weak fairness for process* ***P*** is weak fairness in respect to the transition group of **P**;
2) *Strong fairness for process* ***P*** is strong fairness in respect to the transition group of **P**.

4. Process, alternative and channel fairness in LOTOS

It is known that different fairness properties can be defined for a given computation model or specification language. For LOTOS, for instance, "gate fairness" can be defined as follows: if a gate is always (infinitely) enabled, it will be eventually executed. But gates in LOTOS do not uniquely determine the process or the alternative performing the action. For example, the "gate fairness" above does not ensure that the LOTOS program **P[a]:= a; P[a] [] a; stop** will eventually terminate when gate **a** is always enabled. So, the question turns out to be : What kind of fairness concepts are suitable for LOTOS?

"Process", "guard" and "channel" are three important concepts in CSP[Hoar78]. Related fairness definitions were given in [Kuip83]. We believe that such concepts are also important for LOTOS. In this paper we show how these concepts can be defined using the concept of transition groups, and how their definition can be extended to the more general conext of LOTOS specifications.

4.1. Process, guard and channel fairness in CSP

Process, guard and channel are three important concepts which are related to the syntax of CSP. The syntax of a subset of CSP used in [Kuip83] is showed as follows, where neither the parallel operator ([...||...]) nor the choice operator (*[...]) is allowed to be used in a nested fashion.

Statements: S::= skip | x:=t | *[b1,c1 -> S1 [] ... [] bm,cm -> Sm] | S1;S2
where t is an integer expression, b is a boolean expression and c is either Pi!x or Pj?y, i,j $\in$ {1, ...n}

Programs: [P1::S1 || ... || Pn::Sn]
where Pi, i$\in${1...n}, is called a process. Processes have no shared variables.

The parallel operator [...||...] defines parallelism, that is, in the expression **[P1::S1 || ... || Pn::Sn]**, **Pi** (i = 1, ..., n) are called *processes* and can be executed in parallel with any **Pj** (j = 1, ..., n and j $\neq$ i), that is, the actions of **Pi** and **Pj** can be executed in any sequence (interleavings). The choice operator *[...] defines *alternatives*, that is, for the expression ***[b1,c1 -> S1 [] ... [] bm,cm -> Sm]**, **bi** and **ci** (i = 1, ..., m) are called *guards* and if **bi** and **ci** are true **Si** is said be enabled and may be executed, and if there are more than one **Si** that are enabled then there is non-determinism. A pair of guards **<g'**, **g">** is called a *channel* if **g'** and **g"** are syntactically matching communication commands (e.g., **Pi!x** in **Pj** and **Pj?y** in **Pi**).

Informally, the related fairness concepts defined in [Kuip83] for CSP can be described as follows:

Weak (Strong) Process Fairness: Any process that becomes permanently (infinitely often) enabled, must execute infinitely often.

Weak (Strong) Guard Fairness: Any alternative, the guards of which become permanently (infinitely often) true, must be chosen infinitely often.

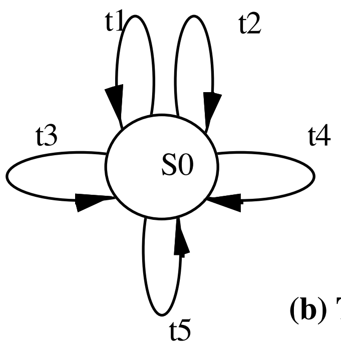
Weak (Strong) Channel Fairness: Any channel that becomes permanently (infinitely often) enabled, must be chosen infinitely often.

If a labeled transition system is given for a CSP program, then the concepts process, guard and channel of CSP defined in [Kuip83] can be viewed as three ways of grouping transitions from the "grouping transition" point of view. More specifically, if we consider the transition system where each transition is labeled by either an action (denoting a local action of a process) or a pair of actions (denoting a rendezvous) (see the example of Figure 2), then we call a "process" the group of transitions containing an action which belongs to a syntactic CSP process; a "guard" the group of transitions containing an action which is in a specific position (of one syntactic CSP process) in the text of the CSP program; and a "channel" the group of transitions whose pair of actions (two syntactically matching actions) are in two specific positions (of two syntactic CSP processes) in the text of the CSP program. Then process, guard and channel *fairness* can be defined based on the framework of Section 3.

For example, the CSP program in Figure 2 defines three processes, six guards and five channels. The execution history $\mathbf{h1} = \mathbf{s_0(\varepsilon),s_0(t_5),s_0(t_5),s_0(t_5),\ldots}$ (that is, **h1** consists of an infinite sequence of $\mathbf{t_5}$ transitions) does not respect process fairness, because $\mathbf{P}_2$ is infinitely often enabled, but is not executed. The execution history $\mathbf{h2} = \mathbf{s_0(\varepsilon),s_0(t_5),s_0(t_1),s_0(t_5),s_0(t_1),\ldots}$ (that is alternatively $\mathbf{t_5}$ and $\mathbf{t_1}$) does respect process fairness, but does not respect guard fairness, because $\mathbf{A}_{P2!2} = \{t3, t4\}$ is infinitely often enabled and is not executed. The execution history $\mathbf{h3} = \mathbf{s_0(\varepsilon),s_0(t_1),s_0(t_3), s_0(t_5),s_0(t_1),s_0(t_3),s_0(t_5),\ldots}$ (that is, repeated sequence of $\mathbf{t_1,t_3,}$ and $\mathbf{t_5}$) does respect guard fairness, but does not respect channel fairness, because $\mathbf{C}_{<P2!1,P1?y>} = \{t2\}$ is infinitely often enabled and is not executed.

```
P:: [P1 || P2 || P3]
where
P1:: *[P2!1 -> a:=1
        []
       P2!2 -> a:=2
        []
      P3!3 -> a:=3 ]
P2:: *[P1?x -> b:=1
        []
       P1?y -> b:=2]
P3:: *[P1?z -> c:=3]
```

(a) CSP specification



t1 = <**P2!1** P1, **P1?x** P2>
t2 = <**P2!1** P1, **P1?y** P2>
t3 = <**P2!2** P1, **P1?x** P2>
t4 = <**P2!2** P1, **P1?y** P2>
t5 = <**P3!3** P1, **P1?z** P3>

(b) Transition system

$\mathbf{P}_{P1} = \{t1, t2, t3, t4, t5\}$
$\mathbf{P}_{P2} = \{t1, t2, t3, t4\}$
$\mathbf{P}_{P3} = \{t5\}$

(c) Processes

$\mathbf{A}_{\langle P1,\, P2!1\rangle} = \{t1, t2\}$
$\mathbf{A}_{\langle P1,\, P2!2\rangle} = \{t3, t4\}$
$\mathbf{A}_{\langle P1,\, P3!3\rangle} = \{t5\}$
$\mathbf{A}_{\langle P2, P1?x\rangle} = \{t1, t3\}$
$\mathbf{A}_{\langle P2, P1?y\rangle} = \{t2, t4\}$
$\mathbf{A}_{\langle P3, P1?z\rangle} = \{t5\}$

(d) Guards

$\mathbf{C}_{\langle P1,\, P2!1\rangle, \langle P2, P1?x\rangle} = \{t1\}$
$\mathbf{C}_{\langle P1,\, P2!1\rangle, \langle P2, P1?y\rangle} = \{t2\}$
$\mathbf{C}_{\langle P1,\, P2!2\rangle, \langle P2, P1?x\rangle} = \{t3\}$
$\mathbf{C}_{\langle P1,\, P2!2\rangle, \langle P2, P1?y\rangle} = \{t4\}$
$\mathbf{C}_{\langle P1,\, P3!3\rangle, \langle P3, P1?y\rangle} = \{t5\}$

(e) Channels

Figure 2: A CSP specification and its transition system

4.2. Process, alternative and channel fairness in LOTOS

In the conext of CSP, as discussed above, the syntactic structure of the program reflects the structure of processes, alternatives and gates, directly. The process structure is static. This is not the case in LOTOS, where process instances may be created dynamically, possibly leading to complex communication relation between these processes, as defined by dynamically identified gates and multi-party rendezvous. In this dynamic setting, it is not so clear what the natural grouping of transitions is, related to the dynamically changing structure of processes.

In the following we propose groupings, which correspond to the CSP groupings described above for the case of static process structure, and which seem also natural in the case of dynamic processes, as discussed below. For process fairness, we introduce a process hierarchy, where each process instance is the "father" of the process instances it creates. The transitions of the "child" processes are considered part of the "father's" transitions, as far as fairness for the "father" process is concerned. Concering alternative fairness, we distinguish not only the syntactic alternatives shown in the program text, but also distinguish between a given alternative being executed by one or the other process instance. This allows a finer kind of alternative fairness, related to a particular process instance. Similar considerations apply for channel fairness. Note that such distinctions are not necessary in the static context of CSP were each alternative belongs to only one process instance.

In LOTOS, the keyword **process** is used to introduce a so-called process definition, that is, a process identifier and a behavior which it represents. In order to create a process instance, the process identifier must be invoked in a behavior expression. An initial process is defined which executes the behavior of the specification. We consider the following example:

```
specification two_processes[a]:noexit:=
    behavior P[a](1) || P[a](2)
    where
    process P[a](x:int) :noexit:=
        a!x; P[a](x:int) [] a?y:int; P[a](x)
    endproc
endspec
```

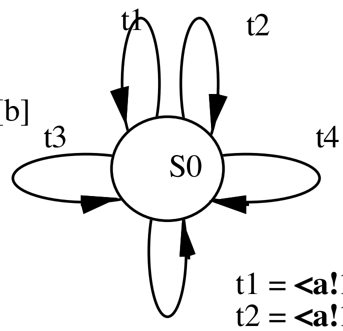
It contains one processes definition and two process instances executing in parallel, namely P[a](1) and P[a](2). The process definition contains two alternatives (a!1) and (a?y:int); we distinguish whether these alternatives are executec by the process instances P[a](1) or P[a](2). The specification contains three channels <a!1, a?y:int>, <a?y:int, a!2>, and <a?y:int, a?y:int> between P[a](1) and P[a](2); the fourth (syntactic) channel <a!1, a!2> does not allow any rendezvous, since the parameter values do not match.

In general, however, these concepts are not very clear because, in contrast to CSP, process instances can be created dynamically in LOTOS, possibly an unlimited number in the case of recursion (for example P:= a; ((b;stop) |||P)). Two process instances, possibly created by different father processes, may refer to the same process definition. For example, in the example above, P[a](1) and P[a](2) are two process instances of P[a](x:int); in the example of Figure 3, Q[a,b] creates P1[a,b], P2[a], and P3[b]. Note process creations are introduced by the parallel operator || and |||. In the following, one process instance is called *super-process* of the other if the former creates directly or indirectly the latter. Analogously, we have the concept of *sub-process*. As mentioned above, we consider that a super-process contains the transitions of its sub-processes. In the following, we introduce the concepts of "process", "alternative" and "channel" at the (dynamic) instance level, instead of at the (static) syntactic level (as done above for CSP). We will use again the idea of "grouping labeled transitions" to define these concepts and the corresponding fairness concepts.

For a given LOTOS specification, we assume that a labeled transition system is given according to the semantic definition of LOTOS, and that each of its transitions is labeled by a list of actions (LOTOS gates) **<a_{p1},a_{p2}, ...,a_{pn}>**(n>0). Transition **<a_{pi}>** denotes a local action of process instance $\mathbf{p_i}$, and $\mathbf{\langle a_{p1},a_{p2}, ...,a_{pn}\rangle}$(n>1) denotes a rendezvous which involves the process instances $\mathbf{p_i}$ (i=1..n) which do the action $\mathbf{a_{pi}}$. For example, Figure 3(b) shows the transition system of the specification of Figure 3(a), which defines the same behavior as the CSP program shown in Figure 2.

```
behavior Q[a,b]
where
process Q[a,b]:=
 P2[a] |[a]| P1[a,b] |[b]| P3[b]
  where
  process P1[a,b]:=
    a!1; P1[a,b]
    []
    a!2; P1[a,b]
    []
    b!3; P1[a,b]
  endproc
 process P2[a]:=
    a?x:int; P2[a]
    []
    a?y:int; P2[a]
  endproc
 process P3[b]:=
    b?z:int; P3[b]
  endproc
endproc
```

(a) LOTOS specification



t1 = **<a!1** P1[a,b], **a?x:int** P2[a]**>**
t2 = **<a!1** P1[a,b], **a?y:int** P2[a]**>**
t3 = **<a!2** P1[a,b], **a?x:int** P2[a]**>**
t4 = **<a!2** P1[a,b], **a?y:int** P2[a]**>**
t5 = **<b!3** P1[a,b], **b?z:int** P3[b]**>**

(b) Transition system

$\mathbf{P}_{Q[a,b]} = \{t1, t2, t3, t4, t5\}$

$\mathbf{P}_{P1[a,b]} = \{t1, t2, t3, t4, t5\}$
$\mathbf{P}_{P2[a]} = \{t1, t2, t3, t4\}$
$\mathbf{P}_{P3[b]} = \{t5\}$

(c) Process transition groups

$\mathbf{A}_{\langle Q[a,b],\ a!1\rangle} = \{t1, t2\}$
$\mathbf{A}_{\langle Q[a,b],\ a!2\rangle} = \{t3,t4\}$
$\mathbf{A}_{\langle Q[a,b],\ b!3\rangle} = \{t5\}$
$\mathbf{A}_{\langle Q[a,b],\ a?x:int\rangle} = \{t1, t3\}$
$\mathbf{A}_{\langle Q[a,b],\ a?y:int\rangle} = \{t2,t4\}$
$\mathbf{A}_{\langle Q[a,b],\ b?z:int\rangle} = \{t5\}$

$\mathbf{A}_{\langle P1[a,b],\ a!1\rangle} = \{t1, t2\}$
$\mathbf{A}_{\langle P1[a,b],\ a!2\rangle} = \{t3,t4\}$
$\mathbf{A}_{\langle P1[a,b],\ b!3\rangle} = \{t5\}$
$\mathbf{A}_{\langle P2[a],\ a?x:int\rangle} = \{t1, t3\}$
$\mathbf{A}_{\langle P2[a],\ a?y:int\rangle} = \{t2,t4\}$
$\mathbf{A}_{\langle P3[b],\ b?z:int\rangle} = \{t5\}$

(d) Alternative transition groups

$\mathbf{C}_{\langle Q[a,b],\ a!1\rangle, \langle Q[a,b],\ a?x:int\rangle} = \{t1\}$
$\mathbf{C}_{\langle Q[a,b],\ a!1\rangle, \langle Q[a,b],\ a?y:int\rangle} = \{t2\}$
$\mathbf{C}_{\langle Q[a,b],\ a!2\rangle, \langle Q[a,b],\ a?x:int\rangle} = \{t3\}$
$\mathbf{C}_{\langle Q[a,b],\ a!2\rangle, \langle Q[a,b],\ a?y:int\rangle} = \{t4\}$
$\mathbf{C}_{\langle Q[a,b],\ b!3\rangle, \langle Q[a,b],\ b?z:int\rangle} = \{t5\}$

$\mathbf{C}_{\langle P1[a,b],\ a!1\rangle, \langle P2[a],\ a?x:int\rangle} = \{t1\}$
$\mathbf{C}_{\langle P1[a,b],\ a!1\rangle, \langle P2[a],\ a?y:int\rangle} = \{t2\}$
$\mathbf{C}_{\langle P1[a,b],\ a!2\rangle, \langle P2[a],\ a?x:int\rangle} = \{t3\}$
$\mathbf{C}_{\langle P1[a,b],\ a!2\rangle, \langle P2[a],\ a?y:int\rangle} = \{t4\}$
$\mathbf{C}_{\langle P1[a,b],\ b!3\rangle, \langle P3[b],\ b?z:int\rangle} = \{t5\}$

(e) Channel transitions groups

Figure 3: A LOTOS specification and its transition system

We define a *process transition group* $\mathbf{P_p}$ to be the group of transitions containing an action which belongs to a specific process instance **p** or its sub-process instances. We define an *alternative transition group* $\mathbf{A_{\langle p,l\rangle}}$ to be the group of transitions containing an action which is defined at a specific position **l** in the text of the LOTOS program and belongs to a specific process instance **p** or its sub-process instances. Finally, we define a *channel* *transition group* $\mathbf{C}_{<\mathbf{p}1,\mathbf{l}1>,<\mathbf{p}2,\mathbf{l}2>,\ldots,<\mathbf{p}n,\mathbf{l}n>}$ (n≥1) to be the group of transitions containing an action which is defined at a specific position $\mathbf{l}_i$ in the text of the LOTOS program and belong to a specific process instance $\mathbf{p}_i$ or its sub-process instances for each i (i=1..n). For example, the specification of Figure 3 defines four process transition groups, twelve alternative transition groups and ten channel transition groups (see Figure 3(c)). In this example, we use the interaction parameters (1 and z:int, etc.) to distinguish the different positions in the text of the specification. As the behavior of this specification is the same as for the CSP specification of Figure 2, it is not surprising that the process, alternative and channel transition groups of Figure 3 include those of Figure 2. However, Figure 3 also contains the groups belonging to the super-process **Q,** corresponding to the initial behavior of the specification.

Now we can defined fairness properties as follows: For a given LOTOS specification, an execution history **h** is said to respect process (alternative, channel) fairness if it respects process (alternative, channel) transition group fairness for each process (alternative, channel) transition group defined for the LOTOS specification.

It is obvious that the above concepts are extensions of the ones for CSP. The sub-set of CSP of Section 4.1 only allows two-way rendezvous and does not allow for process creation.

5. Formalization of LOTOS fairness concepts

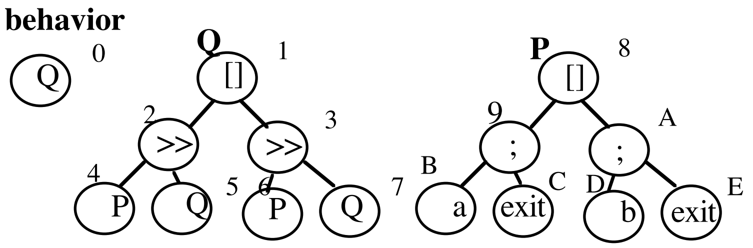
In this section, we will formalize the fairness concepts for LOTOS. For a given LOTOS specification, we first define a (syntactic) index for each action (gate) in the specification. Then, we build a labeled transition system with each of its transitions being labeled by a list of indexed actions. Finally, we characterize the groups of transition which we are interested in, based on action indexes, and define related fairness concepts based on the framework of Section 3.

5.1. LOTOS action indexes

In this section we define an index for each action in the LOTOS specification. For this purpose, we first consider the abstract syntax of the specification which is a set of trees. Each tree represents a process definition in the specification. We call each tree a r*eference tree* and the set of trees the *reference forest*. In a given tree, internal nodes are LOTOS operators and leaf nodes are either actions (LOTOS gates) or **process** names which denote **process** instantiations (the word **process** here is a LOTOS term) (see Figure 4 (a) and (b)). Then we define *reference indexes* for each node of the reference forest. Reference indexes are assigned in such a way that no reference index occurs more than once at different nodes. Actually reference indexes define positions in the text of the LOTOS specification. For example, Figure 4(a) and (b) show reference forests with reference indexes. Let **RI** be the set of reference indexes for a given LOTOS specification. It is obvious that **RI** is finite.

(a)

```
behavior Q[a,b]
where
process Q[a,b]:=
 (P[a,b]>>Q[a,b]) [] (P[a,b]>>Q[a,b])
 where
 process P[a,b]:=
  a; exit [] b; exit
 endproc
endproc
```



(b)

```
behavior Q[a,b]
where
process Q[a,b]:=
 (P[a,b]>>Q[a,b]) ||| (P[a,b]>>Q[a,b])
 where
 process P[a,b]:=
  a; exit [] b; exit
 endproc
endproc
```

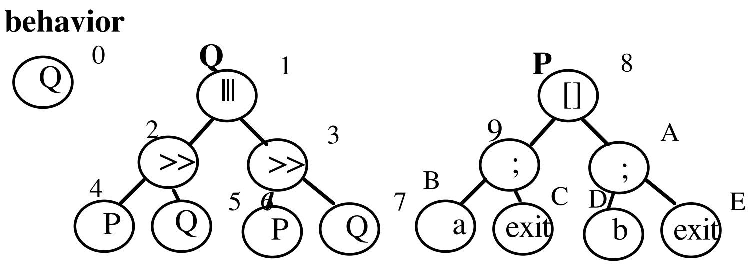


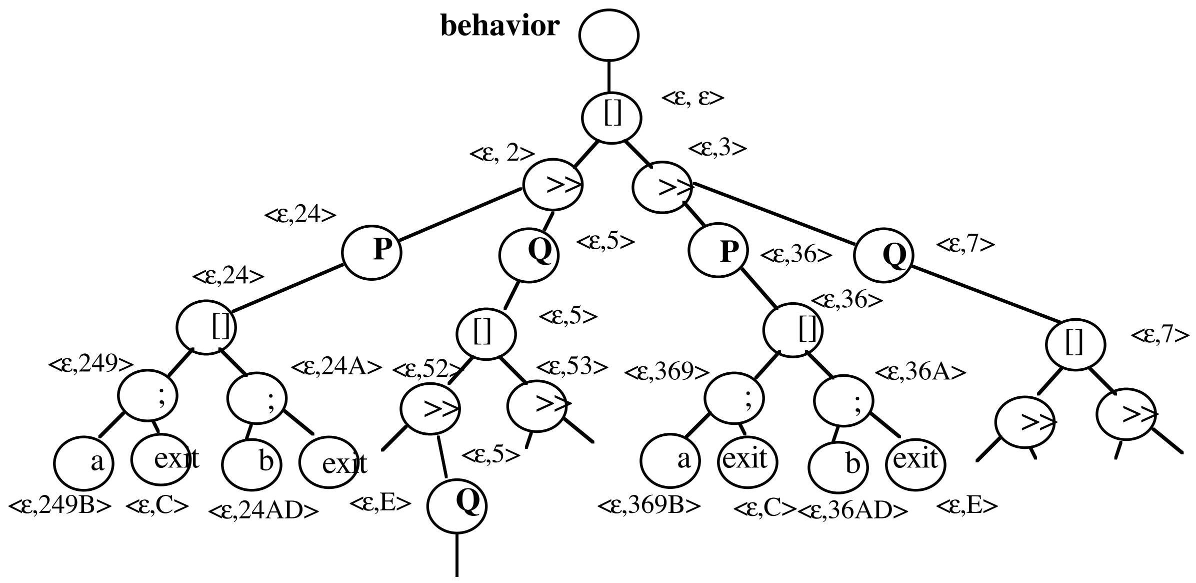
Figure 4: Reference forests and indexes

Secondly we define a *reduction tree* for the given LOTOS specification. A reduction tree is obtained by replacing the process names (leaf nodes) of the reference tree with their (tree form) definitions (see Figure 5(a) and (b)). It is obvious that the reduction tree may be infinite due to recursion. Then we define indexes for the nodes of the reduction tree based on the reference indexes of the related reference forest. The following table shows the rules of building a reduction tree as well as the rules of indexing nodes of the tree, where $\mathbf{l_B}$ denotes the reference index of the related definition of **B** in the reference forest, $\mathbf{l_1 \cdot l_2}$ is the concatenation of $\mathbf{l_1}$ and $\mathbf{l_2}$, and ε denotes the empty string. For example, Figure 5(a) and (b) show reduction trees with indexes. The leaf nodes of the reduction tree represent actions and their indexes are so-called action indexes which will be used in the rest of this paper. An index of an action is a tuple **<p,l>**, where **p** and **l** are strings in **RI***. We call **p** a *process index* (**p**-index), which is a process instance identification to which the action belongs, process instance **p1** is a super-process of process instance **p2**, written **p2≥p1**, if **p1** is prefix of **p2**. We call **l** a *location index* (**l**-index), which records the position of the action in the text of the LOTOS specification (**l** has a similar meaning as the control point of CSP in [Kuip83]).

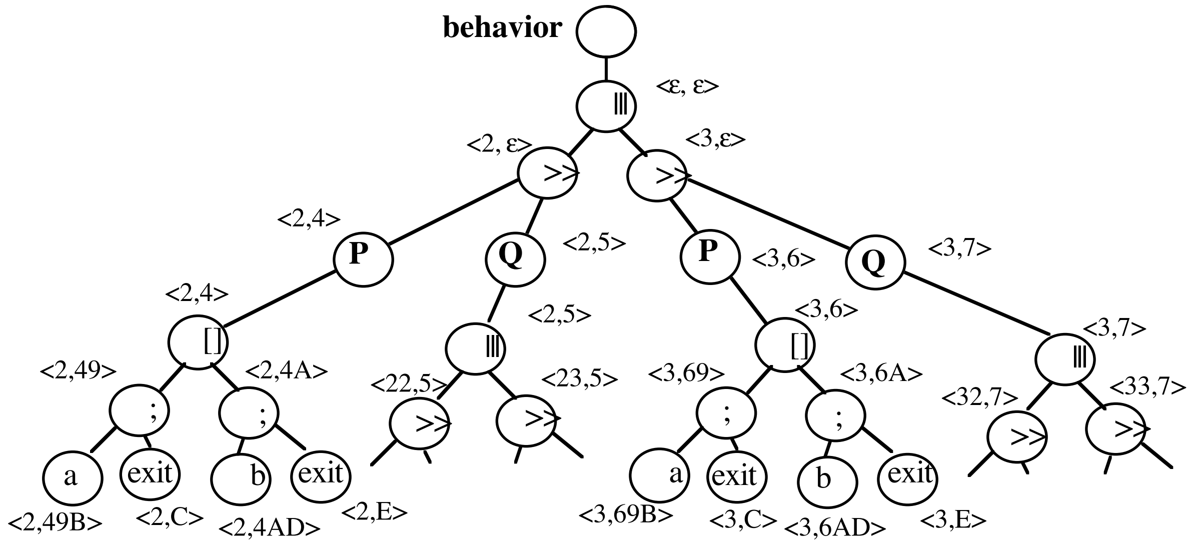
0) **behavior** → B (initial behavior)	B is indexed as <ε,ε>;
1) B → a; B1 (sequential execution)	if B has index **<p,l>** then **a** is indexed as **<p, l·$\mathbf{l_a}$>**, B1 is indexed as **<p, $\mathbf{l_{B1}}$>**;
2) B → B1 >> B2 (sequential execution)	if B has index **<p,l>** then B1 is indexed as **<p, l·$\mathbf{l_{B1}}$>**, B2 is indexed as **<p, $\mathbf{l_{B2}}$>**;
3) B → B1 [] B2 (alternatives)	
B → B1 [> B2 (B1 possibly disrupted by B2)	if B has index **<p,l>** then B1 is indexed as **<p, l·$\mathbf{l_{B1}}$>**, B2 is indexed as **<p, l·$\mathbf{l_{B2}}$>**;
4) B → B1 ll B2 (coupled parallelism)	
B → B1 lll B2 (independent parallelism)	if B has index **<p,l>** then B1 is indexed as **<p·$\mathbf{l_{B1}}$, l>**, B2 is indexed as **<p·$\mathbf{l_{B2}}$, l>**;
5) P → Bp (process invokation)	if P has index **<p,l>**, then Bp is indexed as **<p, l>**

The rational for the above rules are as follows. For the rules concerning sequential execution and alternatives (in a broad sense, namely for the LOTOS operators **;**, **>>**, **[]**, and **[>**) the process index does not change, but the location index records the position corresponding to the LOTOS text. For the operators **ll** and **lll**, two subprocesses are created, identified by a process index which has the index of the father process as prefix. In the case of a process invokation, neither the process index nor the location index are changed, since this construct corresponds to tail recursion and is normally used to define same form of looping behavior (conceptually no new "process").

In the example of Figure 5(b), for instance, the action **a** with index <2,49B> means that **a** belongs to the process instance **2,** as well as to process ε which is the super-process of the former. In the example of Figure 5(a), the index <ε,249B> of action **a** means that it belongs to the alternative **24** in the definition of Q as well as to the alternative **9B** in the definition of P (see Figure 4(a)). The **Q** node indexed by <ε,5> appearing more than once in Figure 5(a) shows a loop in process ε.



(a) partial reduction tree with indexes of the example in Figure 4(a)



(b) partial reduction tree with indexes of the example in Figure 4(b)

Figure 5: Two examples of reduction trees

5.2. Process, alternative, and channel transition groups, and the formalization of LOTOS fairness concepts

We can easily have a labeled transition system for a given LOTOS specification by directly applying the standard inference rules[LOTOS87]. But here we are interested in having a labeled transition systems where each of its transitions is labeled by a list of indexed actions, instead of being labeled by a single LOTOS gate. This can be done by changing the standard inference rules as shown in the following two examples:
The standard LOTOS rules:

1) g; B - g -> B

2) $\dfrac{\text{B1 - g -> B1'and B2 - g -> B2'and } g \in S}{\text{B1 |S| B2 - g -> B1' |S| B2'}}$

are rewritten as:

1) g_s; B - $<g_s>$ -> B

2) $\dfrac{\text{B1 - } <g_{s1},...,g_{sn}> \text{ -> B1'and B2 - } <g_{r1},...,g_{rm}> \text{ -> B2'and } g \in S}{\text{B1 |S| B2 - } <g_{s1},...,g_{sn},g_{r1},...,g_{rm}> \text{ -> B1' |S| B2'}}$

where the $\mathbf{s_i}$ and $\mathbf{r_i}$ denote the indexes defined in the previous section.

Now we group transitions by using the action indexes and give related fairness definitions. The following definitions are in respect to a labeled transition system with a set of transitions **T** whose elements are lists of indexed actions. In the following definitions, **t** denotes an element of **T.**

Definition 5.1 (Process)

Given a process **P**, we call the group $\{\mathbf{t} \mid \mathbf{f_P}(\mathbf{t})\}$ the *process transition group* of process **P** where $\mathbf{f_P}$**: T -> {true, false}** is the *process assignment function* for process **P** which is defined as

$\mathbf{f_p}(<\mathbf{a}_{<p1,\ l1>} ...,\mathbf{a}_{<pn,\ ln>}>)$ = **true** if there is a $\mathbf{a}_{<pi,li>}$such that $\mathbf{p_i} \geq \mathbf{p}$, or
false otherwise

Definition 5.2 (Alternative)

Given an alternative **<p,l>** (the alternative **l** in process **p**), we call the group $\{\mathbf{t} \mid \mathbf{f}_{<p,l>}(\mathbf{t})\}$ the *alternative transition group* of alternative **<p,l>** where $\mathbf{f}_{<p,l>}$**: T -> {true, false}** is the *alternative assignment function* for alternative **<p,l>** which is defined as

$\mathbf{f}_{<p,l>}(<\mathbf{a}_{<p1,\ l1>} ...,\mathbf{a}_{<pn,\ ln>}>)$ = **true** if there is a $\mathbf{a}_{<pi,li>}$such that $\mathbf{p_i} \geq \mathbf{p}$ and $\mathbf{l_i}=\mathbf{l}$, or

false otherwise

Definition 5.3 (Channel)
Given a channel **<p1,l1>,...,<pm,lm>**, we call the group $\{$**t** | $\mathbf{f}_{<p1,l1>,\ldots,<pm,lm>}$ (**t**)$\}$ the *channel transition group* of channel **<p1,l1>,...,<pm,lm>** where $\mathbf{f}_{<p1,l1>,\ldots,<pm,lm>}$**: T -> {true, false}** is the *channel assignment function* for channel **<p1,l1>,...,<pm,lm>** which is defined as

$\mathbf{f}_{<p1,l1>,\ldots,<pm,lm>}(<\mathbf{a}_{<p'1,\ l'1>} \ldots, \mathbf{a}_{<p'n,\ l'n>}>)$

= **true** if $n \geq m$ and for all $<\mathbf{p_i},\mathbf{l_i}>$ there is a $\mathbf{a}_{<p'j,l'j>}$ such that $\mathbf{p'_j} \geq \mathbf{p_i}$ and $\mathbf{l'_j} = \mathbf{l_i}$,or
false otherwise

Definition 5.4 (Process fairness)
*Weak (strong) fairness for process **P*** is weak (strong) fairness in respect to the transition group of process **P**. *Weak (strong) process fairness* is weak (strong) fairness in respect to the transition groups of all process instances defined by the LOTOS specification.

Definition 5.5 (Alternative fairness)
*Weak (strong) fairness for alternative **<p,l>*** is weak (strong) fairness in respect to the transition group of alternative **<p,l>**. *Weak (strong) alternative fairness* is weak (strong) fairness in respect to the transition groups of all alternatives defined by the LOTOS specification.

Definition 5.6 (Channel fairness)
*Weak (strong) fairness for channel **<p1,l1>,...,<pm,lm>*** is weak (strong) fairness in respect to the transition group of **channel <p1,l1>,...,<pm,lm>**. *Weak (strong) channel fairness* is weak (strong) fairness in respect to the transition groups of all channels defined by the LOTOS specification.

Corollary
From Theorem 3.1, we have the following:
Strong process (alternative, channel) fairness implies weak process (alternative, channel) fairness

From the above definitions, it is obvious that for each process transition group **pg** there exist a set of alternative transition groups **AG'** such that $\mathbf{pg} = \cup_{\mathbf{ag} \in \mathbf{AG'}} \mathbf{ag}$ and for each alternative transition group **ag** there exist a set of channel transition groups **CG'** such that $\mathbf{ag} = \cup_{\mathbf{cg} \in \mathbf{CG'}} \mathbf{cg}$. Therefore, if **T** is finite, from Corollary 3.2, we have the following:

Strong channel fairness implies strong alternative fairness, and the latter implies strong process fairness.

6. Proving liveness

As discussed in Section 1, for concurrent and non-deterministic computation models, such as LOTOS, fairness properties (sometimes called fairness assumptions) are important for proving liveness properties. We will show, in this section, how to prove liveness properties for LOTOS specifications based on our fairness assumptions.

To build a proving system, one first needs to formalize the underlying computation model and its related fairness assumptions in a common formalism. Then the interesting liveness properties can be described in the same formalism. Finally, one uses the axioms and inference rules of the formalism to prove, manually or mechanically, the liveness properties of the specified system.

Temporal logic provides various axioms and inference rules [Pnue77][Owic82]. Together with certain fairness assumptions, which are defined in terms of temporal logic, this formalism can be used to prove liveness properties for concurrent and non-deterministic computation models [Kuip83][Fant89]][Owic82]. LOTOS semantics can be defined in terms of labeled transition systems [LOTOS87] as well as temporal logic [Fant89][Fant90]. We believe that the fairness formalisms given in Sections 3 and 5, given in the framework of temporal logic, can be used as a basis for building a proof system for LOTOS specifications.

As a very simple example, we consider the specifications of Figure 6 which represent two vending machines: **VM1** and **VM2**. **VM1** consists of two sub-machines, a **good-machine** and a **bad-machine**. After dropping a coin into **VM1**, little Tom may obtain a candy, if (and only if) his coin is accepted by the **good-machine** of **VM1**. **VM2** is obtained from **VM1** by replacing the **good-machine** by a machine, called **better-machine**, which has a storing function. After getting a coin, the **better-machine** may store the coin or may dispense n candies corresponding to the n-1 coins accepted previously by the **better-machine**. We consider the following liveness property of the **VM**'s: Little Tom will eventually get some candy if he insists putting coins into the vending machine.

The liveness property holds for **VM1** if it satisfies strong process fairness: by always putting coins into VM1, the process **good-machine** is infinitely often enabled, thus it will eventually execute, that is, little Tom will eventually get a candy. However, the process fairness assumption does not ensure the liveness property for **VM2** , because the **better-machine** may always store coins. However, little Tom will eventually get candies if **VM2** satisfies in addition the alternative fairness assumption.

```
specification VM1[coin, candy]:noexit:=
    behavior
        good-machine[coin, candy] ||| bad-machine[coin, candy]
    where
    process good-machine[coin, candy]:noexit:=
        coin; candy; good-machine[coin, candy]
    endproc
    process bad-machine[coin, candy]:noexit:=
        coin; bad-machine[coin, candy]
    endproc
endspec

specification VM2[coin, candy]:noexit:=
    behavior
        better-machine[coin, candy](1) ||| bad-machine[coin, candy]
    where
    process better-machine[coin, candy](n:integer):noexit:=
        coin; (candy!n; better-machine[coin, candy](1)
            []
            better-machine[coin, candy](n+1))
    endproc
    process bad-machine[coin, candy]:noexit:=
        coin; bad-machine[coin, candy]
    endproc
endspec
```

Figure 6: Two vending machine systems

Concerning the dynamic creation of processes, we consider another specification in Figure 7 which represents a **Game** system. The **Game** system allows children login one by one. For each logged-in child, the system creates a **Server** to allow the child to make one play and then logout. The gate **stop_game** of the system is for parents to stop the game. The interesting liveness property is: The game will eventually stop (the termination property) if parents insist trying the interaction **stop_game**. There may be infinite number of children that want to login the game (a child who just finished a previous game may try to login again). Therefore, the liveness property can not be ensured unless the Game satisfies weak process and alternative fairness. The process fairness ensures that a logged-in child will be served by a **Server** process and will eventually logout. The alternative fairness ensures that the signal of **stop_game** will eventually be responded by the **Game**. Thus the whole system will eventually terminate.

```
Specification Game[login, logout, play, stop_game]:exit:=
    behavior Game[login, logout, play, stop_game]
    where
    process Game[login, logout, play, stop_game]:exit:=
        login;
        (
            Server[play, logout]:exit:=
            |||
            (Game[login, logout, play, stop_game] [] (stop_game;exit))
        )
    endproc
    process Server[play, logout]:exit:=
        play; logout; exit
    endproc
endspec
```

Figure 7: A game system

Besides these simple examples, fairness assumptions are also useful for proving liveness properties for more complex LOTOS specifications. For instance in the case of the OSI Transport Service specification, communication is provided over a number of parallel connections. Over each connection, data transfer is possible independently in both directions. The following liveness property is of interest: After a connection is established, user data will be eventually transmitted if the sender and receiver are always ready. The LOTOS specification of [OSI87] foresees at each connection endpoint two processes, called TCEPHalf, which deal with the two directions of data transfer respectively. Therefore, weak process fairness will ensure the above liveness property, since the related TCEPHalf process will always be ready and therefore be eventually executed. However, in the case of the Transport protocol specification of [Boch90a], which implements the above Transport service, process fairness is not sufficient to assure the liveness assertion. This specification contains a process, called AP_open, which handles the data transfer over a given connection and includes two alternatives for dealing with transfer in the two respective directions. In this case, process fairness assures the progress of data transfer over different connections, independently from one another, and alternative fairness is required to assure the independence of data transfer in both direction over a given connection.

7. On the fair execution of LOTOS specifications

In the previous sections, we introduced process, alternative and channel fairness for LOTOS. In this section, we discuss the construction of a 'fair execution model'. A fair execution model is one which will only produce finite or infinite execution sequences which are allowed by the specification and satisfy certain fairness assumptions (see Sections 3 and 5). Such a fair execution model is therefore a means for executing LOTOS specifications that satisfy those liveness properties that can be proved based on the underlying fairness assumptions, as discussed in the section above.

LOTOS is designed as an executable specification language. Several LOTOS interpreters are described in the literatures [BRIA86][Logr 88]. Usually they work in two modes: *user guided execution* and *system guided execution*. In user guided execution, it is the user who chooses an action to be executed when there is non-determinism (more than one action enabled). In system guided execution, it is the system which chooses automatically an action to be executed. Here, we are interested in fair strategies for choosing actions in the system guided execution mode.

In the following we will discuss the fair execution of LOTOS specifications in a centralized environment, based on the execution model described in [WuBo90].

7.1. An execution model for LOTOS

A model for executing LOTOS specifications is suggested in [WuBo90]. It is based on a so-called activity tree. The activity tree reflects the system state. By growing and evaluating the activity tree at run time, the system can find enabled actions in a given state. After choosing one of the enabled actions to execute, the system changes its state by updating the activity tree according to the LOTOS semantics (updating rules in [WuBo90]), and grows and evaluates the activity tree again.

The activity tree is a partially developed reduction tree as described in Section 5.1. Some leaf nodes of the activity tree represent finite or infinite sub-trees of the reduction tree. These nodes denote behaviors in which the system is presently not interested. For example, behaviors which are not active in the present state are represented by non-expanded nodes instead of by the sub-tree in the reduction tree; also loops for non-well-guarded expression not fully expanded. In addition, those behaviors which are not possible any more after a given rendezvous happened do not appear in the tree.

The activity tree consists of leaf nodes and internal nodes. An internal node of the activity tree represents the relation between its descendent nodes, like in the reduction tree, i.e. one of the LOTOS operators [], ||, |||, and [> etc. , or contains the description of the behavior to be activated after the successful termination of its descendants, i.e. >>B (where B is a behavior expression). There are two kinds of leaf nodes: terminal and non-terminal. A terminal node corresponds to a behavior expression 'g;B', where 'g' is called an active action and 'B' is the behavior expression which will be activated after a rendezvous happens at gate 'g'. A non-terminal node cannot directly participate in an interaction, it must first be expanded. A non-terminal node corresponds to a behavior expression 'B1#B2', where # is one of the operators ||, |||, [], [> and >>, and 'B1' and 'B2' are behavior expressions. During the execution of a LOTOS specification, the activity tree is grown and updated. During growing, the system expands non-terminal nodes in order to find terminal nodes with possible actions. During updating, the system prunes those sub-trees of the activity tree which represent alternative behaviors not possible any more after the choice of the last rendezvous.

One of the characteristics of the execution model is that it can commit enabled actions at an early stage. As soon as a rendezvous has been determined to be possible, it may be executed without all alternatives of other actions being explored. This is done by growing the activity tree step by step and choosing an enabled action to execute as soon as one is found. By doing so, the execution model can efficiently execute LOTOS specifications and deal with non-well-guarded expressions (see [WuBo90]).

It is noted, however, that the 'growing' concept in the execution model may implicitly introduce priorities among the enabled actions. A given growing strategy may allow some enabled gate to have higher priority than others. For example, the breadth-first growing strategy described in [WuBo90] always commits gates near the root in the activity tree prior to the ones further down. This may lead to unfair execution sequences.

7.2. Fair execution: general ideas

A particular fair strategy is **random choice**: whenever the system has several choices, it randomly selects one of them. The choices to be considered in the LOTOS execution model above are (1) selection among several possible rendezvous, and (2) the choice between execution of a rendezvous and further expansion of the activity tree (up to a limited depth or until another rendezvous is found). Such a random strategy leads to fair execution sequences with probability one. However, in the following, we are interested in strategies which do not depend on probability for executing LOTOS specifications in a fair manner.

Many methods are know to ensure fair scheduling of processes in a multi-user computer operating system. One of these methods uses counters to count the execution time for the executing process and the waiting time for waiting processes. When the execution time reaches a pre-defined threshold, the executing process will be disabled and the system chooses a new process for execution among the waiting processes; the process with the largest waiting time will be chosen. We can also use counters for fairly executing LOTOS specifications. The general idea is to count the execution time related to process, alternative and channel transition groups, as defined in Section 5.

7.3. A growing strategy with strong process fairness

To ensure that the execution model produces only execution sequences which satisfy the Strong Process Fairness defined above, the system uses a counter for each process transition group. Whenever a rendezvous is executed related to a given process transition group, its counter is increased by one. The system indexes the activity tree by the rules defined in Section 5.1. Based on the index of a given non-terminal node of the activity tree, the system can tell to which process transition group it belongs. During the growing phase, the system first calculates for each non-terminal node the sum of its depth from the root and the counter of the process transition group to which it belongs. Then the system expands non-terminal nodes which have the smallest values. The growing phase stops when a rendezvous is found and there are no non-terminal nodes that have a smaller value than the rendezvous. Then the rendezvous which has the smallest value will be executed. Note that in the pure breadth-first growing strategy [WuBo90], which is not necessarily fair, only the 'depth' is considered.

It is clear that the growing strategy above respects strong process fairness. The more a process transition group is executed, the bigger its counter becomes, and the more likely non-terminal nodes related to other process transition groups will be expanded in the growing phase. Thus a process transition group will be eventually executed if it is infinitely often enabled.

We can use the same idea to implement other fairness properties, such as alternative and channel fairness. In those cases, counters relate to alternative and channel transition groups, instead of process transition groups.

8. Conclusions

Fairness has been being studied for in relation with various specification languages. In this paper, we defined process fairness, alternative fairness and channel fairness for LOTOS. these concept are extension of those defined for CSP [Kuip83].

Certain features of LOTOS, such as the dynamic creation of processes, the dynamic relation between gates and processes, and related membership in multi-way rendezvous, not present in CSP, make the definition of fairness difficult. However, by introducing the concept of "transitions groups", we generalize the problem of defining fairness. The introduction of the concept of "action indexes" allows us to define the concepts of processes, alternatives and channels for LOTOS. We use temporal logic not only for clearly defining fairness properties, but also for proving liveness properties of LOTOS specifications, based on underlying farness assumptions. We have discussed a simple example, and we believe that our fairness assumptions are also useful for proving liveness properties for LOTOS specifications for complex systems, such as communication protocols and services.

We have also considered the construction of fair execution models for LOTOS specifications. The approach of Section 7.3 works in a centralized environment. However, it turns out to be much more difficult to ensure fairness in a distributed execution environment [Atti90]. This area needs further study.

Acknowledgements:

This work was partly supported by the Natural Sciences and Engineering Research Council of Canada, and the Ministry of Education of Quebec.

References

[Atti90] Paul C. Attie, Ira R. Forman, Eliezer Levy, "On Fairness as an abstraction for the design of distributed systems", 1990.

[Boch89] Gregor v. Bochmann, Qiang Gao and Cheng Wu, "On the Way of Distributed Implementation of LOTOS specifications", FORTE'89, Vancouver, Dec. 1989.

[Boch90a] G. v. Bochmann, "Specifications of a simplified Transport protocol using different formal description techniques", Computer Networks and ISDN Systems, Vol. 18, no.5 (June 1990), pp. 335-377.

[Bolo87] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS", Computer Network and ISDN Systems, vol. 14, no. 1, pp. 3- , 1987.

[Bria86] J.P. Briand, M.C. Fehri, L. Logrippo, A. Obaid, "Executing LOTOS Specifications", in Protocol Specification, testing and verification, B. Sarikaya and G. v. Bochmann (eds), North Holland, 1986.

[Cost84] Costa, G., Stirling, C., "A fair calculus of communicating systems", Acta Informatica 21, pp. 417 - 441, Springer - Verlag, 1984.

[Cost84a] Costa, G., Stirling, C., "Weak and strong fairness in CCS", Procs. of Symposium on Mathematical Foundations of Computer Science, Prague, LNCS 176, pp. 245 - 254, 1984.

[Fant89] A. Fantechi, S.Gnesi, C. Laneve, "An expressive temporal logic for basic LOTOS", FORTE'89, Vancouver, Canada, 1989.

[FantT90] A. Fantechi, S. Gnesi, G. Ristori, "Compositional logic semantics and LOTOS", Tenth International IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verificaton, Ottawa, Canada, June, 1990.

[Fran86] Nissim Francez, "Fairness", Springer-Verlag New York, 1986.

[Hoar78] C. A. R. Hoare, "Communication Sequential Processes", Comm ACM 21(8), pp. 666 - 677, 1978.

[ISO78] ISO TC97/6/WG4/N317, "Formal Description of ISO8072 in LOTOS", 1987.

[Kuip83] Kuiper R, de Roever WP, "Fairness assumptions for CSP in a temporal logic framework", In Bjorner D (ed) Proceeding of TC.2 Working

Conference on the Formal Description of Programming Concepts, Garmisch Partenkirchen, North Holland, 1983.

[Logr 88] L. Logrippo and e. al., "An interpreter for LOTOS: A specification language for distributed systems", Software Practice and Experience, Vol. 18(4), pp.365-385, April 1988.

[LOTOS87] "LOTOS - A formal description technique based on the temporal ordering of observational behavior", ISO, DIS 8807, 1987.

[Miln80] R. Milner, "A Calculus of Communicating Systems", Lecture Notes in CS, No. 92, Springer Verlag, 1980.

[Owic82] Owicki, S., "Proving Liveness Properties of Concurrent Programs", ACM Transactions on Programming Languages and Systems, vl. 4, no.3, July 1982, pp. 455-495.

[Parr85] J. Parrow, "Fairness properties in process algebra", Ph.D. thesis, Dept. of Computer Systems, Uppsala University, Uppsala, Sweden, 1985.

[Pneu77] Pnueli, A. "The temporal logic of programs", in Proceedings of the 18th Symposium on the Foundations of Computer Science, IEEE, Providence, Nov. 1977, pp. 46-57.

[Pneu79] Pnueli, A. "The temporal semantics of concurrent programs", in Lecture Notes in Computer Science, vol. 70: Semantics of Concurrent Computation. Springer-Verlag, New York, 1979, pp. 1-20.

[Quei83] J.P. Queille and J. Sifakis, "Fairness and related properties in Transition Systems - a temporal logic to deal with fairnes", Acta Informatioa 19, pp. 195 - 220, Springer - Verlag, 1983.

[WuBo90] Cheng Wu and Gregor v. Bochmann, "An execution model for LOTOS specifications", GLOBCOMM'90, San Diego, Dec., 1990.