Using Fairness Constraints in Process-Algebraic Verification

Antti Puhakka

Tampere University of Technology, Institute of Software Systems,
P.O. Box 553, FIN-33101 Tampere, Finland,
email: antti.puhakka@tut.fi


The copyright of this text belongs to Springer-Verlag. The homepage of the LNCS series is http://www.springer.de/comp/lncs/index.html.
Using Fairness Constraints in Process-Algebraic Verification

Antti Puhakka

Tampere University of Technology, Institute of Software Systems, P.O. Box 553, FIN-33101 Tampere, Finland,
email: antti.puhakka@tut.fi

Abstract. Although liveness and fairness have been used for a long time in classical model checking, with process-algebraic methods they have seen far less use. One problem is that it is difficult to combine fairness constraints with the compositionality of process algebra. Here we show how a class of fairness constraints can be applied in a consistent way to processes in the compositional setting. We use only ordinary, but possibly infinite, LTSs as our models of processes. In many cases the infinite LTSs are part of a larger system, which can again be represented as a finite LTS. We show how this finiteness can be recovered, namely, we present an algorithm that checks whether a finite representation exists and, if it does, constructs a finite LTS that is equivalent to the infinite system. Even in the negative case, the system produced by the algorithm is a conservative estimate of the infinite system. Such a finite representation can be placed as a component in further compositional analysis just like any other LTS.

1 Introduction

In the verification of concurrent systems it is often important to show that the system eventually performs some desired task. Such properties are called liveness properties [2]. For proving liveness properties some fairness assumptions [3, 9, 14] often have to be added to the system, meaning that the system is not allowed to continually favour some choices at the expense of others.

Within classical model checking [6, 22] liveness and fairness have been used in one form or another for quite some time. However, in the context of process-algebraic methods such as CCS [16] and CSP [12, 23] they have seen relatively little use. As shown in [21], one reason for this is that fairness properties are tricky to combine with the compositionality of process algebra, because outside processes can interfere with the actions used in the fairness constraint. As also discussed in [21], another problem is that most process-algebraic semantics do not preserve enough fairness-related information about the behaviour of systems.

In this article we use a variant of CSP called CFFD (Chaos-Free Failures Divergences), which is especially well suited for handling liveness properties, because it preserves both divergences (livelocks) and the behaviour after executing
a divergence trace. Also, it allows the use of infinite and infinitely branching processes [13, 27, 26].

Some previous approaches have aimed at using a “global” fairness assumption in a process-algebraic system, meaning that all processes or enabled actions should eventually proceed. This has been done either by changing the operational semantics, as in [8], or by considering only the fair executions in the semantics, as in [5, 11, 17]. Others have suggested adding some extra constraints to the process to restrict the infinite executions, such as ω-regular expressions [18] or Büchi states [7]. However, a potential problem in such an approach is that a process may be unable to fulfill the requirements of the additional constraint, which creates a situation with no meaningful interpretation. Furthermore, this can happen as the result of the parallel composition of “healthy” subprocesses, as discussed in [21]. These and a number of other references are discussed in more detail in [20].

In this paper we use only ordinary LTSs (Labelled Transition Systems), so processes will always have an unambiguous behaviour, and we can clearly define how fairness constraints should change the behaviour of systems. The fairness requirements are expressed in linear temporal logic, and we allow the resulting LTSs to be infinite, if necessary. However, we will see that such infinite processes are often part of a larger system which can again be represented as a finite LTS. We will describe an algorithm for checking whether a finite representation exists and, if so, for constructing a finite LTS that is equivalent to the original system. The result can then be used as a component in further compositional analysis. In fact, even when no exact finite representation of the fair system exists, the finite model is a conservative estimate of the fair system. However, it turns out that the complexity of deciding whether an exact finite representation exists is higher than that of building the representation. Fortunately, the parameter system of the analysis can be significantly smaller than the full state-space of the system.

This paper extends the earlier work in [21] by using a significantly more general class of fairness constraints, by presenting the above-mentioned algorithm for constructing a finite model of the fair system, and also by extending the set of systems to which such constraints can be applied.

The paper is organised as follows. In the next section we will review the basic definitions concerning LTSs, process operators and behavioural equivalences. In Section 3 we will consider fairness operators that add fairness constraints to systems, and we state the requirements we believe these operators should fulfill in order to be meaningful in process-algebraic verification. Then, in Section 4 we present the class of fairness properties that we will use. These are properties of the form “if something happens infinitely many times, then something else also has to happen infinitely many times”. In Section 5 we define a fairness operator that implements the fairness constraints by placing the target system in parallel with a “fairness LTS”. We then show that the operator fulfills all the stated requirements. In Section 6 we present an algorithm for constructing a finite representation of the fair system. In Section 7 we present some examples using the approach, and in Section 8 we present our conclusions.
2 Background

The behaviour of a process consists of executing actions. There are two kinds of actions: visible and invisible. Invisible actions are denoted with a special symbol $\tau$. The behaviour of a process is represented as a labelled transition system. This is a directed graph whose edges are labelled with action names, with one state distinguished as the initial state.

Definition 1. A labelled transition system, abbreviated LTS, is a four-tuple $(S, \Sigma, \Delta, \delta)$, where

- $S$ is the set of states,
- $\Sigma$, the alphabet, is the set of the visible actions of the process; we assume that $\tau \notin \Sigma$,
- $\Delta \subseteq S \times (\Sigma \cup \{\tau\}) \times S$ is the set of transitions, and
- $\delta \in S$ is the initial state.

We also use $\Sigma_L$ to denote the alphabet of $L$, and similarly with $S_L$, $\Delta_L$ and $\delta_L$.

Let $A^*$ denote the set of finite and $A^\omega$ the set of infinite sequences of elements of a set $A$. The empty sequence is denoted with $\varepsilon$, and $a^\omega$ denotes the infinite sequence of the symbol $a$. For a finite or infinite sequence $\rho$, the restriction of $\rho$ to $B$, denoted $\text{restr}(\rho, B)$, means the result of removing all actions from $\rho$ that are not in $B$.

The following notation is useful for talking about the execution of a process. The $\rho \Rightarrow$ notation requires that all actions along the execution path are listed, while the $\tau$-actions are skipped in the $\rho \Rightarrow$ notation.

Definition 2. Let $(S, \Sigma, \Delta, \delta)$ be an LTS, let $s, s' \in S$, $a, a_1, a_2, \ldots \in \Sigma \cup \{\tau\}$. We write

- $s \xrightarrow{a} s'$ if and only if $(s, a, s') \in \Delta$,
- $s \xrightarrow{a_1, a_2, \ldots, a_n} s'$ if and only if there are $s_0, s_1, \ldots, s_n \in S$ such that $s = s_0$, $s_n = s'$, and $s_{i-1} \xrightarrow{a_i} s_i$ when $1 \leq i \leq n$,
- $s \xrightarrow{a_1, a_2, \ldots} s'$ is defined similarly for an infinite execution,
- $s \xrightarrow{a_1, a_2, \ldots, a_n} \text{ if and only if there is } s' \in S \text{ such that } s \xrightarrow{a_1, a_2, \ldots, a_n} s'$.

We also write $s \Rightarrow_\sigma s'$ for $\sigma \in \Sigma^*$ if and only if there is $\rho \in (\Sigma \cup \{\tau\})^*$ such that $s \xrightarrow{\rho} s'$ and $\text{restr}(\rho, \Sigma) = \sigma$, and similarly for $s \Rightarrow_\eta$, where $\eta \in \Sigma^* \cup \Sigma^\omega$.

We need the following semantic sets extracted from an LTS. A trace of an LTS is the sequence of visible actions generated by any finite execution that starts in the initial state. An infinite execution that starts in the initial state generates either an infinite trace or a divergence trace, depending on whether the number of visible actions in the execution is infinite or finite. The stable failures describe the ability of the LTS to refuse actions after executing a particular trace.

Definition 3. Let $L = (S, \Sigma, \Delta, \delta)$ be an LTS.

- $\text{Tr}(L) = \{ \sigma \in \Sigma^* \mid \delta \Rightarrow_\sigma \}$ is the set of traces of $L$. 
\[- \text{Inftr}(L) = \{ \xi \in \Sigma^\omega \mid \hat{s} = \xi \Rightarrow \} \text{ is the set of infinite traces of } L.\]
\[- \text{Divtr}(L) = \{ \sigma \in \Sigma^* \mid \exists s \in S : \hat{s} = \sigma \Rightarrow s \wedge \tau^\omega \Rightarrow \} \text{ is the set of divergence traces of } L.\]
\[- \text{Sfail}(L) = \{ (\sigma, A) \in \Sigma^* \times 2^\Sigma \mid \exists s \in S : \hat{s} = \sigma \Rightarrow s \wedge \forall a \in A \cup \{ \tau \} : \neg (s \wedge a \Rightarrow) \} \text{ is the set of stable failures of } L.\]

The parallel composition operator defined below forces precisely those component processes to participate in the execution of a visible action that have the action in their alphabets. The invisible action is always executed by exactly one component process at a time. We first define the product of LTSs as the LTS that satisfies the above description and has as its set of states the Cartesian product of the component state sets. We then define parallel composition by picking the part of the product that is reachable from the initial state of the product.

**Definition 4.** Let \( L_1 = (S_1, \Sigma_1, \Delta_1, \hat{s}_1), \ldots, L_n = (S_n, \Sigma_n, \Delta_n, \hat{s}_n) \) be LTSs. Their product is the LTS \((S', \Sigma, \Delta', \hat{s})\) such that the following hold:

\[- S' = S_1 \times \cdots \times S_n \]
\[- \Sigma = \Sigma_1 \cup \cdots \cup \Sigma_n \]
\[- ((s_1, \ldots, s_n), a, (s'_1, \ldots, s'_n)) \in \Delta' \text{ if and only if either} \]
\[- \bullet a = \tau, \text{ and } (s_i, a, s'_i) \in \Delta_i \text{ for some } 1 \leq i \leq n, \]
\[- \text{ and } s_j = s'_j \text{ for all } 1 \leq j \leq n, j \neq i \]
\[- \bullet a \in \Sigma_i, \text{ and for each } 1 \leq i \leq n \text{ either } a \in \Sigma_i \text{ and } (s_i, a, s'_i) \in \Delta_i, \text{ or} \]
\[- a \not\in \Sigma_i \text{ and } s_i = s'_i \]
\[- \hat{s} = (\hat{s}_1, \ldots, \hat{s}_n) \]

The parallel composition \( L_1 \parallel \cdots \parallel L_2 \) is the LTS \((S, \Sigma, \Delta, \hat{s})\) such that

\[- S = \{ s \in S' \mid \exists \sigma \in \Sigma^* : \hat{s} = \sigma \Rightarrow s \} \]
\[- \Delta = \Delta' \cap (S \times (\Sigma \cup \{ \tau \}) \times S) \]

It is straightforward to show that “” is symmetric and associative, so that \( L_1 \parallel L_2 \cong L_2 \parallel L_1 \parallel L_3 \cong L_3 \parallel (L_1 \parallel L_2) \parallel L_3 \), where “” denotes isomorphism. Therefore, if we wish, we can discard the parentheses and write \( L_1 \parallel L_2 \parallel L_3 \), and similarly with any greater number of processes.

The hiding operator converts given visible actions into \( \tau \)-actions and removes them from the alphabet.

**Definition 5.** Let \( L = (S, \Sigma, \Delta, \hat{s}) \) be an LTS and \( X \) any set of action names. Then hide \( X \) in \( L \) is the LTS \((S', \Sigma', \Delta', \hat{s})\) such that the following hold:

\[- \Sigma' = \Sigma - X \]
\[- (s, a, s') \in \Delta' \text{ if and only if} \]
\[- a = \tau \land \exists b \in X : (s, b, s') \in \Delta, \text{ or } a \not\in X \land (s, a, s') \in \Delta. \]

We now define the CFFD-model and CFFD-equivalence, which will be our main equivalence notion in this article. We also define CFFD-preorder. Intuitively, preorder means that the smaller process is “better” or “more deterministic” than the larger one.
Definition 6. Let L and L’ be LTSs with the same alphabet.

- The CFFD model of L is the 3-tuple \((S_{\text{fail}}(L), \text{Divtr}(L), \text{Infr}(L))\)
- \(L \equiv_{\text{CFFD}} L' \iff \left[ S_{\text{fail}}(L) = S_{\text{fail}}(L') \land \text{Divtr}(L) = \text{Divtr}(L') \land \text{Infr}(L) = \text{Infr}(L') \right] \)
- \(L \leq_{\text{CFFD}} L' \iff \left[ S_{\text{fail}}(L) \subseteq S_{\text{fail}}(L') \land \text{Divtr}(L) \subseteq \text{Divtr}(L') \land \text{Infr}(L) \subseteq \text{Infr}(L') \right] \)

The traces are not included in the CFFD model because they can be determined from \(S_{\text{fail}}\) and \(\text{Divtr}\) by the equation \(\text{Tr}(L) = \text{Divtr}(L) \cup \{ \sigma \in \Sigma^* \mid (\sigma, 0) \in S_{\text{fail}}(L) \} \) [27].

It should be noted that when certain process-algebraic operators are used, a component called stability must be included in the CFFD model. This one bit of information tells whether or not there are \(\tau\)-transitions from the initial state of the LTS. However, with parallel composition and hiding this component is not needed, so we will not use it here.

An important property of an equivalence is that when a component process in a system is replaced by an equivalent process, the system should remain equivalent to the original one. This is formally captured by the congruence property.

Definition 7. An equivalence \(\simeq\) is a congruence with respect to a process operator \(\text{op}(L_1, \ldots, L_n)\) if and only if \(L_1 \simeq L'_1 \land \cdots \land L_n \simeq L'_n\) implies \(\text{op}(L_1, \ldots, L_n) \simeq \text{op}(L'_1, \ldots, L'_n)\).

CFFD-equivalence is a congruence with respect to parallel composition and hiding. Similarly, CFFD-preorder is a precongruence (monotonic) with respect to parallel composition and hiding [27].

3 Temporal Logic and Fairness Operators

The desired properties of reactive and concurrent systems are often expressed by using linear temporal logic [15, 19]. We next present a straightforward adaptation of the logic to our process-algebraic framework.

Definition 8. A formula is generated by the grammar \(\psi ::= \text{true} \mid a \mid \neg \psi \mid \psi \lor \psi \mid \psi \land \psi\), where \(a\) is an action name. We also use the following denotations: \(\text{false} \equiv \neg \text{true}, \psi \land \phi \equiv (\neg \psi \lor \neg \phi), \psi \Rightarrow \phi \equiv \neg \psi \lor \phi, \Diamond \psi \equiv \text{true} \lor \psi\) ("eventually"), \(\Box \psi \equiv \neg \Diamond \neg \psi\) ("always").

We will define the semantics of formulas on the infinite executions of systems.

Definition 9. Let \(L = (S, \Sigma, \Delta, \bar{s})\) be an LTS. The set of the infinite executions of \(L\) is \(\text{infex}(L) = \{ s_0a_1s_1a_2s_2a_3 \cdots \mid \bar{s} = s_0 \land \forall i \geq 1: s_{i-1} \rightarrow a_i \rightarrow s_i \}\).

Below we use the following notation: if \(\eta = s_0a_1s_1a_2s_2a_3 \cdots\) is an infinite execution, then \(\text{acts}(\eta)\) is the sequence of actions \(a_1a_2a_3 \cdots\) and \(\eta^i\) is the \(i\)th suffix \(s_i a_{i+1} s_{i+1} a_{i+2} \cdots\).
Definition 10. Let $L = (S, \Sigma, \Delta, \delta)$ be an LTS and $\eta = s_0 a_1 s_1 a_2 s_2 a_3 \cdots$ an infinite execution of $L$. Then

- $(L, \eta) \models true$
- $(L, \eta) \models a$ iff $a_1 = a$
- $(L, \eta) \models \neg \psi$ iff not $(L, \eta) \models \psi$
- $(L, \eta) \models \psi \lor \phi$ iff $(L, \eta) \models \psi$ or $(L, \eta) \models \phi$
- $(L, \eta) \models \psi \cup \phi$ iff $\exists j \geq 0 : (L, \eta^j) \models \phi$ and $\forall k, 0 \leq k < j : (L, \eta^k) \not\models \psi$

The properties of reactive systems are usually divided into safety properties, expressing requirements of the form “nothing bad must ever happen”, and liveness properties, expressing requirements of the form “something good must eventually happen” [2]. Fairness properties are liveness properties which state that even though the system makes nondeterministic choices, it does not infinitely favour some choices at the expense of others.

Often to verify liveness properties we first add fairness constraints to the system. These are fairness properties that are assumed to hold of the system, expressing the idea that the modelled system is behaving fairly. To use such assumptions in process-algebraic verification, we want to have, for each fairness constraint $\phi$, a corresponding “fairness operator” $\Phi_\phi$ that can be applied to an LTS, and which produces a new LTS having the same finite behaviour, namely traces and stable failures, and precisely those infinite behaviours (infinite traces and divergences) that can be executed while assuming the fairness constraint.

We should notice, however, that not every fairness constraint can be applied to every system. This is because the constraint may require something that the system is unable to do without also changing its finite behaviour. For this reason we assume that for each fairness formula we have defined a corresponding set of compatible LTSs, denoted by $COMP$, to which $\phi$ may be applied.

However, it will sometimes be very useful to associate a hiding operation with the fairness operator. Namely, we can apply the fairness constraint to a wider class of systems if there is a guarantee that certain actions in the constraint will be hidden immediately after applying the fairness operator. This will be illustrated in Section 7. Therefore, the compatibility set will depend on both $\psi$ and a “hiding set” $H$, which is to be hidden after applying the fairness operator (notice that the latter is different from [20] and [21]). We formulate the requirement as follows:

Definition 11. An operator $\Phi_\phi$ is a fairness operator for formula $\phi$ with compatibility set $COMP(\phi, H)$ if and only if for every $L \in COMP(\phi, H)$

- $Tr(L_1) = Tr(L_2)$ and $Sfail(L_1) = Sfail(L_2)$, and
- $Distr(L_1) = \{ \sigma \in \Sigma^*_{L_1} \mid \exists \eta \in \text{infex}(L) : ((L, \eta) \models \phi) \land \text{restr}(\eta, \Sigma_{L_2}) = \sigma \}$, and
- $Infr(L_1) = \{ \xi \in \Sigma^*_{L_2} \mid \exists \eta \in \text{infex}(L) : ((L, \eta) \models \phi) \land \text{restr}(\eta, \Sigma_{L_2}) = \xi \}$,

where $L_1 = \text{hide } H \text{ in } \Phi_\phi(L)$ and $L_2 = \text{hide } H \text{ in } L$. 
However, some care must be taken in using fairness in the compositional setting of process algebra, because in such a setting other, yet unknown, processes may interfere with the behaviour of the process for which we define the fairness constraint. Firstly, when we are using a particular behavioural equivalence, we should make sure that this a congruence with respect to the fairness operator. Furthermore, the fairness operator would typically be applied to some subprocess $L$ (e.g., a communication channel) which can be placed in a larger context $C[\cdot]$ (e.g., a protocol system). The property of the underlying system expressed by the fairness constraint should remain the same in the larger context. Therefore, within some reasonable limits, it should make no difference whether the same fairness constraint is assumed of $L$ or of the composition $C[L]$. We will refer to these desired properties of the fairness operator as context-independence:

**Definition 12.** Let $\Phi_\phi$ be a fairness operator for formula $\phi$ which is expressed in terms of the actions $\mathcal{F}$. We say that $\Phi_\phi$ is context-independent with respect to $\simeq$, if and only if $\simeq$ is a congruence with respect to $\Phi_\phi$, and for all $L$ in $\text{COMP}(\psi, H)$ it holds that

1. $\Phi_\phi(L) \parallel L' \simeq \Phi_\phi(L \parallel L')$ for any LTS $L'$ such that $\Sigma_L \cap H = \emptyset$
2. hide $X$ in $\Phi_\phi(L) \simeq \Phi_\phi(\text{hide} X \in L)$ for any $X$ such that $X \cap \mathcal{F} = \emptyset$.

4 A Class of Fairness Constraints

Some types of fairness properties (such as weak fairness and strong fairness) require the execution of an action if it is sufficiently often enabled. However, as discussed in [21], it is difficult to use such properties with weak behavioural process-algebraic semantics, because these do not preserve information about the enabledness of actions in infinite executions. Therefore, we here concentrate on fairness properties that can be used with such semantics. These will be fairness properties of the form “if something happens infinitely often, then something else also has to happen infinitely often”.

In [21], fairness formulas of the form $\Box \Diamond a_1 \lor \cdots \lor \Box \Diamond a_m \Rightarrow \Box \Diamond b_1 \lor \cdots \lor \Box \Diamond b_n$ were used. Here, we consider formulas of the form

$$\alpha \Rightarrow \beta$$

where $\alpha$ and $\beta$ are any formulas constructed from action names by using the operators “$\lor$”, “$\land$” and “$\Box \Diamond$” (“infinitely often”), with the restriction that every action name must reside within the scope of at least one “$\Box \Diamond$”-operator (because we are interested in fairness properties, not individual actions).

Formally, $\alpha$ and $\beta$ are any formulas generated by the grammar $\phi ::= \text{false} | \phi \land \phi | \phi \lor \phi | \Box \Diamond \phi_1$ and $\phi_1 ::= \phi_1 \land \phi_1 | \phi_1 \lor \phi_1 | \Box \Diamond \phi_1 | a$, where $a$ is any visible action name. Because we allow $\beta$ to be $\text{false}$, the complete formula may become $\neg \alpha$.

Let us denote the set of formulas of this form by $F$. We next transform the fairness formulas into a normal form where, for technical convenience, the left side is in a conjunctive form and the right side in a disjunctive form; the detailed proof is given in [20].
Proposition 1. Every formula in \( F \) that is not trivially true can be given in the form
\[
A_1 \land A_2 \land \cdots \land A_m \Rightarrow B_1 \lor B_2 \lor \cdots \lor B_n,
\]
where \( A_i = \Box \Diamond a_i^1 \lor \Box \Diamond a_i^2 \lor \cdots \lor \Box \Diamond a_i^{u_i} \), \( B_j = \Box \Diamond b_j^1 \lor \Box \Diamond b_j^2 \lor \cdots \lor \Box \Diamond b_j^{v_j} \), and \( a_i^j \) and \( b_j^j \) are names of actions. If \( n = 0 \), then the formula is \( \neg (A_1 \land A_2 \land \cdots \land A_m) \).

From now on we will assume that the fairness formulas have already been given in the above form. We will denote such a formula by
\[
\psi(A_1, \ldots, A_m; B_1, \ldots, B_n).
\]
We will use \( A_i \) and \( B_j \) to denote the sets of actions \( \{a_i^1, \ldots, a_i^{u_i}\} \) and \( \{b_j^1, \ldots, b_j^{v_j}\} \), respectively, and we will write \( A \) for \( A_1 \cup \cdots \cup A_m \), and \( B \) for \( B_1 \cup \cdots \cup B_n \).

As indicated above, we must define a set of LTSs which are compatible with the fairness formula. One intuitive idea could be to require that the actions \( a_i^k \) always have an alternative \( \tau \)-transition or a transition that will be hidden, so that the system can always nondeterministically choose a different route. In fact, it turns out that a closely related but weaker requirement suffices:

Definition 13. LTS \( L = (S, \Sigma, D, \delta) \) is in \( \text{COMP}(\psi, H) \) if and only if \( A \cup B \subseteq \Sigma \), \( H \cap A = \emptyset \) and \( \forall (\sigma, X) \in S_{\text{fail}}(L) : (\sigma, X \cup H) \notin S_{\text{fail}}(L) \lor (\sigma, X \cup A) \notin S_{\text{fail}}(L) \).

Notice that the given condition can be determined from the CFFD-model of an LTS. It is also straightforward to show that if \( L \) is in \( \text{COMP}(\psi, H) \) then so are \( L \parallel L' \) and \text{hide} \( X \) in \( L \) with the same restrictions as in Definition 12.

5 Fairness LTS and Fairness Operator

We will next define a “fairness LTS” that corresponds to a fairness formula. We first illustrate the idea with some examples. Figure 1 shows a fairness LTS corresponding to the fairness constraint \( \Box \Diamond a \Rightarrow \Box \Diamond b \). It has an infinite number of branches with lengths 1, 2, 3, \ldots and it can execute any finite number of consecutive \( a \)-actions, but not infinitely many \( a \), before executing a \( b \)-action and returning to the initial state. Then, the same can be repeated.

We can easily add new actions to the formula in a disjunctive manner by simply adding new parallel arcs to the LTS with the new action names. Using conjunctive conditions, on the other hand, requires a more elaborate structure. Figure 2 shows a fairness LTS that corresponds to the fairness constraint \( \Box \Diamond a \land \Box \Diamond c \Rightarrow \Box \Diamond b \land \Box \Diamond d \). Like before, there are branches of length 1, 2, 3, and so on. However, now there are two branches of each length, one of which limits action \( a \) to the given finite number and the other which limits action \( c \). Furthermore, each branch has an internal structure which keeps track of which of the two actions \( b \) and \( d \) have been detected so far. Once both \( b \) and \( d \) have been detected we return to the initial state.

We should point out that the fairness LTS is a theoretical concept that allows us to handle the fairness constraint by using the well-known properties of parallel composition, and, as we will later see, we do not have to construct such LTSs in actual verification. The following gives the formal definition of a fairness LTS.
Definition 14. For a formula $\psi(A_1, \ldots, A_m; B_1, \ldots, B_n)$, $L_\psi$ is the LTS $(S, \Sigma, \Delta, \delta)$, where

- $S = \{()\} \cup \{(A, l, r, B) \in 2^A \times \mathbb{N} \times \mathbb{N} \times 2^B \mid \exists i \in \{1, \ldots, m\} : A = A_i \wedge 0 \leq l \wedge \forall j \in \{1, \ldots, n\} : B_j \not\subseteq B\}$
- $\Sigma = A \cup B$ and $\delta = ()$
- $\Delta = \{\hat{\delta}\} \times B \times \{\hat{\delta}\} \cup \{(\hat{\delta}, a, (A, l, 0, \emptyset)) \in \{\hat{\delta}\} \times A \times S \mid l \geq 1 \lor a \in A\}$
  $\cup \{(A, l, r, B), a, (A, l, r-1, B) \in S \times A \times S \mid a \notin A\}$
  $\cup \{(A, l, r, B), b, (A, l, r, B \cup \{b\}) \in S \times B \times S \mid true\}$
  $\cup \{(A, l, r, B), b, \hat{\delta}) \in S \times B \times \{\hat{\delta}\} \mid \exists j : B_j \subseteq B \cup \{b\}\}$

It can be shown that the fairness LTS has the following properties; the proof is given in [20]. It should be noted that our construction is not unique in the sense that there are many LTSs with the same properties, and, indeed, we could use any such LTS, but our construction shows that at least one such LTS exists.

Proposition 2. For $L_\psi = (S, \Sigma, \Delta, \delta)$ it holds that $Tr(L_\psi) = \Sigma^*$, $Divtr(L_\psi) = \emptyset$, $Sfail(L_\psi) \subseteq \Sigma^* \times 2^A$, and $Infr(L_\psi) = \{\eta \in \Sigma^* \mid \eta \models \psi\}$.

Now we define the fairness operator. As indicated above, it works simply by placing the target system in parallel with the fairness LTS.

Definition 15. Given the formula $\psi$, operator $\Psi_\psi^\parallel$ is the following mapping from LTSs to LTSs: $\Psi_\psi^\parallel(L) = L \mid\mid L_\psi$.

The following results show that $\Psi_\psi^\parallel$ really is a fairness operator, in the sense of Definition 11, and context-independent with respect to CFFD. The proofs, which are based on the previous proposition and the properties of parallel composition and hiding, can be found in [20], although the former requires some modifications to cater for the hiding set $H$.

Theorem 1. $\Psi_\psi^\parallel$ is a fairness operator for $\psi$ with compatibility set $COMP(\psi, H)$.

Theorem 2. $\Psi_\psi^\parallel$ is context-independent with respect to $\simeq_{CFFD}$.

It is also straightforward to show that our fairness operators commute, and the compatibility of an LTS is preserved by fairness operators that do not use the actions in the hiding set $H$. 
Fig. 2. A fairness LTS for the constraint $\square \diamond a \land \square \diamond c \Rightarrow \square \diamond b \land \square \diamond d$

6 Algorithm for Verification

In this section we will show how the fairness operators can be used in verification without having to construct infinite systems. As a starting point we assume a system $P$ composed by using "$\ll$" and "hide" from LTSs. Next, we want to add some fairness constraints and we change the system by adding fairness operators $\psi_1\ll, \ldots, \psi_b\ll$ so that each $\psi_i\ll$ is applied to a subsystem that is in $\text{COMP}(\psi_i, H_i)$, and which is, if $H_i$ is nonempty, under a hiding operator that hides $H_i$. The new "fair" system is denoted by $P_{\text{fair}}$.

For technical convenience, we will from now on assume that any actions that are hidden in an expression of the form hide $X$ in $R$ only occur in the subsystem $R$. We do not lose any generality in this assumption; if it does not hold we can simply rename the hidden actions in hide $X$ in $R$ with new, unique names without affecting the end result. Also, we will denote by $\mathcal{F}$ the actions that are used in some fairness formula, and by $\mathcal{F}_h$ those of $\mathcal{F}$ that are also used in some hiding operator.

$P_{\text{fair}}$ can be (and typically is) infinite. Therefore, our aim in the following algorithm is to construct a finite representation of $P_{\text{fair}}$, that is, a finite LTS $P^*_{\text{fair}}$ such that

$$P^*_{\text{fair}} \simeq_{\text{FFD}} P_{\text{fair}}.$$

Step 1: Construct a system $P^\ll$ in the same way as $P$ except that the actions $\mathcal{F}_h$ are not hidden.
The significance of this step is revealed by the following proposition, which shows that the fairness operators and the hiding of the related actions can be moved to the outmost level in the system, so that we can examine their effect on the remaining finite parameter system $P^1$. The proof is based on the context-independence property and other properties of operators. The (lengthy) details are given in [20].

**Proposition 3.** $P_{fair} \simeq_{CFFD} \text{hide } \mathcal{F}_h \text{ in } \psi_1 \| (\ldots \psi_k \| (P^1) \ldots )$.

Intuitively, any divergence (cycle of $\tau$-actions) that shows up in the complete system is caused by a cycle consisting of actions $\mathcal{F}_h \cup \{\tau\}$ in $P^1$. Furthermore, each such cycle is a part of a unique maximal strongly connected component of actions $\mathcal{F}_h \cup \{\tau\}$. Next we identify all such components (see the next section for illustrating examples), which we will call $\mathcal{F}_h$-components.

**Step 2:** Taking into account only the actions $\mathcal{F}_h \cup \{\tau\}$ in $P^1$, identify the maximal nontrivial strongly connected components $C_1, \ldots, C_l$ (e.g., by using Tarjan’s algorithm [1]).

For each component $C$, we remove all transitions and states of $C$ except one state $s_C$, which can be any state of $C$, but if $C$ contains the initial state, this is selected as $s_C$. We redirect transitions in and out of $C$ into $s_C$. If there are transitions between states of $C$ which themselves are not part of $C$, those transitions become loops from $s_C$ to itself. We also check (see below) whether $C$ contains an infinite execution (starting from any state) that is allowed by the fairness formulas. If it does, we add a $\tau$-loop from $s_C$ to itself. When this has been done for each $C$, we hide the actions $\mathcal{F}_h$.

**Step 3:** for $C \in \{C_1, \ldots, C_l\}$ do

- if $\delta_{P^1} \in S_C$ then $s_C := \delta_{P^1}$ else choose any $s_C \in S_C$
- $S_{P^1} := S_{P^1} \setminus (S_C \setminus \{s_C\})$
- $\Delta_{P^1} := \{(s_1, b, s_2) \mid \exists (s'_1, b, s'_2) \in \Delta_{P^1} \setminus \Delta_C :$
  - $(s'_1 \not\in S_C \land s_1 = s'_1 \lor s'_1 \in S_C \land s_1 = s_C) \land$
  - $(s'_2 \not\in S_C \land s_2 = s'_2 \lor s'_2 \in S_C \land s_2 = s_C)\}$
- if $C$ has an infinite execution $\eta$ such that $\eta \models \psi_1, \ldots, \psi_k$ then $\Delta_{P^1} := \Delta_{P^1} \cup \{(s_C, \tau, s_C)\}$

**Step 4:** Hide the actions $\mathcal{F}_h$.

The result of the above steps is our finite model $P^*_{fair}$. We will show that $P^*_{fair}$ is an exact model of $P_{fair}$ if one exists, and otherwise it is a conservative estimate of it. However, let us first consider the complexity of the construction. Other parts of the algorithm can be done in time linear in the number of states and transitions, except checking for allowed infinite executions in a component. Infinite sequences of $\tau$-actions are allowed by the formulas and can be detected with a depth-first search. One way to detect allowed sequences with infinitely many $\mathcal{F}_h$-actions is by using a variation of the B"uchi automata [25] based verification of linear temporal logic [10]. We can construct an automaton with
\[ \text{hide } X \text{ in } \begin{array}{c}
\begin{array}{c}
\text{P}_1
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{S}
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{P}_2
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{P}
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{use}_{-res_1}, v_1, p_1
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{p}_1, p_2, v_1, v_2
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{use}_{-res_2}, v_2, p_2
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{use}_{-res_2}
\end{array}
\end{array} \end{array} \]

Fig. 3. The semaphore system (alphabets shown), where \( X = \{ p_1, v_1, \text{use}_{-res_1}, p_2, v_2 \} \)

\( O(k \prod_{i=1}^{k} |\psi_i|) \) states that accepts precisely the infinite \( \mathcal{F}_k \)-sequences that fulfill the formulas \( \psi_1, \ldots, \psi_k \); here, \( |\Delta_C| \) is the number of transitions in \( C \), and \( |\psi_i| \) is the length of the formula \( \psi_i \). In this way we can show (see [20] for details) that checking whether an \( \mathcal{F}_k \)-component \( C \) contains an acceptable infinite execution can be done in time \( O(|\Delta_C| k \prod_{i=1}^{k} |\psi_i|) \).

The following result states that \( P_{\text{fair}}^* \) has precisely the same stable failures and divergences as \( P_{\text{fair}} \). This can be shown by a modification of the proof of Theorem 40 in [20] to cater for the hiding set \( H \); the proof is based on the fact that replacing the components preserves traces of \( \Sigma_P \)-actions in the system, on the definition of compatibility, and the fact that our construction adds \( \tau \)-loops to precisely the states that replace components with allowed infinite executions.

**Proposition 4.** \( S\text{fail}(P_{\text{fair}}^*) = S\text{fail}(P_{\text{fair}}) \) and \( \text{Divtr}(P_{\text{fair}}^*) = \text{Divtr}(P_{\text{fair}}) \)

It only remains to consider the set of infinite traces, \( \text{Inftr} \). It can be shown that our construction preserves both finite and infinite traces, so \( \text{Inftr}(P_{\text{fair}}^*) = \text{Inftr}(P) \). Therefore, the question is now whether the fairness operators also preserve infinite traces, that is, whether \( \text{Inftr}(P_{\text{fair}}) = \text{Inftr}(P) \). If they do, then our model \( P_{\text{fair}}^* \) and \( P_{\text{fair}} \) are equivalent. However, if they do not, then it turns out that a finite model of \( P_{\text{fair}} \) does not even exist. This is stated in the following; we again refer to the proof of Theorem 40 in [20].

**Theorem 3.** If \( \text{Inftr}(P_{\text{fair}}) = \text{Inftr}(P) \) then \( P_{\text{fair}}^* \simeq_{\text{CFFD}} P_{\text{fair}} \).

If \( \text{Inftr}(P_{\text{fair}}) \neq \text{Inftr}(P) \), then there does not exist a finite LTS \( Q \simeq_{\text{CFFD}} P_{\text{fair}} \).

We still need an algorithm for checking whether \( \text{Inftr}(P_{\text{fair}}) = \text{Inftr}(P) \). Unfortunately, it turns out that the complexity of checking this is higher than that of constructing the model \( P_{\text{fair}} \). By using a similar construction as above we can reduce this problem to language containment of Büchi automata, and solve it by using the Büchi automaton complementation construction in [24] (with a small modification to include \( \tau \)-actions), as shown in [20]. The result is that the problem can be decided in \( \text{PSPACE}(|S_P| k \prod_{i=1}^{k} |\psi_i|) \).

Fortunately, even in the negative case, \( P_{\text{fair}} \) can only have some infinite traces that \( P_{\text{fair}} \) does not have. Therefore, in every case \( P_{\text{fair}}^* \), the product of the algorithm, is a conservative estimate of the possibly infinite \( P_{\text{fair}} \), and so it is always safe to use it in place of \( P_{\text{fair}} \) in verification.

**Theorem 4.** \( P_{\text{fair}} \leq_{\text{CFFD}} P_{\text{fair}}^* \).

It is also important to remember that the system \( P^! \) which is used as a
parameter in the construction, is not the complete state-space of the original system, but an intermediate system where the actions of the fairness formulas have been left visible. $P^1$ can be constructed by using any semantics-preserving reduced LTS construction method.

7 Examples

As a simple example, consider a semaphore $S$ that controls the access of two processes $P_1$ and $P_2$ to a critical region, as shown in Figure 3. We look at the system from the point of view of $P_2$, and hide all actions except $\text{use\_res}_2$, and reduce the system with a CFFD-preserving reduction algorithm. We notice that there is a divergence in the result, so $P_2$ may never get access to the resource. We therefore add the fairness constraint $\psi \equiv \Box \Diamond p_1 \Rightarrow \Box \Diamond p_2$ which forces $S$ to eventually give access to $P_2$. Notice that we cannot add the constraint directly to $S$; intuitively, this is because if $S$ were connected with a different $P_2$-process that could refuse $p_2$, this would create a new deadlock. However, when we connect $P_1$ and $P_2$ to $S$, we can see that the system is in $COMP(\psi, \{p_2\})$. We construct the LTS $P^1$, and the result has one $F_k^r$-component, as shown in Figure 4. This $F_k^r$-component does not have an allowed infinite execution, so when we finish the algorithm and reduce the result, we get the rightmost process in Figure 4, and this behaviour is clearly satisfactory.

We will next consider the classic example of the alternating bit protocol [4], which is used for sending data messages over unreliable channels. The protocol is based on retransmitting data messages for which an acknowledgement message does not arrive in time, and using sequence numbers 0 and 1 to distinguish new messages from retransmissions. The sender $S$, receiver $R$, data channel $DC$ and acknowledgement channel $AC$ of the protocol are shown in Figure 5.

After receiving a message, $DC$ chooses either the action $\text{passd}$ (pass the
data message through) or *lost* (lose it). *AC* works similarly. When the entire system is constructed and reduced, we see that there are two $\tau$-loops in the behaviour, as shown in Figure 6. We can guess that these represent an infinite sequence of message losses and retransmissions. Therefore, we will first try applying the condition $\psi_{DC} \equiv \Box \Diamond \text{lost} \Rightarrow \Box \Diamond \text{passed}$ on $DC$. We notice that $DC \in \text{COMP}(\psi_{DC}, \{\text{passed}\})$, so we can use it on $DC$. We obtain a $P^1$ that has 11 states and four $F^1_k$-components, one of which has an allowed infinite execution. After replacing the components, hiding, and reducing the result, we get the second process in Figure 6. We notice that one of the divergences has disappeared, but one remains. We therefore add a similar requirement for the acknowledgement channel, $\psi_{AC} \equiv \Box \Diamond \text{lost} \Rightarrow \Box \Diamond \text{passed}$. In the new $P^1$ (with 52 states), none of the $F^1_k$-components contain an allowed infinite execution. The final result is the rightmost LTS in Figure 6, which clearly satisfies any reasonable specification of the behaviour of the protocol.

8 Conclusions

In this article we have shown how a class of fairness constraints can be added to process-algebraic systems in a consistent way. We have presented an algorithm for constructing a finite representation of the resulting system in every case that one exists. The result can be placed as a component in further compositional analysis just like any other LTS. A remaining task is to build the approach into tools supporting LTS-based verification. Also, we believe that the theory can be extended to other types of fairness constraints and a wider class of systems.

References