Fair SMG and Linear Time Model Checking

Howard Barringer, Michael D. Fisher and Graham D. Gough *

Department of Computer Science
University of Manchester
Oxford Road
Manchester, M13 9PL

Abstract

SMG [GB88] is a system designed to generate a finite state model of a program from the program itself and an operational semantics for the programming language. Model-checking can then be used to verify that the program satisfies a set of desired temporal properties.

In this paper we first show how we have incorporated notions of fairness into SMG; in particular, a user is now able to define semantics with "fair" constructs, for example, parallel composition, repetitive choice, etc. A program may contain several fair constructs, each with its own fairness type. Secondly we describe a practical approach to model checking of linear temporal formulae over the fair structures generated by the new SMG_F. Our approach is a refinement and extension of the fair-satisfiability algorithms, presented earlier by Lichtenstein and Pnueli [LP85], together with techniques developed in our practical implementations of decision procedures for linear temporal logic [Gou84].

1 Introduction

In this paper, we present an approach for handling fairness in our *generic* system for the verification of temporal properties of finite state programming languages. The system couples the verification paradigm based on model checking of finite programs [CES86, LP85, RRSV87] with language presentation via formal semantic description such as Structural Operational Semantics [Plo81]. The paper [GB88] presented an overview of our basic system. In the system described there, fairness was handled in a rather direct and visible fashion. Additional program variables were explicitly introduced by the user of SMG and used to "specify" the fair paths of the program in temporal assertions. As we indicated in that paper, "*the obligation on the programmer to include extra information that is actually a consequence of the language semantics is obviously unsatisfactory*". We mentioned two possible ways forward. The first was to follow the original approach, but achieve it automatically using the given semantic description and program. The second suggestion was to introduce some form of transition labelling onto the edges of the state machine and modify the model checkers to make use of the labelling to restrict their search space. It is the latter approach that we have actually adopted and present here.

In the following sections we provide a brief overview of SMG, the system for generating finite state machines from language semantics and finite state program, introduce the extensions necessary to extract the "fairness" labelling, then describe our approach to model checking of linear-time formulae. The latter is based on techniques used in our *practical* decision procedures for linear-time temporal formulae [Gou84] and the satisfiability algorithms presented in [LP85].

*This research has been performed as part of the TEMPLE project and is fully supported under Alvey PRJ/SE/054 (SERC grant GR/D/57492)

2 Extending SMG for Fair Constructs

2.1 SMG Structure

The basic structure of the state machine generator (compiler) is shown in figure 1.

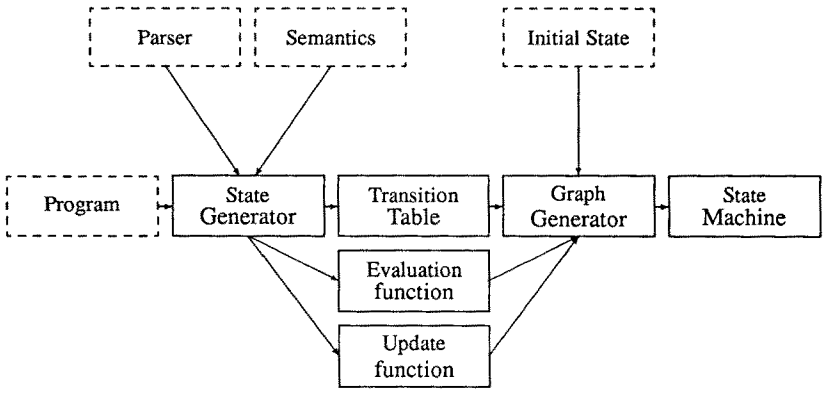

Figure 1: SMG architecture

SMG takes as input a parser and operational rewrite rule semantics for the user's language $\mathcal{L}$, together with a program in $\mathcal{L}$ and an initial state assignment for the program variables. It then constructs a *transition table* for the program $\mathcal{P}$ by instantiating, for the parse tree of $\mathcal{P}$, the transition rules given in the semantics. Using the given initial states of $\mathcal{P}$, together with evaluation and state update functions obtained from the rewrite rule semantics, a *state transition graph* is then generated for $\mathcal{P}$.

The semantics is given, in S.O.S. style [Plo81], as a set of labelled transition rules of the form

$$S_1 \xrightarrow{ec,\ sm,\ em} S_2$$

and inference rules of the form

$$\frac{R_1 \ldots R_n}{R}$$

where S_1, S_2 are program phrases, *ec* is a boolean expression defining the enabling condition for the transition, *sm* is the state modification effected by the transition, *em* the environment modification and $R, R_1 \ldots R_n$ are transition rules.

The output format of the state transition graph can be chosen as appropriate for the model checker to be used, in particular for mcb [Bro86] or a linear time temporal logic model checker such as MDF [Fis89].

2.2 A shared variables language

A simple shared variables language has been chosen to illustrate our approach to handling fair constructs in SMG (other languages will be considered in a later paper). It consists of Boolean variables and expressions, assignment, sequential composition, guarded choice, repetitive guarded choice and parallel composition. Repetitive choice and parallel composition are each presented in four flavours corresponding to *unrestricted, impartial, just* and *fair* forms of fairness constraint [LPS81](see Definition2.3).

Language Syntax We describe the syntax (omitting that of expressions) using a BNF-like notation. Assuming the syntactic classes *Var*, *Expression* and *Name* for the obvious entities, we have

$$
\begin{array}{lll}
\textit{Program} & ::= & \textit{Statement-list}. \\
\textit{Statement-list} & ::= & \textit{Statement} \mid \textit{Statement}; \textit{Statement-list} \\
\textit{Statement} & ::= & \textit{Var} := \textit{Expression} \mid [\textit{Choice}] \mid \\
 & & [\textit{Choice}]^*_{ft} \mid \textit{Parallel} \mid \textbf{skip} \\
\textit{Choice} & ::= & \textit{Expression} \rightarrow \textit{Statement} \mid \\
 & & \textit{Expression} \rightarrow \textit{Statement} \ [] \ \textit{Choice} \\
\textit{Parallel} & ::= & \textit{Statement} \parallel_{ft} \textit{Statement} \\
ft & ::= & \mid I \mid J \mid F
\end{array}
$$

Actually, this is not the current form of parser input, SMG currently requires the user to define or modify a yacc [Joh79] based parser.

Language Semantics A dynamic semantics for the language is given below in the SOS style mentioned earlier. We assume

$$SL \in \textit{Statement-list}, S, S_i \in \textit{Statement}$$
$$x \in \textit{Var}, e, e_i \in \textit{Expression}$$

For clarity of exposition, we omit the environment component.

$$\textbf{skip} \ . \xrightarrow{\textbf{true},\ [\,]} \textbf{skip} \ .$$

$$\textbf{skip} \ ; SL \xrightarrow{\textbf{true},\ [\,]} SL$$

$$\textbf{skip} \parallel_{ft} S \xrightarrow{\textbf{true},\ [\,]} S$$

$$S \parallel_{ft} \textbf{skip} \xrightarrow{\textbf{true},\ [\,]} S$$

$$x := e \xrightarrow{\textbf{true},\ [x/e]} \textbf{skip}$$

$$[] \ e_i \rightarrow S_i] \xrightarrow{e_i,\ [\,]} S_i$$

$$[] \ e_i \rightarrow S_i] \xrightarrow{\bigwedge \neg e_i,\ [\,]} [] \ e_i \rightarrow S_i]$$

$$[] \ e_i \rightarrow S_i]^*_{ft} \xrightarrow{e_i,\ [\,]} S_i \ ; \ [] \ e_i \rightarrow S_i]^*_{ft}$$

$$[] \ e_i \rightarrow S_i]^*_{ft} \xrightarrow{\bigwedge \neg e_i,\ [\,]} \textbf{skip}$$

and the inference rules

$$\frac{S \xrightarrow{ec,\ em} S'}{S \ ; SL \xrightarrow{ec,\ em} S' \ ; SL}$$

$$\frac{S_1 \xrightarrow{ec,\ sm} S_1'}{S_1 \parallel_{ft} S_2 \xrightarrow{ec,\ sm} S_1' \parallel_{ft} S_2 \ , \ S_2 \parallel_{ft} S_1 \xrightarrow{ec,\ sm} S_2 \parallel_{ft} S_1'}$$

and the declarations for "fair" operators

$$
\begin{array}{lll}
\text{Impartial} & = & \{[\,]^*_I, \parallel_I\} \\
\text{Just} & = & \{[\,]^*_J, \parallel_J\} \\
\text{Fair} & = & \{[\,]^*_F, \parallel_F\}
\end{array}
$$

All operator symbols not appearing in this declaration part are assumed to be unrestricted.

The new SMG_F generates a state machine as before by unwinding the given program using the supplied semantic rules. In addition, labels are generated whenever a transition involving a "fair" non-unary operator is made. Each instance of a fair operator in the program is supplied with a unique set of labels. Furthermore SMG_F constructs fairness constraints for each "fair" operator (see section 2.3 for formal details).

Example 1 Consider the program schema

$$[b \to c := \neg c \,[]\, b \wedge c \to b := \neg b]^*_{ft}.$$

which can have ft instantiated as empty string (for unrestricted choice), I for impartial choice, J for just choice and F for fair choice. The state structure generated by SMG_F, when started in a state with b and c true, is as in Figure 2. State s_1, the initial state with b and c true, has two possible outgoing transitions

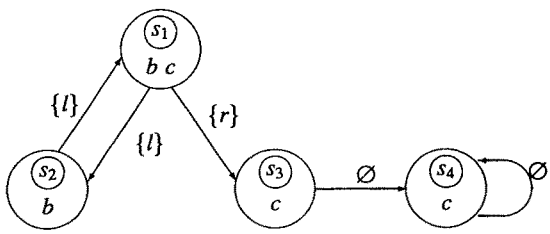

Figure 2: State structure for example 1

corresponding to the choices of the program. The left choice has been labelled $\{l\}$ and the right choice has been labelled $\{r\}$. We say s_1 is *enabled* for l and r. State s_2 on the other hand is only enabled for l; it is disabled for r. The transition representing exit from the repetitive choice is not labelled by a guard choice label.

The fairness constraint generated[1] is

$$\{\langle\{s_1, s_2, s_3\}, \{l, r\}, ft\rangle\}$$

where ft is the fairness type obtained from the program instance. The first component of the constraint is a set of states of interest, the second is a set of possible choices at the states of interest and the third component defines the fairness type. The interpretation of these constraints is defined formally in the next section.

Example 2 Consider next the program schema

$$[b \to d := \neg d \,;\, d := \neg d \,[]\, b \wedge c \to b := \neg b]^*_{ft_1} \,\|_{ft_2}\, [b \to c := \neg c]^*.$$

started in a state with b and c both true. Figure 3 presents the state structure generated by SMG_F. Again, state s_1 is the initial state with b and c true, and d false. All states except s_6, the terminal state, involve

[1] When ft is empty (i.e., unrestricted), no fairness constraint is generated.

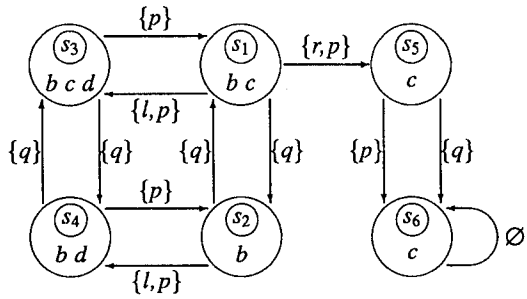

Figure 3: State structure for example 2

transitions of the parallel construct. These transitions have been labelled by p (for the first process) and q (for the second). s_1, s_2, and s_3 are states involving transitions of the repetitive non-unary choice.

The fairness constraints generated are

$$\{\langle\{s_1, s_2, s_3\}, \{l, r\}, ft_1\rangle\}$$
$$\{\langle\{s_1, s_2, s_3, s_4, s_5\}, \{p, q\}, ft_2\rangle\}$$

This program will terminate if the choice fairness type ft_1 is I, and the parallel fairness type is not unrestricted, but not under any other conditions.

2.3 Fair Automaton Structure

The automata constructed by SMG_F are similar to the fair structures described in [CVW86]. The main difference is the expression of fairness constraints based on a set of pairs of state sets and edge-label sets. Furthermore, for simplicity of presentation, it is assumed that SMG_F generates automata with only infinite paths; terminating executions of programs are handled in the normal way by looping on the "final" state. The motivation for these modifications is essentially to keep the structures over which temporal formulae are interpreted close to the structures generated by SMG_F.

Definition 2.1 A fair structure $\mathcal{A}_F$ is a tuple $\langle S, L, T, V, F, S_0\rangle$ where

- S is a nonempty, finite, set of states,
- L is a nonempty, finite, set of labels,
- T is a labelled transition relation on S – a subset of $S \times 2^L \times S$, and is total in its first component,
- V assigns truth values to atomic propositions, PROP, i.e. $V : S \to 2^{\text{PROP}}$,
- F is the fairness condition used to define the fair paths of the structure – a subset of $2^S \times 2^L \times \{\text{IMPARTIAL}, \text{JUST}, \text{FAIR}\}$
- S_0 is a subset of S giving the initial states of the structure.

Definition 2.2 An ω-path, σ, of a fair structure $\mathcal{A}_F$, is a total function $\sigma : \mathbf{N} \to S$, such that for all $i \in \mathbf{N}$, there is some labelling, $ls \subseteq 2^L$, such that $\langle\sigma(i), ls, \sigma(i+1)\rangle \in T$.

A set of states X *occurs infinitely* on an ω-path, σ, **iff**

$$\forall x \in X \cdot |\{i \mid \sigma(i) = x, i \in \mathbf{N}\}| = \omega$$

A set of labels M *occurs infinitely* on an ω-path, σ, of structure $\mathcal{A}_F$ **iff**

$$\forall m \in M \cdot |\{i \mid \langle \sigma(i), ls, \sigma(i+1)\rangle \in T, m \in ls, i \in \mathbf{N}\}| = \omega$$

A set of labels M is *disabled* in a state s of structure $\mathcal{A}_F$ **iff** no transition from s has a label set containing elements of M, i.e.

$$\forall s' \in S, \forall X \in 2^L \cdot \langle s, X, s'\rangle \in T \Rightarrow X \cap M = \emptyset.$$

Definition 2.3 Given a fairness constraint $f \in F$ of structure $\mathcal{A}_F$, an f-path is an ω-path σ satisfying one of the following:

$f = \langle E, M, \text{IMPARTIAL}\rangle$:
if a nonempty subset E' of E occurs infinitely on σ then
M must occur infinitely on σ,

$f = \langle E, M, \text{JUST}\rangle$:
if a nonempty subset E' of E occurs infinitely on σ then
either some nonempty subset of M is disabled on some state of E'
or M occurs infinitely on σ,

$f = \langle E, M, \text{FAIR}\rangle$:
if a nonempty subset E' of E occurs infinitely on σ then
for every $m \in M$
either there is a state $\sigma(i)$ beyond which $\{m\}$ is disabled
in every recurring state of E'
or $\{m\}$ occurs infinitely on σ.

Definition 2.4 A path σ of a structure $\mathcal{A}_F$ is *fair* w.r.t. a set of fairness constraints G if for every fairness constraint $f \in G$, σ is an f-path.

Definition 2.5 A *fair path* of a structure $\mathcal{A}_F$ is a path which is fair w.r.t. the set of fairness constraints F of $\mathcal{A}_F$.

Lemma 2.1 *Given a state s in a structure $\mathcal{A}_F$ and a set of* JUST *or* FAIR *fairness constraints G, then there exists a path from s which is fair w.r.t. G.*

One way to justify this result requires the notions of maximal strongly connected components (MSCCs) and terminal MSCCs, with which we assume the reader is familiar. For, given any state s of $\mathcal{A}_F$, either s lies in a terminal MSCC, $\mathcal{T}$, of $\mathcal{A}_F$ or there is a path that leads from s to such a terminal MSCC. Construct from $\mathcal{T}$ an ω-path that traverses every edge in $\mathcal{T}$ infinitely often; this path obviously satisfies *any* JUST or FAIR constraint.

Such a result does not hold in the case of arbitrary impartiality constraints, $\langle E, M, \text{IMPARTIAL}\rangle$, simply because we can, for example, choose the set M to be distinct from any labels occurring on edges of $\mathcal{A}_F$; this does not pose a problem for JUST and FAIR constraints since they have disabledness guards.

The above result leads to a useful optimisation in the algorithm for checking satisfiability, presented in section 3.4; if a formula is found to be satisfied on a finite prefix, the result ensures that the prefix can be extended to a JUST or FAIR path.

Example Consider the fair structure $\mathcal{A}_F$, shown in Figure 2, which would be generated from the program schema

$$[b \to c := \neg c \;[]\; b \wedge c \to b := \neg b]^*_{ft}.$$

with an initial state in which b and c are true. This program terminates if the repetitive guarded choice is fair or impartial, but not if it is merely just. In particular, we have

$$S = \{s_1, s_2, s_3, s_4\}$$

$L = \{l, r\}$
$T = \{(s_1, \{l\}, s_2), (s_1, \{r\}, s_3), (s_2, \{l\}, s_1), (s_3, \emptyset, s_4), (s_4, \emptyset, s_4)\}$
$V = \{s_1 \mapsto \{b, c\}, s_2 \mapsto \{b\}, s_3 \mapsto \{c\}, s_4 \mapsto \{c\}\}$

If the fairness condition F were $\{\langle\{s_1, s_2, s_3\}, \{l, r\}, \text{IMPARTIAL}\rangle\}$, then all fair paths of $\mathcal{A}_F$ containing infinitely many s_1, s_2, or s_3 states would have to have infinitely many $\{l\}$-transitions and infinitely many $\{r\}$-transitions; there are no such paths in $\mathcal{A}_F$. Therefore the only fair paths of $\mathcal{A}_F$ are those that have the infinite tail

$$\ldots\ldots s_1 \xrightarrow{\{r\}} s_3 \xrightarrow{\emptyset} s_4 \xrightarrow{\emptyset} s_4 \ldots\ldots$$

If the fairness condition F is $\{\langle\{s_1, s_2, s_3\}, \{l, r\}, \text{JUST}\rangle\}$, then the structure admits the fair path

$$s_1 \xrightarrow{\{l\}} s_2 \xrightarrow{\{l\}} s_1 \xrightarrow{\{l\}} s_2 \xrightarrow{\{l\}} \ldots\ldots \tag{1}$$

This is because the set $\{r\}$ is disabled for state s_2; the only transition from s_2 is labelled by $\{l\}$.

If the fairness condition is $\{\langle\{s_1, s_2, s_3\}, \{l, r\}, \text{FAIR}\rangle\}$, then the only fair paths of the structure are those that have the infinite tail

$$\ldots\ldots s_1 \xrightarrow{\{r\}} s_3 \xrightarrow{\emptyset} s_4 \xrightarrow{\emptyset} s_4 \ldots\ldots$$

The path (1) is no longer fair since, although states s_1 and s_2 occur infinitely, $\{l, r\}$ is not disabled in either s_1 or s_2, and $\{l, r\}$ does not occur infinitely on σ.

3 Fair Satisfiability Checking of Linear Temporal Logic

We present here an algorithm for checking satisfiability of linear time temporal logic formulae in the $\mathcal{A}_F$ structure generated by SMG_F. Note that we are really concerned in determining whether a formula φ is satisfied on all fair paths of a structure $\mathcal{A}_F$, generated from some program P. We therefore determine whether $\neg\varphi$ is satisfied on some fair path of the structure $\mathcal{A}_F$; if it is not satisfiable, then φ will be satisfied on all fair paths of $\mathcal{A}_F$.

In [LP85], Lichtenstein and Pnueli describe an approach to fair satisfiability checking of finite state programs. Given a finite state machine P, and a linear time temporal formula φ, they construct a structure which is, essentially, the product of P and a maximal tableau for φ; they then check for satisfying paths within the resulting product structure. The time complexity of this model checking is linear in the size of the resulting structure which is linear in the size of program but exponential in the size of the formula. Although for "small" formula this may appear reasonable, the suggested construction is unworkable in practice due to the large constants involved, particularly in the construction of the maximal tableau.

The approach we adopt here uses techniques that we have developed for building practical linear time temporal logic decision procedures. Rather than constructing a maximal tableau, e.g. as in [LP85], we construct a smaller tableau which contains just enough information to characterise the possible models. Furthermore, since our program structures are also kept minimal, we make a double gain. Of course we do not eliminate the exponential in the complexity, but the constants are significantly reduced, making linear temporal logic model checking rather more feasible.

3.1 Linear Time Temporal Logic

The temporal logic used, $\mathcal{TL}$, is a classical propositional logic over a set of propositions, PROP, augmented by the temporal operators $\bigcirc$ and $\mathcal{U}$. Additional operators are defined by

$$\begin{aligned} \Diamond\varphi &\stackrel{\text{def}}{=} \textbf{true}\ \mathcal{U}\ \varphi \\ \Box\varphi &\stackrel{\text{def}}{=} \neg\Diamond\neg\varphi \\ \varphi\mathcal{W}\psi &\stackrel{\text{def}}{=} \varphi\ \mathcal{U}\ \psi \vee \Box\varphi \end{aligned}$$

Formulae are interpreted over linear, discrete, infinite sequences and it is assumed, in the sequel, that σ refers to an ω-path of the structure $\mathcal{A}_F$.

Definition 3.1 A linear time temporal formula φ is *fairly* satisfied in structure $\mathcal{A}_F$ from state s, denoted by $\langle \mathcal{A}_F, s\rangle \models_{\mathcal{F}} \varphi$ **iff** there is a *fair* path σ in $\mathcal{A}_F$ such that $\sigma(0) = s$ and $\langle \mathcal{A}_F, \sigma\rangle \models \varphi$. The satisfiability relation $\models$ is defined by:-
for $\varphi, \psi \in \mathcal{TL}$, $p \in$ PROP

$$
\begin{array}{lll}
\langle \mathcal{A}_F, \sigma\rangle \models \textbf{true} & & \\
\langle \mathcal{A}_F, \sigma\rangle \not\models \textbf{false} & & \\
\langle \mathcal{A}_F, \sigma\rangle \models p & \text{iff} & p \in V(\sigma(0)) \\
\langle \mathcal{A}_F, \sigma\rangle \models \neg\varphi & \text{iff} & \langle \mathcal{A}_F, \sigma\rangle \not\models \varphi \\
\langle \mathcal{A}_F, \sigma\rangle \models \varphi \wedge \psi & \text{iff} & \langle \mathcal{A}_F, \sigma\rangle \models \varphi \text{ and } \langle \mathcal{A}_F, \sigma\rangle \models \psi \\
\langle \mathcal{A}_F, \sigma\rangle \models \bigcirc\varphi & \text{iff} & \langle \mathcal{A}_F, \sigma^{(1)}\rangle \models \varphi \\
\langle \mathcal{A}_F, \sigma\rangle \models \varphi\, \mathcal{U}\, \psi & \text{iff} & \text{there is an } i \geq 0, \text{ with } \langle \mathcal{A}_F, \sigma^{(i)}\rangle \models \psi \text{ and} \\
 & & \text{for all } j \text{ with } 0 \leq j < i,\ \langle \mathcal{A}_F, \sigma^{(j)}\rangle \models \varphi
\end{array}
$$

It can be shown that any formula of $\mathcal{TL}$ is equivalent to a formula in which negations appear only on propositions. We assume that all formulae are in this normalised form.

3.1.1 Alpha and Beta formulae

We classify all normalised formulae in $\mathcal{TL}$ into four types :-

i) Literals, i.e. propositions and their negations.
ii) Next-time formulae, i.e. formulae whose leading connective is $\bigcirc$.
iii) α-formulae, i.e. conjunction-like formulae.
iv) β-formulae, i.e. disjunction-like formulae.

The table below makes this classification explicit.

α	α_1	α_2	β	β_1	β_2
$\varphi \wedge \psi$	φ	ψ	$\varphi \vee \psi$	φ	ψ
$\Box\varphi$	φ	$\bigcirc\Box\varphi$	$\Diamond\varphi$	φ	$\bigcirc\Diamond\varphi$
			$\varphi\, \mathcal{U}\, \psi$	ψ	$\varphi \wedge \bigcirc(\varphi\, \mathcal{U}\, \psi)$
			$\varphi \mathcal{W} \psi$	ψ	$\varphi \wedge \bigcirc(\varphi \mathcal{W} \psi)$

If φ is an α (or β) formula then the formulae formed according to the rules α_1 and α_2 (or β_1 and β_2) are called the α-components (β-components) of φ.

We define a function $\alpha^*: \mathcal{TL} \to 2^{\mathcal{TL}}$ by

$$
\begin{array}{ll}
\alpha^*(\varphi) = \{\varphi\} & \text{if } \varphi \text{ is a literal, a } \beta \text{ formula or a next-time formula} \\
\alpha^*(\varphi) = \alpha^*(\varphi_1) \cup \alpha^*(\varphi_2) & \text{if } \varphi \text{ is an } \alpha \text{ formula with components } \varphi_1, \varphi_2
\end{array}
$$

The function α^* has the following useful (and obvious) property:-

Lemma 3.1 $\langle \mathcal{A}_F, \sigma\rangle \models \varphi$ **iff** $\langle \mathcal{A}_F, \sigma\rangle \models \alpha^*(\varphi)$.
(where $\langle \mathcal{A}_F, \sigma\rangle \models \Phi$, *for* Φ *a subset of* $\mathcal{TL}$ **iff** *for all* $\phi \in \Phi$, $\langle \mathcal{A}_F, \sigma\rangle \models \phi$*).*

We assume below that α^* is extended in the natural way to $\alpha^*: 2^{\mathcal{TL}} \to 2^{\mathcal{TL}}$

3.1.2 Eventualities

In any interpretation in which either of the formulae $\varphi\, \mathcal{U}\, \psi$ or $\Diamond\psi$ is satisfied there must be a future state in which ψ is satisfied. To restate this more precisely, we have

if $\langle \mathcal{A}_F, \sigma\rangle \models \varphi\, \mathcal{U}\, \psi$, then there is a k, with $k \geq 0$ and $\langle \mathcal{A}_F, \sigma^{(k)}\rangle \models \psi$.
if $\langle \mathcal{A}_F, \sigma\rangle \models \Diamond\psi$, then there is a k, with $k \geq 0$ and $\langle \mathcal{A}_F, \sigma^{(k)}\rangle \models \psi$.

Both of the above formulae are β-formulae with β_1 component ψ. Denote either formula by ξ and its β components by ξ_1 and ξ_2, so that $\xi_1 = \psi$.
The formula ξ is satisfied in an interpretation if and only if either ξ_1 or ξ_2 is satisfied, but can not be satisfied in an interpretation consisting of an infinite sequence of states in which ξ_2 is satisfied and ξ_1 is not satisfied; both formulae imply that eventually there is a state in which ξ_1 is satisfied.

We say that $\Diamond\psi$ and $\varphi\ \mathcal{U}\ \psi$ have an associated eventuality ψ (or eventuality set $\alpha^*(\psi)$).

3.2 Properties of subsets of $\mathcal{TL}$

In subsequent sections we describe the construction of a structure, $\mathcal{D}_\phi$, which can be used to generate all models of a formula ϕ. The nodes of this structure will be labelled with sets of temporal formulae. In this section we describe the properties of these label sets which ensure that $\mathcal{D}_\phi$ does indeed generate all models of ϕ and yet does not contain redundant nodes.

Definition 3.2 A set of temporal formulae $\Phi \in 2^{\mathcal{TL}}$ is said to be *proper* iff it satisfies

i) If p is a proposition and $\neg p \in \Phi$, then $p \notin \Phi$.
ii) **false** $\notin \Phi$.

i.e. Φ is proper if it contains no propositional contradictions.

Definition 3.3 A set of temporal formulae $\Phi \in 2^{\mathcal{TL}}$ is said to be *downward closed* iff it satisfies

i) Φ contains no α-formulae;
ii) if $\varphi \in \Phi$ is a β-formula, with β-components φ_1 and φ_2 then

$$\alpha^*(\varphi_1) \subseteq \Phi \text{ or } \alpha^*(\varphi_2) \subseteq \Phi$$

For example, if $p, q \in$ PROP, the sets $\{\Diamond p \vee \Diamond q, \Diamond q, q\}$ and $\{p\ \mathcal{U}\ q, p, \bigcirc(p\ \mathcal{U}\ q)\}$ are downward closed, but $\{\Diamond p \vee \Diamond q, \Diamond q, p\}$ is not, since it contains neither β-component of $\Diamond q$.

Given a $\Phi \in 2^{\mathcal{TL}}$ containing no α-formulae there are many downward closed sets containing Φ. We now introduce the idea of a *downward closure* of a set Φ, a downward closed set containing Φ which is in some sense minimal.

Definition 3.4 If $\Psi, \Psi_1 \in 2^{\mathcal{TL}}$, and $\Psi_1 \subseteq \Psi$ then $\varphi \in \Psi$ is *necessary* for Ψ_1 if *one* of the following conditions holds:-

i) $\varphi \in \Psi_1$;
ii) there is a β-formula ψ which is necessary for Ψ_1 (with components ψ_1 and ψ_2) and either
  a) $\varphi \in \alpha^*(\psi_1) \cap \alpha^*(\psi_2)$.
  or b) $\varphi \in \alpha^*(\psi_1) \subseteq \Psi$ and $\alpha^*(\psi_2) \not\subseteq \Psi$
  or c) $\varphi \in \alpha^*(\psi_2) \subseteq \Psi$ and $\alpha^*(\psi_1) \not\subseteq \Psi$
iii) there is a β-formula ψ which is necessary for Ψ_1 with eventuality ψ_1 and

$$\varphi \in \alpha^*(\psi_1) \subseteq \Psi.$$

Definition 3.5 An set Φ_1 is a *downward closure* of a set Φ iff

i) $\alpha^*(\Phi) \subseteq \Phi_1$;
ii) Φ_1 is downward closed;
iii) all formulae in Φ_1 are necessary for $\alpha^*(\Phi)$

For example, if $\Phi = \{(\neg p \vee q), r\}$ where $p, q, r \in$ PROP then the downward closures of Φ are $\{(\neg p \vee q), \neg p, r\}$ and $\{(\neg p \vee q), q, r\}$
We denote by $DC(\Phi)$ the set of all downward closures of Φ, and by $PDC(\Phi)$ the set of downward closures of Φ which are also proper.

3.2.1 Construction of the downward closure

Given $\Phi \in 2^{\mathcal{TL}}$, the following algorithm will generate the set $PDC(\Phi)$.

i) If Φ contains α-formulae then replace Φ with $\alpha^*(\Phi)$.
ii) Set $D = \{\Phi\}$ and all β-formulae as unmarked.
iii) If $\Psi \in D$ and $\varphi \in \Psi$ is an unmarked β-formula with components φ_1 and φ_2 then replace Ψ in D by the sets $\Psi_1 = \Psi \cup \alpha^*(\varphi_1)$ and $\Psi_2 = \Psi \cup \alpha^*(\varphi_2)$ in which the formula φ is marked.
iv) Repeat (iii) until each set in D consists entirely of marked β-formulae, next-time formulae and literals.
v) Delete any sets from D which are not proper or contain formulae which are not necessary for $\alpha^*(\Phi)$.

On termination we have $D = PDC(\Phi)$.

(This process must terminate since all new β-formulae introduced at stage (iii) have lower degree than φ).

For example if $\Phi = \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p\}$ then stages (i)-(iv) of the above algorithm generate the set

$$
\begin{array}{lll}
D = & \{\{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, p\}, & \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, \bigcirc\Diamond p\}, & \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, \bigcirc\Diamond q, p\}, & ** \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, \bigcirc\Diamond q, \bigcirc\Diamond p\}, & ** \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, p, \bigcirc(p\,\mathcal{U}\,q), q\}, & ** \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, p, \bigcirc(p\,\mathcal{U}\,q), q, \bigcirc\Diamond p\}, & ** \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, p, \bigcirc(p\,\mathcal{U}\,q), \bigcirc\Diamond q\}, & \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, p, \bigcirc(p\,\mathcal{U}\,q), \bigcirc\Diamond q, \bigcirc\Diamond p\} & **\}
\end{array}
$$

but stage (v) deletes the starred sets to leave

$$
\begin{array}{ll}
PDC(\Phi) = & \{\{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, p\}, \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, \bigcirc\Diamond p\}, \\
& \{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, p, \bigcirc(p\,\mathcal{U}\,q), \bigcirc\Diamond q\}\}
\end{array}
$$

3.2.2 Constructing a $\mathcal{D}_\varphi$ structure

Given a formula φ, we define a structure $\mathcal{D}_\varphi = \langle X, N, FR, X_0, X_d\rangle$ where

X is a set of proper downward closed sets,
N is a next-time relation on X,
FR is a fulfillment relation,
X_0 is a set of initial states, and
X_d is a set of "dead" states, i.e. have no successors.

which has the following important property.

Lemma 3.2 *φ has a model if and only if, in the structure $\mathcal{D}_\varphi$, either*

a) $X_d \neq \emptyset$, or
b) there is a sequence (x_i) such that
  i) $x_o \in X_0$,
  ii) $x_i N x_{i+1}$ and
  iii) $\forall i.$ if $x_i \in \mathbf{dom}\,FR$ then $\exists sx.\ x_i FR\, sx \wedge (\forall x' \in sx.\ \exists j.\ j > i \wedge x' = x_j)$
  i.e. required eventualities of φ occur in the sequence.

This result provides the basis of a decision mechanism for temporal logic. We now define the components of the structure $\mathcal{D}_\varphi$. First, define a function $NT: 2^{\mathcal{TL}} \to 2^{\mathcal{TL}}$ by

$$NT(\Phi) = \{\psi \mid \bigcirc\psi \in \Phi\}$$

Then, given a $\mathcal{TL}$ formula φ, we define a set X by

i) X is a proper, downward closed set.
ii) $PDC(\{\varphi\}) \subseteq X$
iii) $\forall\Phi \in X.\, NT(\Phi) \neq \emptyset \Rightarrow PDC(NT(\Phi)) \subseteq X$
iv) X is the smallest set satisfying (i),(ii) and (iii).

Define a relation N on X by

$$\Phi N \Psi \quad \text{iff} \quad \Psi \in PDC(NT(\Phi))$$

Let $ev(\Phi)$ be the set

$$\{(\psi, \xi) \mid \psi \in \Phi, \psi \text{ has an eventuality } \xi \text{ and } \alpha^*(\xi) \subseteq \Phi\}$$

Given Φ, a subset Π of X is called a *satisfying set* for Φ if and only if

$$\forall(\psi, \xi) \in ev(\Phi).\, \exists\Psi \in \Pi.\, (\{\psi\} \cup \alpha^*(\xi)) \subseteq \Psi\}$$

Define $FR \subseteq X \times 2^X$ by $(\Phi, \Pi) \in FR$ if and only if

i) $ev(\Phi) \neq \emptyset$;
ii) either a) $\Pi = \emptyset$ and Φ has no satisfying set in X, or
b) Π is a satisfying set for Φ and no proper subset of Π is a satisfying set for Φ.

The definition of $\mathcal{D}_\varphi$ is completed by setting $X_0 = PDC(\{\phi\})$ and $X_d = \{\Phi \in X \mid NT(\Phi) = \emptyset\}$
Note that although the relation FR is potentially quite large, in practice it is never explicitly constructed since, given $\Phi \in X$, $\Pi \in 2^X$, it is relatively easy to determine whether or not $\Phi FR \Pi$.

Example If Φ is defined as in the previous example, then

$$\begin{aligned} PDC(\Phi) = \{&\{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, p\}, && \Phi_1 \\ &\{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, q, \bigcirc\Diamond p\}, && \Phi_2 \\ &\{p\,\mathcal{U}\,q, \Diamond q, \Diamond p, p, \bigcirc(p\,\mathcal{U}\,q), \bigcirc\Diamond q\}\} && \Phi_3 \end{aligned}$$

The next-time formulae of the elements of Φ are as follows:

$$\begin{aligned} NT(\Phi_1) &= \emptyset \\ NT(\Phi_2) &= \{\Diamond p\} \\ NT(\Phi_3) &= \{p\,\mathcal{U}\,q, \Diamond q\} \end{aligned}$$

The downward closed sets for these are

$$\begin{aligned} PDC(NT(\Phi_1)) &= \{\emptyset\} \\ PDC(NT(\Phi_2)) &= \{\{\Diamond p, p\}, && \Phi_{21} \\ &\quad \{\Diamond p, \bigcirc\Diamond p\}\} && \Phi_{22} \\ PDC(NT(\Phi_3)) &= \{\{p\,\mathcal{U}\,q, \Diamond q, q\}, && \Phi_{31} \\ &\quad \{p\,\mathcal{U}\,q, \Diamond q, p, \bigcirc(p\,\mathcal{U}\,q), \bigcirc\Diamond q\}\}, && \Phi_{32} \end{aligned}$$

This gives rise to the structure $\mathcal{D}_\varphi = \langle X, N, FR, X_0\rangle$ where

$$X = \{\Phi_1, \Phi_2, \Phi_3, \Phi_{21}, \Phi_{22}, \Phi_{31}, \Phi_{32}\}$$

$$N = \{(\Phi_2, \Phi_{21}), (\Phi_2, \Phi_{22}), (\Phi_3, \Phi_{31}), (\Phi_3, \Phi_{32}), (\Phi_{22}, \Phi_{21}), (\Phi_{22}, \Phi_{22}), (\Phi_{32}, \Phi_{31}), (\Phi_{32}, \Phi_{32})\}$$

$$FR = \{(\Phi_2, \{\Phi_1\}), (\Phi_2, \{\Phi_3\}), (\Phi_2, \{\Phi_{21}\}), (\Phi_2, \{\Phi_{32}\}), (\Phi_3, \{\Phi_1\}), (\Phi_3, \{\Phi_2\}), (\Phi_3, \{\Phi_{31}\}), (\Phi_{22}, \{\Phi_1\}), (\Phi_{22}, \{\Phi_3\}), (\Phi_{22}, \{\Phi_{21}\}), (\Phi_{22}, \{\Phi_{32}\}), (\Phi_{32}, \{\Phi_1\}), (\Phi_{32}, \{\Phi_2\}), (\Phi_{32}, \{\Phi_{31}\})\}$$

$$X_0 = \{\Phi_1, \Phi_2, \Phi_3\}$$

Given $\mathcal{D}_\varphi = \langle X, N, FR, X_0, X_d\rangle$ and $\mathcal{A}_F = \langle S, L, T, V, F, S_0\rangle$ we define the set $C \subseteq S \times X$ of 'contradictory' states by

$$(s, \Phi) \in C \text{ iff } \Phi \cup \{p \mid p \in V(s)\} \cup \{\neg q \mid q \in (\text{PROP} - V(s))\} \text{ is } \textit{not} \text{ proper.}$$

3.3 Combining the $\mathcal{A}_F$ and $\mathcal{D}_\varphi$ Structures

We now describe the final structure, a product of $\mathcal{A}_F$ and $\mathcal{D}_\varphi$.

Definition 3.6 Given a fair structure $\mathcal{A}_F$, where $\mathcal{A}_F$ is a tuple $\langle S, L, T, V, F, S_0\rangle$, a structure $\mathcal{D}_\varphi$, where $\mathcal{D}_\varphi$ is a tuple $\langle X, N, FR, X_0, X_d\rangle$, and a set $C \subseteq S \times X$, we can define a combined structure $\mathcal{A}_F^{\mathcal{D}}$ with respect to C as a tuple $\langle Y, R, \mathcal{A}_F, Y_0, Y_d\rangle$ where

P_0 is $\{ ((s,x), ls, (s',x')) \in (S \times X) \times 2^L \times (S \times X) \mid (s, ls, s') \in T \text{ and } (x,x') \in N \}$

P_1 is $\{ ((s,x), ls, (s',x)) \in (S \times X) \times 2^L \times (S \times X) \mid (s, ls, s') \in T \text{ and } x \in X_d \}$

P is P_0 if F contains no impartiality constraints, and $P_0 \cup P_1$ otherwise

P' is the projection of P onto $(S \times X) \times (S \times X)$

P'^* is the reflexive transitive closure of P'

Y is the set defined by

(i) $(S_0 \times X_0) - C \subseteq Y$

(ii) $Y \cap C = \emptyset$

(iii) $\forall (s,x) \in Y, s' \in S, x' \in X.$
$(s,x) P' (s',x') \Rightarrow \exists s'' \in S, x'' \in X. (s'',x'') \in Y \wedge (s,x) P' (s'',x'')$

(iv) $\forall (s,x) \in Y . \exists (s_0, x_0) \in (S_0 \times X_0) . (s_0, x_0) P'^* (s,x)$

Y_0 is the set of initial states, $(S_0 \times X_0) \cap Y$

Y_d is the set of dead states, $\{(s,x) \in Y \mid (s,x) \text{ has no } P' \text{ successors}\}$

R is $P \cap (Y \times 2^L \times Y)$

The combination is essentially the product of $\mathcal{A}_F$ and $\mathcal{D}_\varphi$, under the restrictions that no contradictory states exist (ii), no next-time inconsistent states are allowed (iii), and that all states are rooted in some initial state (iv). Optimisation of the P relation occurs if no impartiality constraints occur in $\mathcal{A}_F$ and relies on the result of lemma 2.1. The labelling from $\mathcal{A}_F$ is carried through to $\mathcal{A}_F^{\mathcal{D}}$.

Definition 3.7 A *fulfilling* path through $\mathcal{A}_F^{\mathcal{D}}$ is a sequence $((s_i, x_i))$ such that

$$
\begin{array}{l}
(s_0, x_0) \in Y_0 \wedge \\
\forall i.\ \exists ls.\ ((s_i, x_i), ls, (s_{i+1}, x_{i+1})) \in R \wedge \\
\forall i.\ \text{if } x_i \in \mathbf{dom} FR \\
\quad \text{then } \exists sx.\ x_i\, FR\, sx \wedge (\forall x' \in sx.\ \exists j.\ j > i \wedge x' = x_j)
\end{array}
$$

Definition 3.8 A *fair* path through the structure $\mathcal{A}_F^{\mathcal{D}}$ is a fulfilling path $((s_i, x_i))$ of $\mathcal{A}_F^{\mathcal{D}}$ such that the sequence (s_i) is a fair path of $\mathcal{A}_F$.

Definition 3.9 A *admissible* path through the structure $\mathcal{A}_F^{\mathcal{D}}$ is either a fair path of $\mathcal{A}_F^{\mathcal{D}}$ or a path starting from an initial state and leading to a dead state of $\mathcal{A}_F^{\mathcal{D}}$.

Theorem 1 Given $\mathcal{A}_F$ and a $\mathcal{D}_\varphi$ structure constructed from φ and a C constructed from $\mathcal{A}_F$, $\mathcal{D}_\varphi$, and φ as above, then

1. If $\langle \mathcal{A}_F, s \rangle \models_{\mathcal{F}} \varphi$ for some $s \in S_0$ then there exists an admissible path of $\mathcal{A}_F^{\mathcal{D}}$,
2. If there exists an admissible path of $\mathcal{A}_F^{\mathcal{D}}$ then $\langle \mathcal{A}_F, s \rangle \models_{\mathcal{F}} \varphi$ for some $s \in S_0$.

3.4 Checking for Fair Paths

We now turn our attention to the algorithm for detecting a fair path in a structure $\mathcal{A}_F^{\mathcal{D}}$. The technique is based on the approach described by Lichtenstein and Pnueli [LP85] (the LP algorithm) and relies on detecting "fair" maximal strongly connected components (MSCCs). The modifications needed are to cope with the contracted edge-labelled fair structures we generate and, in particular, to cope with handling simultaneous application of non-global impartiality, justness and fairness constraints on a single path.

First, recall from Section 3.2.2 that if $\mathcal{D}_\varphi$ contains a dead state, then φ has a model (the paths leading to the dead state will characterise prefixes of models satisfying φ). A dead state of $\mathcal{D}_\varphi$ may be carried through to $\mathcal{A}_F^{\mathcal{D}}$ when no impartiality constraints exist, so we first check $\mathcal{A}_F^{\mathcal{D}}$ for containment of dead states before looking for fair paths[2].

Clearly, if a fair path exists in the structure $\mathcal{A}_F^{\mathcal{D}}$, it must have a tail contained in a strongly connected subcomponent (SCS) of $\mathcal{A}_F^{\mathcal{D}}$. It is prudent to check terminal MSCCs for containment of a fair path, i.e. containment of a fair SCS; if one is found, then our task is done and we will have demonstrated the existence of a fair path of $\mathcal{A}_F^{\mathcal{D}}$ (and hence a fair path of $\mathcal{A}_F$ satisfying the given formula φ).

Let us define some of these notions more carefully.[3] The structure $\mathcal{A}_F^{\mathcal{D}}$ will contain ω-paths that are not necessarily fulfilling, corresponding to ω-paths with unfulfilled temporal eventualities. Remember, the construction of $\mathcal{A}_F^{\mathcal{D}}$ only guarantees propositional and next-time consistency. Hence, in checking an SCS for a fair path we must first check that it is fulfilled.

Definition 3.10 An SCS $\mathcal{B}$ of $\mathcal{A}_F^{\mathcal{D}}$ is *fulfilled* **iff**

$$\forall (s,x) \in \mathcal{B}.\ x \in \mathbf{dom} FR \Rightarrow \exists \mathcal{B}' \subseteq \mathcal{B}.\ x\, FR\, \{x' \mid \exists s'.\ (s', x') \in \mathcal{B}'\}$$

Lemma 3.3 *Given a SCS $\mathcal{B}$ which is fulfilled, then any ω-path in $\mathcal{B}$ that is not fulfilled can be expanded, by taking suitable detours, to a ω-path of $\mathcal{B}$ that is fulfilled.*

Lemma 3.4 *Given a fulfilled SCS $\mathcal{B}$ of $\mathcal{A}_F^{\mathcal{D}}$ and an SCS $\mathcal{B}_1$, if $\mathcal{B} \subseteq \mathcal{B}_1$, then $\mathcal{B}_1$ is a fulfilled SCS.*

It should be noted that if $\mathcal{B}_1 \subseteq \mathcal{B}$ then $\mathcal{B}_1$ need not be fulfilled.

[2] Of course, if φ, in its normalised form, does not contain $\Box$ or $\mathcal{W}$ operators, then φ will only have a model if $\mathcal{D}_\varphi$ has a dead state.

[3] We assume the notions of SCS, terminal SCS, MSCC and terminal MSCC are well understood.

Definition 3.11 Given a set of states E, and a set of labels M, an SCS $\mathcal{B}$ of $\mathcal{A}_F^{\mathcal{D}}$ is called $\langle E,M\rangle$-*labelled* iff every $m \in M$ labels some edge of $\mathcal{B}$ from an E-state. More precisely,

$$\forall m \in M.\ \exists (s,x),\ (s',x') \in \mathcal{B},\ ls \in 2^L.\ ((s,x),ls,(s',x')) \in R \wedge m \in ls \wedge s \in E$$

Definition 3.12 For a fairness constraint $f = \langle E,M,type\rangle$, an SCS $\mathcal{B}$ of $\mathcal{A}_F^{\mathcal{D}}$ is an f-*SCS* **iff**

$\mathcal{B}$ is fulfilled and
either $\mathcal{B}$ contains no E-states
or
type = IMPARTIAL:
$\mathcal{B}$ is $\langle E,M\rangle$-labelled
type = JUST:
either there is a nonempty $M' \subseteq M$ that is disabled[4]
for some s where $(s,x) \in B$ and $s \in E$
or $\mathcal{B}$ is $\langle E,M\rangle$-labelled
type = FAIR:
for every $m \in M$
either $\{m\}$ is disabled for those s where $(s,x) \in B$ and $s \in E$
or $\mathcal{B}$ is $\langle E,\{m\}\rangle$-labelled

Definition 3.13 Given a set of fairness constraints G, an SCS $\mathcal{B}$ is *fair* w.r.t. G **iff** $\mathcal{B}$ is an f-SCS for every fairness constraint $f \in G$.

An immediate consequence of the above definitions is

Lemma 3.5 *A structure $\mathcal{A}_F^{\mathcal{D}}$ contains an admissible path* **iff** *$\mathcal{A}_F^{\mathcal{D}}$ contains either*

i) an SCS $\mathcal{B}$ that is fair w.r.t. F of $\mathcal{A}_F^{\mathcal{D}}$, or
ii) a dead state.

Lemma 3.4 asserted that the property of fulfillment of SCSs is monotone w.r.t. $\subseteq$. In [LP85], where only global fairness constraints are considered, the property of impartiality and justness of SCSs is also shown to be monotone w.r.t. $\subseteq$; these monotonicity properties enable one to check only MSCCs for impartiality and justness. Unfortunately, the monotonicity property does not hold for our non-global, or relativised, fairness constraints. Consider the following example. Let $\mathcal{B}_1, \mathcal{B}_2$ be SCSs such that $\mathcal{B}_1 \subseteq \mathcal{B}_2$. Take a fairness constraint $f = \langle E,M,type\rangle$ such that $\mathcal{B}_1$ contains no E-states. It therefore follows by Definition 3.11 that $\mathcal{B}_1$ is $\langle E,M\rangle$-labelled. Suppose also that E-states *do* occur in $\mathcal{B}_2$ but that some $m \in M$ does not label any edge from an E-state; it then follows that $\mathcal{B}_2$ is *not* $\langle E,M\rangle$-labelled and thus contradicts monotonicity. The relativisation of fairness constraints thus poses problems, but not insurmountable ones; a weaker monotonicity result holds.

Definition 3.14 An f-SCS $\mathcal{B}$ for constraint $f = \langle E,M,type\rangle$ is said to be *trivially* fair, if no E-states occur in the SCS $\mathcal{B}$, and *non-trivially* fair otherwise.

Lemma 3.6 *Given an* IMPARTIAL *or* JUST *fairness constraint f and a non-trivial f-SCS $\mathcal{B}_1$, an SCS $\mathcal{B}_2$ containing $\mathcal{B}_1$ will also be a non-trivial f-SCS.*

The above result generalises in the obvious way to SCSs non-trivially fair w.r.t. a set of IMPARTIAL or JUST fairness constraints.

Disregarding temporarily both FAIR constraints and trivial satisfaction of fairness, the monotonicity result for non-trivial SCSs allows one to check whether some MSCC is non-trivially fair w.r.t. to a set of IMPARTIAL and JUST constraints in a piecemeal fashion.

[4]Determined from $\mathcal{A}_F$, see section 2.3.

Lemma 3.7 *Given sets G_1 and G_2 of* IMPARTIAL *or* JUST *constraints and an MSCC $\mathcal{B}$ containing an SCS non-trivially fair w.r.t. G_1 and an SCS non-trivially fair w.r.t. G_2, then $\mathcal{B}$ is non-trivially fair w.r.t. $G_1 \cup G_2$.*

We present below an algorithm for checking whether a structure $\mathcal{A}_F^{\mathcal{D}}$ contains an SCS that is fair w.r.t. the set F of $\mathcal{A}_F^{\mathcal{D}}$. The strategy applied to handle FAIR constraints is similar to that presented in [LP85] in that a given MSCC is recursively decomposed until a contained SCS is found that satisfies all the fairness conditions. However, they are many differences due to having to check non-global, arbitrary, fairness constraints.

The major work is done by the procedure FAIR. To check for an SCS satisfying all fairness constraints, we first attempt to find an SCS satisfying all the given FAIR constraints. Each FAIR constraint is checked in turn; however, as soon as the SCS being checked is reduced to a smaller SCS, it is necessary to recheck all previously checked FAIR fairness constraints (this is because of the non-monotonicity of FAIR fairness). When the procedure has successfully found an SCS which is fair w.r.t. the set of FAIR constraints, it checks that particular SCS, say $\mathcal{B}$, for the remaining, IMPARTIAL and JUST constraints. This check is done in two stages. First we check to see whether the SCS is non-trivially fair; if it is then we are done. If the SCS is not non-trivially fair (which, of course, does not mean that is trivially fair) then the SCS is decomposed to smaller SCSs. This is done by deleting, from $\mathcal{B}$, the E-states of the constraints that were not non-trivially satisfied, and then decomposing the remaining graph into MSCCs. As in the decomposition for FAIR constraints, if no MSCCs remain, then the algorithm returns false, otherwise all fairness conditions, including the previously checked FAIR constraints, must be rechecked on the smaller SCSs.

function FAIR-SAT($\mathcal{A}_F^{\mathcal{D}}$): **Boolean** ;
{ This function determines whether the structure $\mathcal{A}_F^{\mathcal{D}}$ contains a fair path.}
begin
  if $Y_d \neq \emptyset$ { which implies there are no IMPARTIAL-conditions of $\mathcal{A}_F$}
  then return true ;
  let U = the set of FAIR-conditions of $\mathcal{A}_F$;
  let O = the set of JUST and IMPARTIAL-conditions of $\mathcal{A}_F$;
  Decompose $\mathcal{A}_F^{\mathcal{D}}$ into fulfilled MSCCs;
  return there is a fulfilled MSCC $\mathcal{B}$ such that FAIR ($\mathcal{B}$, U, $\emptyset$, O);
end { FAIR-SAT }

procedure FAIR ($\mathcal{B}$, U, C, O): **Boolean** ;
{ $\mathcal{B}$ is a strongly connected subgraph,
  U is a set of unchecked strong fairness conditions
  C is a set of checked strong fairness conditions
  O is a set of unchecked impartiality and justness conditions
  The function checks that $\mathcal{B}$ contains an SCS which simultaneously satisfies
  the given conditions}
begin
  if $U = \emptyset$ **then**
    $FO := \emptyset$;
    for every o $\in$ O **do**
      if $\mathcal{B}$ is not an o-SCS, **then** $FO := FO \cup \{o\}$;
    if $FO = \emptyset$ **then return true** ;
    let $N = \bigcup\{E \mid \langle E, M, type\rangle \in FO\}$;
    let $\mathcal{B}^N$ = subgraph obtained by deleting the N-states from $\mathcal{B}$;
    if $\mathcal{B}^N = \emptyset$ **then return false** ;
    Decompose $\mathcal{B}^N$ into fulfilled MSCCs $\mathcal{B}_1 \ldots \mathcal{B}_n$;
    for every $i \in \{1..n\}$ **do**

```
        if FAIR(B_i, C, ∅, O − FO) then return true ;
    return false ;
  let u = ⟨E, M⟩ be an element of U;
  if B is u-impartial then return FAIR(B, U − {u}, C ∪ {u}, O);
  let K = {l ∈ M | B is not ⟨E, {l}⟩-impartial };
  let B^K be the subgraph obtained from B by deleting all states
      s ∈ (E "∩" B) such that enabled(s) ∩ K ≠ ∅;
  if B^K = ∅ then return false ;
  Decompose B^K into fulfilled MSCCs B_1 ... B_n
  for every i ∈ {1..n} do
      if FAIR(B_i, C ∪ (U − {u}) ∪ {⟨E, M − K⟩}, ∅, O)
      then return true ;
  return false ;
end { FAIR }

function FULFILLED(B): Boolean ;
{ This function determines whether the SCS B is fulfilled. }
end { FULFILLED }
```

Lemma 3.8 *Given a structure $\mathcal{A}_F^{\mathcal{D}}$, FAIR-SAT($\mathcal{A}_F^{\mathcal{D}}$) terminates.*

Proof (outline): Since $\mathcal{A}_F^{\mathcal{D}}$ is decomposed into fulfilled MSCCs, FAIR-SAT($\mathcal{A}_F^{\mathcal{D}}$) terminates if the procedure FAIR() terminates when given any such MSCC,$\mathcal{B}$.

In the procedure FAIR(), either $\mathcal{B}$ satisfies all the required fairness conditions (and thus the procedure terminates) or there is at least one unsatisfied fairness constraint. In this case the states in $\mathcal{B}$ that caused the unsatisfied constraints to be enabled are then removed and FAIR() is called again with a reduced set of fairness constraints and any fulfilled MSCC that is part of the remaining structure. Thus, either an SCS satisfying the fairness constraints will be found, or the set of states considered will be reduced. In either case the procedure terminates.

Lemma 3.9 *Given a structure $\mathcal{A}_F^{\mathcal{D}}$, if FAIR-SAT($\mathcal{A}_F^{\mathcal{D}}$) returns true then either*

i) $\mathcal{A}_F^{\mathcal{D}}$ contains an SCS $\mathcal{B}$ which is fair w.r.t. the fairness constraints F of $\mathcal{A}_F^{\mathcal{D}}$, or
ii) $\mathcal{A}_F^{\mathcal{D}}$ contains a dead state.

Proof (outline): This result can be seen by looking at the points in the procedures FAIR-SAT() and FAIR() where true can be returned. Note that if true is returned from FAIR() for an SCS $\mathcal{B}$, then $\mathcal{B}$ must satisfy *all* the remaining fairness constraints.

Lemma 3.10 *Given a structure $\mathcal{A}_F^{\mathcal{D}}$ then*

i) if it contains an SCS which is fair w.r.t. the fairness constraints F of $\mathcal{A}_F^{\mathcal{D}}$, FAIR-SAT($\mathcal{A}_F^{\mathcal{D}}$) returns true.
ii) if it contains a dead state, FAIR-SAT($\mathcal{A}_F^{\mathcal{D}}$) returns true.

Proof (outline): For any structure $\mathcal{A}_F^{\mathcal{D}}$, (ii) is obviously true. If $\mathcal{A}_F^{\mathcal{D}}$ contains an SCS, $\mathcal{B}$, that is fair with respect to the fairness constraints then $\mathcal{B}$ must be contained in an MSCC $\mathcal{B}_1$. Unless some other fair SCS has been found, the algorithm will eventually examine $\mathcal{B}_1$. Once this occurs, since the only states discarded are those that enable unsatisfiable fairness constraints, the decomposition will guarantee that $\mathcal{B}$ is found. From the above Lemmas we can immediately deduce

Theorem 2 *FAIR-SAT($\mathcal{A}_F^{\mathcal{D}}$) is true if and only there is an admissible path through $\mathcal{A}_F^{\mathcal{D}}$.*

The above, together with Theorem 1, establishes the desired result, namely:

Theorem 3 *Given a structure $\mathcal{A}_F$, a temporal formula φ and a structure $\mathcal{A}_F^{\mathcal{D}}$ built from $\mathcal{A}_F$ and $\mathcal{D}_\varphi$, then*

$$\langle \mathcal{A}_F, s \rangle \models_{\mathcal{F}} \varphi \quad \textit{for some } s \in S_0 \textbf{ iff } \textit{FAIR-SAT}(\mathcal{A}_F^{\mathcal{D}}).$$

References

[Bro86] Michael C. Browne.
An Improved Algorithm for the Automatic Verification of Finite State Systems using Temporal Logic.
Technical report, Department of Computer Science, Carnegie Mellon University, December 1986.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla.
Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.
ACM Transactions on Programming Languages and Systems, 8(2):244–263, 1986.

[CVW86] Constantin Courcoubetis, Moshe Y. Vardi, and Pierre Wolper.
Reasoning About Fair Concurrent Programs.
Technical Report RJ 5149 (53489), IBM Almaden Research Center, San Jose, California, May 1986.

[Fis89] Michael D. Fisher.
A Model Checker for Linear Time Temporal Logic.
Temple project report, Department of Computer Science, University of Manchester, February 1989.

[GB88] G. D. Gough and H. Barringer.
A Semantics Driven Temporal Verification System.
In *Proceedings of ESOP '88*, March 1988.

[GB89] G. D. Gough and H. Barringer.
Mechanisation of Temporal Logics. Part 2: Practical Decision Procedures.
Temple project report, Department of Computer Science, University of Manchester, 1989. (forthcoming).

[Gou84] G. D. Gough.
Decision Procedures for Temporal Logic.
Master's thesis, Department of Computer Science, University of Manchester, October 1984.

[Joh79] Stephen C. Johnson.
YACC: Yet another compiler-compiler.
Unix Programmer's Manual Vol 2b, 1979.

[LP85] O. Lichtenstein and A. Pnueli.
Checking that Finite State Concurrent Programs Satisfy their Linear Specification.
In *Proceedings of the Twelfth ACM Symposium on the Principles of Programming Languages*, New Orleans, Louisiana, January 1985.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi.
Impartiality, Justice and Fairness: The Ethics of Concurrent Termination.
Lecture Notes in Computer Science, 115, July 1981.

[Plo81] G. D. Plotkin.
A Structural Approach to Operational Semantics.
Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, September 1981.

[RRSV87] J.L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron.
Xesar User's Manual.
Technical report, Laboratoire de Genie Informatique, IMAG, Grenoble, 1987.