Formal Description of Programming Concepts - II
D. Bjørner (ed.)
North-Holland Publishing Company

FAIRNESS ASSUMPTIONS FOR CSP IN A TEMPORAL LOGIC FRAMEWORK

R. Kuiper
Mathematical Centre
Kruislaan 413
Amsterdam

W.P. de Roever
Department of Computer Science
University of Utrecht
Princetonplein 5
Utrecht

Six fairness assumptions for the repetitive construct $*[\ldots \Box b_\ell , c_\ell \rightarrow S_\ell \Box \ldots]$ in a subset of CSP are given and classified with respect to the programs they cause to terminate. A total correctness proof system for the subset of CSP is given, incorporating the different fairness assumptions.

0. INTRODUCTION

The research in this paper originated from work by FRANCEZ AND DE ROEVER [F de R]. The aim of the paper is twofold, both cases having to do with temporal logic. On the one hand, we consider six different fairness assumptions for a subset of CSP, i.e. Communicating Sequential Processes, a language for distributed computing without shared variables defined by HOARE in [H]. These assumptions will be expressed using temporal logic, which enables us to formulate them at a level convenient for intuitive understanding of their meaning as well as for use in formal proofs. They will be compared with respect to the sets of programs they cause to terminate. On the other hand we need a framework to reason about the effects of such fairness assumptions. To do so we give a (low level) temporal logic proof system for this subset of CSP. We use the idea of temporal semantics as developed for shared variable languages by PNUELI [P]. We have been helped by BEN ARI's thesis [BA], especially by his way of reasoning with conditional invariants. It is shown here that by this method also non-shared variables and synchronized communication as in CSP can be modelled in a natural way.

The set up is as follows. Section I gives the preliminary facts of CSP, section 2 the temporal logic semantics and section 3 the fairness assumptions; section 4 indicates the temporal logic we use. In section 5 several examples are given. Finally section 6 contains discussion.

When this paper was being typed, we received a paper by SMOLKA [S] dealing with related matters.

I. PRELIMINARIES

The syntax of the subset of CSP we use is as follows.

The research reported in this paper originated from work by Francez and de Roever. Francez' stay at the University of Utrecht was supported by the Netherlands Organization for the advancement of Pure Research (Z.W.O), as was part of the research of de Roever in the form of numerous travel grants for collaborating with Francez at the Technion and Pnueli at the Weizmann Institute, both in Israel. De Roever's collaboration with Pnueli was partly supported by the Department of Applied Mathematics of the Weizmann Institute of Science.

DEFINITION

Statements: $S ::= skip \mid x:=t \mid *[b_1,c_1 \to S_1 \,\square \ldots \square b_m,c_m \to S_m] \mid S_1;S_2$
where t is an integer expression
b a boolean expression and
c either $P_i!x$ or $P_j?y$ $\quad i,j \in \{1,\ldots n\}$

Programs : $[P_1::S_1 \parallel \ldots \parallel P_n::S_n]$
where $P_i, i \in I = \{1,\ldots,n\}$, is called a process.
Processes have no shared variables.

Neither [...‖...] nor *[...] is allowed to be used in nested fashion.

2. TEMPORAL SEMANTICS

We introduce control locations ℓ_i, ℓ_i', $i \in I$, as follows. ℓ_i (or ℓ_i') can be at S or after S for S in P_i defined in the natural way (cf. [0],[0L]). Obvious identifications like: "for $P_1::S_1;S_2$ we have after $S_1 \equiv$ at S_2 and at $P_1 \equiv$ at $S_1;S_2 \equiv$ at S_1" are made. The guarded command case needs some further clarification:
1) For S_ℓ in $*[\ldots \square b_\ell\, c_\ell \to S_\ell \,\square \ldots]$, after $S_\ell \equiv$ at $*[\ldots]$.
2) There are no control locations concerning the b_ℓ, c_ℓ construct, as, when control is active at a guarded command *[ ], all guards are evaluated at the same time instant, after which control is still at the same point or resides either at one of the guarded statements or after the whole command.

States S are tuples $S = \langle \ell,s \rangle = \langle\langle \ell_1,\sigma_1 \rangle, \ldots, \langle \ell_n,\sigma_n \rangle\rangle$ such that for each $i \in I$ ℓ_i is one of the above defined control locations in P_i. Control locations are also used as predicates, $\overline{\ell}_i$ (or $\overline{\ell}_i'$) being true in $s = (\ell,\sigma)$ iff $\ell_i = \overline{\ell}_i$ (respectively $\ell_i' = \overline{\ell_i'}$).

Auxiliary notation:
*[i] denotes a guarded command in P_i; constructs like "for all *[i] in P_i" assume implicit indexing of the *[i]. $g_{i\ell} \equiv b_{i\ell},c_{i\ell}$ is a guard in a guarded command $*[\ldots \square b_{i\ell},c_{i\ell} \to S_{i\ell} \,\square \ldots]$ belonging to the process P_i.
$c_{i\ell} \underline{m}\, c_{jm}$ iff $c_{i\ell}$ and c_{jm} are syntactically matching communication commands (e.g.: $P_i!x$ in P_i and $P_j?y$ in P_j). $g_{i\ell}$ in the guarded command *[i] is true in the state s iff there is a process P_j such that ℓ_j = at *[j] and *[j] contains at least one g_{jm} such that $c_{i\ell} \underline{m}\, c_{jm} \wedge b_{i\ell} \wedge b_{jm}$. Notation $g_{i\ell} \underline{m}\, g_{jm}$. This indicates semantical matching. $\sigma[i\ell \underline{c}\, jm]$ is σ changed according to the effect of the communication between c_{jm} and c_{jm} (e.g.: $g_{i\ell} = P_j!x$ and $g_{jm} = P_i?y$ will lead to $\sigma[il \underline{c}\, jm] = \sigma[x/y]$).
Finally, to enable us to include the distributed termination convention we define: $t(g_{i\ell})$ holds in s iff the process named (as target) in $c_{i\ell}$ is terminated (e.g. ℓ_j = after P_j and $g_{i\ell} = b_{i\ell},P_j?x$).

Now we define the temporal semantics as follows. The meaning of a program is the set of computation sequences satisfying the following axioms. $\bigcirc$ is the next time operator from temporal logic.

Exclusivity Axiom (E)
$\neg(\ell_i \wedge \ell_i')$ for all $i \in I$ and $\ell_i \neq \ell_i'$.

The exclusivity axiom describes that control in each process always is at just one place at the same time.

Local Semantics Axiom (LS)
(i) at skip $\wedge\ \sigma = \overline{\sigma} \supset \bigcirc$ (at skip) $\vee \bigcirc$(after skip $\wedge\ \sigma = \overline{\sigma}$)
(ii) at x:= t $\wedge\ \sigma = \overline{\sigma} \supset \bigcirc$ (at x:=t) $\vee \bigcirc$(after x:=t $\wedge \sigma = \overline{\sigma}\,[t/x]$).
(iii) Let $*[i] = *[b_{i1},c_{i1} \to S_{i1} \,\square \ldots \square b_{in_i},c_{in_i} \to S_{in_i}]$

$$\text{at} * [i] \wedge \sigma = \bar{\sigma} \supset O\ (\text{at} * [i])$$

$$\bigvee_{\substack{j=1 \\ j \neq i}}^{n} \bigvee_{\ell=1}^{n_i} \bigvee_{m=1}^{n_j} (\quad g_{i\ell} \underline{m}\ g_{jm} \ \wedge$$

$$O(\text{at } S_{i\ell} \wedge \text{at } S_{jm} \wedge \sigma = \bar{\sigma}\ [i\ell \underline{c} jm]))$$

$$\bigwedge_{\ell=1}^{n_i} (\neg b_{i\ell} \vee t(g_{i\ell})) \wedge O\ (\text{after} * [i] \wedge \sigma = \bar{\sigma}))$$

The local semantics axiom describes what is usually known (in papers not dealing with fairness) as operational semantics of these constructs. Note, that synchronization and the termination convention of CSP come to the fore in (iii).

Now to state our last axiom we have to refine our notation such that each statement in the program has a unique name.
Enumerate the control locations in process P_i of form at S_k where $S_k \equiv$ skip or $S_k \equiv x := t$ by α_{ik}, $i \in I$, $K \in K_i$. Let α'_{ik} denote the corresponding after S_k location.
Likewise enumerate the control locations of form
at $*[\ldots \Box b_{iq\ell}, c_{iq\ell} \to S_{iq\ell} \Box \ldots]$ in process P_i by γ_{iq}, $i \in I, q \in Q_i$ with corresponding sets of locations

$$\Gamma_{iq} = V_\ell \text{ at } S_{iq\ell} \vee \text{after} * [\ldots], \ \ell \in L_{iq}$$

Then define

$$A_{ik} = \alpha_{ik} \wedge O\ \alpha'_{ik} \quad \text{for } i \in I, k \in K_i$$

$$C_{iq} = \gamma_{iq} \wedge O\ \Gamma_{iq} \quad \text{for } i \in I, q \in Q_i$$

$$T = \bigwedge_{i \in I} (\text{after } P_i \vee (\text{at} * [i] \wedge \bigwedge_\ell \neg g_{i\ell} \wedge \neg \bigwedge_\ell (\neg b_{i\ell} \vee t(g_{i\ell}))))$$

Notice, that A_{ik} and C_{iq} describe that a statement is activated, whereas T indicates that a situation is finished or blocked.
Now let b=0 (respectively 1) denote that b is false (respectively true). Then $\Sigma_{i \in I} b_i = 1$ indicates that exactly one of the b_i is true. Moreover, the execution of a guarded command by selecting a guard containing only the boolean part should be seen as a self-communication between two identical processes.
Then finally we state the

Multiprogramming Axiom (M)

$$\sum_{i \in I} \sum_{k \in k_i} A_{i_k} + \tfrac{1}{2} \sum_{i \in I} \sum_{q \in Q_i} C_{iq} + T = 1$$

The multiprogramming axiom describes that either the program is terminated or blocked (i.e.T=1) or exactly one action changing the state takes place at each time instant. Note, that communication between two processes is viewed as one action (cf.the factor $\frac{1}{2}$ in M).

REMARK. Above we require that, in not terminated or blocked situations, exactly one action is performed at each time instant. Concurrency then is described by considering all sequences of such actions allowed by the semantics; this is the usual treatment in case of concurrent shared variable languages. However, as in CSP the processes have no shared variables, it is more natural to allow atomic actions in different processes to be executed at the same time instant; the same also holds for communications between disjunct pairs of processes. The system can be adapted to this as follows. We now use that s is an n-tuple $\langle\langle \ell_1, \sigma_1 \rangle, \ldots, \langle \ell_n, \sigma_n \rangle\rangle$ where each process P_i only affects (ℓ_i, σ_i). Contrary to the situation above, we cannot assume anymore that only the active process determines the state at the next instant. Therefore we explicitly denote that if a process is not activated, it does not change its part of the state.
We now have :

Local Semantics Axiom * (LS^*)

(i) $\text{at skip} \wedge \sigma = \bar{\sigma} \supset 0\,(\text{at skip} \wedge \sigma_i = \bar{\sigma}_i) \vee 0\,(\text{after skip} \wedge \sigma_i = \bar{\sigma}_i)$ for skip in P_i

(ii) $\text{at } x:=t \wedge \sigma = \bar{\sigma} \supset 0\,(\text{at } x:=t \wedge \sigma_i = \bar{\sigma}_i) \vee 0\,(\text{after } n:=t \wedge \sigma_i = \sigma_i[t/x])$ for $n:=t$ in P_i

(iii) Let $*[i] = *[b_{i1}, c_{i1} \to S_{i1} \,\square \ldots \square b_{in_i}, c_{in_i} \to S_{in_i}]$

$$\text{at } *[i] \wedge \sigma = \bar{\sigma} \supset 0\,(\text{at } *[i] \wedge \sigma_i = \bar{\sigma}_i)$$

$$\vee\Big(\bigvee_{j=1}^{n} \bigvee_{\ell=1}^{n_i} \bigvee_{m=1}^{n_j} \big(g_{i\ell} \underline{m}\, g_{jm} \wedge 0(\text{at } S_{i\ell} \wedge \text{at } S_{jm} \wedge \sigma_i = \bar{\sigma}_i[i\ell \underline{c}\, jm] \wedge \sigma_j = \bar{\sigma}_j[i\ell \underline{c}\, jm])\big)\Big)$$

$$\vee\Big(\bigwedge_{\ell=1}^{n_i}(\neg b_{i\ell} \vee t(g_{i\ell}))\Big) \wedge 0\,(\text{after } *[i] \wedge \sigma_i = \bar{\sigma}_i))$$

Note, that the exclusivity axiom prevents executing more than one of the possible choices in case of a guarded command.

Multiprogramming Axiom * (M^*)

$$\sum_{i\in I} \sum_{k\in K_i} A_{ik} + \sum_{i\in I} \sum_{q\in Q_i} C_{iq} + T \geq 1$$

The further material in this paper can without change (up to *'s) be taken as based on either one of these alternatives.

3. FAIRNESS ASSUMPTIONS

Our aim is to define in the context of CSP a variety of intuitively reasonable fairness assumptions depending on different implementations of the guarded command construction (cf.[D]) as well as on synchronized communication, both being specific CSP features. We compare the different assumptions with respect to the programs they cause to terminate.

We start by considering what kind of fairness is induced by the temporal semantics so far. Note, that the multi-programming axiom (M) ensures that no unnecessary idling occurs; only a blocked or terminal state can (and always will) be repeated unchanged. (M) also ensures that as long as somewhere action is possible, some action will be taken, i.e. the temporal semantics so far imposes minimal liveness (cf.[OL]) . So

Minimal Liveness Axiom: —

Next, as in the presence of one process looping all the time this allows starvation of all other processes, it seems reasonable to impose a stronger liveness requirement. The usual one chosen is fundamental liveness (cf.[OL])ensuring that if a process is continuously enabled to proceed, it eventually will. To express this, we first give the usual axiom for atomic statements, using the temporal operators $\Diamond$ (eventually) and $\Box$ (always).

Atomic Statement Liveness Axiom (ASL)

$$\Box\,\text{at } S \supset \Diamond\,\text{after } S \quad \text{for } S \equiv \text{skip or } S \equiv x := t$$

We now are faced with treating the guarded command in the same way. If all boolean guards are false the axiom is obvious.

Guarded Command Skip Axiom (GCS)

$$\Box\,(\text{at } *[\,] \wedge \wedge_\ell(\neg b_\ell \vee t(g_\ell))) \supset \Diamond\,\text{after } *[\,]$$

Now to deal with enabled guarded commands there are various possibilities, depending on two parameters. Firstly, we consider two fairness assumptions: weak (respectively strong) fairness, stating that those moves which are eventually continuously (respectively eventually infinitely often) enabled are eventually taken (cf., e.g., [GPSS]). Secondly, in CSP we can distinguish three varieties of these two

assumptions, depending on what is taken to be a move in the case of executing guarded commands. As will become clear from the assumptions to follow, we can distinguish a move with respect to a process, a guard or a pair of semantically matching guards, i.e. a channel. Hence the concept of fundamental liveness is captured by requiring the following.

Fundamental Liveness Axiom
(i) Atomic Statement Liveness Axiom
(ii) Guarded Command Skip Axiom
(iii) $\Box$ at $*[\ ] \wedge \Diamond \Box$ (at $*[\ ] \supset V_\ell g_\ell$) $\supset \Diamond V_\ell$ at S_ℓ

As will be seen below, we shall concentrate on different possibilities for (iii), having the above one as the weakest possibility.

REMARK. In the axioms we use constructs like $\Box \Diamond$ at $*[\ldots] \supset \Diamond$ at S_ℓ and $\Box$ at $*[\ldots] \supset \Diamond$ at S_ℓ, which seem self-contradictory. As to the first one, this can eventually happen: $\Box \Diamond$ at $*[$true $\rightarrow S_\ell\ ] \supset \Diamond$ at S_ℓ, even $\Box \Diamond$ at S_ℓ is possible. As to the second one, the axiom is there to exclude all computation sequences for which $\Box$ at $*[\ldots]$ holds, so logically there is no contradiction: the axiom might be replaced by $\neg \Box$ at $*[\ldots]$. We have chosen the above representation as it covers all cases in a uniform way and indicates the next control location to be reached, thus providing intuition for the design of proofs.

We now formulate the fairness assumptions for the $*[\ldots \Box g_\ell \rightarrow S_\ell \Box \ldots]$ construct. When requerying one of the fairness assumptions the atomic statement liveness axiom and the guarded command skip axiom are presupposed. The abbreviations should be obvious.

Weak Process Fairness (WPF)

$\Box$ at $*[\ ] \wedge \Diamond \Box$ (at $*[\ ] \supset V_\ell g_\ell$) $\supset \Diamond V_\ell$ at S_ℓ

Weak Guard Fairness (WGF)

$\Box \Diamond$ at $*[\ ] \wedge \Diamond \Box$ (at $*[\ ] \supset g_\ell$) $\supset \Diamond$ at S_ℓ

Weak Channel Fairness (WCF)

$\Box \Diamond$ (at $*[\ ] \wedge$ at $*[\ ]'$) $\wedge \Diamond \Box$ ((at $*[\ ] \wedge$ at $*[\ ]'$) $\supset g_\ell \underline{m} g'_{\ell'}$) $\supset$
$\supset \Diamond$ (at $S_\ell \wedge$ at $S'_{\ell'}$)

Strong Process Fairness (SPF)

$\Box$ at $*[\ ] \wedge \Box \Diamond V_\ell\ g_\ell \supset \Diamond V_\ell$ at S_ℓ

Strong Guard Fairness (SGF)

$\Box \Diamond$ (at $*[\ ] \wedge g_\ell$) $\supset \Diamond$ at S_ℓ

Strong Channel Fairness (SCF)

$\Box \Diamond$ (at $*[\ ] \wedge$ at $*[\ ]' \wedge g_\ell \underline{m} g'_{\ell'}$) $\supset \Diamond$ (at $S_\ell \wedge$ at $S'_{\ell'}$).

We now compare the various fairness assumptions with respect to the sets of programs they cause to terminate.

DEFINITION. T(f), where f is one of the above fairness assumptions, is the set of CSP programs for which, when executed under the fairness assumption f in any initial state s, all execution sequences contain a state s for which ℓ_i = after P_i for all $i \in \{1,\ldots,n\}$ (i.e., the program terminates).

THEOREM.

$$\begin{array}{ccccc}
T(WPF) & \subset & T(SPF) \\
\not\cap & \neq & \not\cap \\
T(WGF) & \subset & T(SGF) \\
\not\cap & \neq & \not\cap \\
T(WCF) & \subset & T(SCF)\ .
\end{array}$$

PROOF. The inclusions and inequalities between the corresponding weak and strong cases are evident. An example for the inequality for the most interesting case, T(WCF) ≠ T(SCF) is the following.

$[P_1::$ x: = 0; y:= 1; $*[x=0, P_2!\ x \to y := -y \ \Box\ y=1, P_2!\ y \to$ skip $] \parallel$
$P_2::$ u: = 0; v:= 1; $*[u=0, P_1?\ u \to v := -v \ \Box\ v=1, P_1?\ v \to$ skip $]\ ]$

The inclusions and inequalities for the weak cases are easy; for the more interesting strong cases as follows.

T(SPF) ⊂ T(SGF)

By the local semantics axiom, $\Box$ at $*[\ ] \wedge \Box \Diamond V_\ell g_\ell$ is the equivalent to $\Box \Diamond$ (at $*[\ ] \wedge V_\ell g_\ell$), as this is the only way in which control can proceed. As $g_\ell \supset V_\ell g_\ell$ and at $S_\ell \supset V_\ell$ at S_ℓ, this gives T(SPF) ⊂ T(SGF).

T(SPF) ≠ T(SGF) by

b:= true; $*[b \to$ skip $\Box b \to$ b:= false$]$

T(SGF) ⊂ T(SCF)

This follows from the fact that there are only finitely many guards, whence $\Box \Diamond g_\ell$ implies that there is a g'_ℓ, such that $\Box \Diamond g_\ell\ \underline{m}\ g'_\ell$.
T(SGF) ≠ T(SCF) follows from the first example in this proof. □

As there are only finitely many guards, $\Box \Diamond V_\ell$ at S_ℓ implies that there is a g_ℓ such that $\Box \Diamond g_\ell$. As at S_ℓ implies V_ℓ at S_ℓ, this gives T(SPF) ⊂ T(SGF).

4. TEMPORAL LOGIC

We assume as given a temporal logic axiom system and rules for linear time like DUX as presented in, e.g, [P]; to handle assignment we assume extension of this system to predicate logic as outlined in, e.g., [HC] .
In proofs we make use of derived rules as presented in [BA] . E.g.: if $\vdash \Box p \wedge q \supset \bigcirc q$ then $\vdash \Box p \wedge q \supset \Box q$, the conditional invariant rule.

5. EXAMPLES.

We start by giving a very easy example, (i), in all detail. In (ii) we show how synchronization is treated.In practice most of the elementary steps in a proof can be left out, as (iii) shows. As the examples will show, the local semantics axiom and the conditional invariant rule are crucial to enable application of the fairness assumptions; namely to obtain the left hand side of the stated implication,

(i) Under the assumption of WGF a simple CSP program can model mutual exclusion and infinitely often access for two critical sections CS_1 and CS_2 consisting of sequentially composed atomic statements. Note, that WPF is not sufficient to guarantee access.

P::$*[$ true $\to CS_1 \ \Box$ true $\to CS_2]$

PROOF. Mutual exclusion holds by the exclusivity axiom. Proving mutual access amounts, by symmetry, to proving $\vdash$ at $*[\ldots] \supset \Diamond$ at CS_1.
As follows: (in S ≡ at S ∨ V_s, at S', S' substatement of S)

1) $\vdash$ at $*[\ldots] \supset \underbrace{\Box \text{ (at } *[\ ] \vee \text{ in } CS_1 \vee \text{ in } CS_2)}_{I:=}$. (LS)
2) $\vdash$ at $*[\ldots] \supset I \wedge$ at $*[\ldots]$ (1,T.L.,i.e.by temporal logic)
3) $\vdash$ at $*[\ldots] \supset I \wedge \Diamond$ at $*[\ldots]$ (T.L.)
4) $\vdash I \wedge \Diamond$ at $*[\ldots] \supset \bigcirc(\Diamond$ at $*[\ldots])$ (LS,ASL)
5) $\vdash I \wedge \Diamond$ at $*[\ldots] \supset \Box \Diamond$ at $*[\ldots])$ (4,T.L.:cond.invariant rule)

Now the fairness assumption is used;

6) $\vdash \Box \Diamond$ at $*[\ldots] \supset \Diamond$ at CS_1 (WGF)

7) $\vdash$ at $*[\ldots] \supset \Diamond$ at CS_1 (3,5,6,T.L.)

(ii) Termination of a program with synchronization under the assumption of WCF shall be proved. Again we give the proof in much detail.

Let b and c be initially true and not depend on x and y. Then the following program terminates under WCF,

$[P_1 :: *[b, P_2!\, x \rightarrow skip_1 \Box b, P_2?x \rightarrow b := \underline{false}\ ] \parallel$
$P_2 :: *[c, P_1?\, y \rightarrow skip_2 \Box c, P_1!y \rightarrow c := \underline{false}\ ]\ ]$

Note, that WCF is not sufficient to guarantee termination, but SGF is.

PROOF. Proving termination amounts, by symmetry, to proving

$\vdash$ at $*[1] \wedge$ at $*[2] \wedge b \wedge c \supset \Diamond$ after $*[1]$

As follows:

1) $\vdash$ at $*[1] \wedge$ at $*[2] \wedge b \wedge c \supset \Diamond$ (at b := false $\wedge$ at c := false)
$\vee \Box((\text{at } *[1] \vee \text{at } skip_1) \wedge (\text{at } *[2] \vee \text{at } skip_2) \wedge b \wedge c)$, (LS)
$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{I :=}$

Case 1

2) $\vdash$ at b:= false $\wedge$ at c := false $\supset \Diamond$(at $*[\ ] \wedge \neg b$) (LS,ASL)

3) $\vdash$ at $*[1] \wedge \neg b \supset \Diamond$ after $*[1]$ (GCS)

Case 2

4) $\vdash I \wedge$ at $*[1] \wedge$ at $*[2] \supset I \wedge \Diamond$ (at $*[1] \wedge$ at $*[2]$) (T.L.)

5) $\vdash I \wedge \Diamond$ (at $*[1] \wedge$ at $*[2]) \supset 0(\Diamond$ (at $*[1] \wedge$ at $*[2])$)$ (LS,ASL,M)

6) $\vdash I \wedge \Diamond$ (at $*[1] \wedge$ at $*[2]) \supset \Box \Diamond$ (at $*[1] \wedge$ at $*[2]$) (T.L.:cond.inv.rule)

7) $\vdash I \wedge \Diamond$ (at $*[1] \wedge$ at $*[2]) \supset \Box \Diamond$ (at $*[1] \wedge$ at $*[2] \wedge I$) (T.L.)

Now the fairness assumption is used

8) $\vdash I \wedge \Box \Diamond$ (at $*[1] \wedge$ at $*[2]) \supset \Diamond$(at b := false $\wedge$ at c := false) (I,WCF)

9) $\vdash$ at b := false $\supset \Diamond$ after $*[1]$ (2,3)

10) $\vdash$ at $*[1] \wedge$ at $*[2] \wedge b \wedge c \supset \Diamond$ after $*[1]$ (1,3,9,T.L.)

□

(iii) Termination of a program consisting of three processes under WGF shall be proved. We now leave out some straightforward detail to show how in practice proofs are not difficult to handle.

Let a,b and c be initially true and not depend on x,y and z. Then the following program terminates under WGF.

$[P_1 :: *[b, P_2!\, x \rightarrow skip_1 \Box b \rightarrow b := \underline{false}] \parallel$
$P_2 :: *[c, P_1?\, y \rightarrow skip_2 \Box c, P_3!\, y \rightarrow c := \underline{false}] \parallel$
$P_3 :: *[d, P_2?\, z \rightarrow d := \underline{false}\ ]\ ]$

PROOF. To prove : $\vdash \bigwedge_i$ at $*[i] \wedge b \wedge c \wedge d \supset \Diamond \bigwedge_i$ after $*[i]$

As follows:

1) $\vdash \bigwedge_i$ at $*[i] \wedge b \wedge c \wedge d \supset \Diamond \bigwedge_i$ after $[i]$
$\vee \Box ((\text{at } *[1] \vee \text{at skip}_1) \wedge (\text{at } *[2] \vee \text{at skip}_2)$
$\wedge$ at $*[3] \wedge b \wedge c \wedge d)$ $\}=:I$

Analogous to (ii) this leads to

2) $\vdash I \wedge \bigwedge_i$ at $*[i] \supset I \wedge \Box \Diamond \bigwedge_i$ at $*[i]$

Now the fairness assumption is used

3) $\vdash I \wedge \Box \Diamond \bigwedge_i$ at $*[i] \supset \Diamond$(at c := $\underline{\text{false}} \wedge$ at d := $\underline{\text{false}}$) (WGF)
$\wedge \Box$ (in $*[1] \vee$ after $*[1]$) (LS)

4) $\vdash$ at c := $\underline{\text{false}} \supset \Diamond$ after $*[2] \supset \Diamond \Box$ after $*[2]$ (ASL,GCS,M)

5) $\vdash$ at d := $\underline{\text{false}} \supset \Diamond$ after $*[3] \supset \Diamond \Box$ after $*[3]$ (ASL,GCS,M)

6) $\vdash \Box$ (after $*[2] \wedge$ after $*[3] \wedge$ (in $*[1] \vee$ after $*[1]$)) $\supset \Diamond$ after $*[1]$ (ASL,WGF,GCS)

7) $\vdash \bigwedge_i$ at $*[i] \wedge b \wedge c \wedge d \supset \Diamond \bigwedge_i$ after $*[i]$ (1,2,3,4,5,6,T.L.) □

(iv) Changing in example (iii) P_2 to

P_2' :: $*[c_1, P_1?y \rightarrow c_2 := \neg c_2 \ \Box \ c_2, P_3 \ ! \ y \rightarrow c_1 := c_2 := \underline{\text{false}}\ ]$

gives an example of a program for which SGF is, but WGF is not sufficient to ensure termination. The termination proof is analogous to the one for example (iii), employing an invariant I' changed accordingly to the change in P_2.

6. DISCUSSION

The above system enables us to study termination and other liveness properties of CSP programs under various fairness assumptions.
As to future goals the following:

1) Extending the system to full CSP is expected to be more or less straight forward, but careful and simple notation should be used in order not to obscure the intuition behind the axioms.

2) Termination due to properties of the well-foundedness might be described by adding a well-foundnesslike rule to DUX, like

if $\vdash \exists \omega \in \mathcal{W} \ P(\omega)$
and $\vdash \forall u \in \mathbb{N}(0 < u \leq \omega)(P(u) \supset \Diamond P(u-1))$
then $\vdash \Diamond P(0)$.

3) Abstracting to a higher level axiom system might be facilitated by studying examples using the low level system; it is expected that invariants used in the proofs may indicate more general proof principles.

4) Developing a notion of completeness for the system might be helped by comparing it to other total correctness systems for CSP, like given in [A].

5) P. van Emde Boas suggested that using branching time it might be possible to formulate fairness assumptions not defined as a restriction on one computation sequence, but involving several. It then might be possible to enforce, say, termination of programs not terminating under any of the fairness assumptions in this paper
We consider as an example, starting with b = c = d = e = $\underline{\text{true}}$,

$$
\begin{aligned}
&[P_1::*[b,P_2!x \rightarrow \text{skip} \;\square\; b,P_3!x \rightarrow b:= \underline{\text{false}}] \;\|\\
&\;P_2::*[c,P_1?y \rightarrow \text{skip} \;\square\; c,P_4!y \rightarrow c:= \underline{\text{false}}] \;\|\\
&\;P_3::*[d,P_4!z \rightarrow \text{skip} \;\square\; d,P_1?z \rightarrow d:= \underline{\text{false}}] \;\|\\
&\;P_4::*[e\ P_3?u \rightarrow \text{skip} \;\square\; e,P_2?u \rightarrow e:= \underline{\text{false}}]\ ]
\end{aligned}
$$

which is not guaranteed to terminate under any of the above fairness assumptions, but should terminate under the, intuitively formulated, assumption that if there always is a terminating branch in the future, then such branch will eventually be chosen.

ACKNOWLEDGEMENT. We are very grateful to Amir Pnueli, who gave an outline of CSP semantics as worked out for a subset in this paper.
We wish to thank Nissim Francez for both directly and indirectly contributing to this paper. Leslie Lamport we thank for illuminating discussions.

REFERENCES

[A] Apt, K.R., Justification of a proof system for communicating sequential processes, Erasmus University, Rotterdam (1981).

[AFdeR] Apt, K.R., Francez N. and De Roever, W.P., A proof system for communicating processes, RUU-80-4, University of Utrecht (1980).

[BA] Ben Ari, M., Complexity of proofs and models in programming logics, Thesis, Tel Aviv (1981).

[D] Dijkstra, E.W., Guarded commands, nondeterminacy and formal derivation of programs, CACM 18, 453-457 (1975).

[FdeR] Francez, N. and De Roever W.P., Fairness in communicating processes, Unpublished Extended Abstract (1980).

[GPSS] Gabbay, D., Pnueli, A., Shelah, S. and Stavi, J., On the Temporal Analysis of Fairness, Proc. 7th ACM Conf. on Principles of Programming Languages, Las Vegas (1980).

[HC] Hughes, G.E. and Cresswell, M.J., An introduction to model logic, Meth en & Co Ltd (1971).

[H] Hoare, C.A.R., Communicating Sequential Processes, CACM, 21, 666-677 (1978).

[O] Owicki, S., Axiomatic Proof Techniques for Parallel Programs. Diss. Cornell University (Comp.Sc.) TR 251 (1975).

[OL] Owicki, S. and Lamport, L., Proving Liveness properties of concurrent programs (1980).

[P] Pnueli, A., The temporal semantics of concurrent programs, Theoretical Computer Science 13, 45-60 (1981).

[S] Smolka, S.A., A Deductive-Operational Semantics for Distributed Programs, Technical Report No. CS-64 (1980). Brown Univ., R.I.

QUESTIONS AND ANSWERS

Lauer: Your analysis of fairness assumptions seems very oriented to a specific language. Would these distinctions between fairness assumptions still be valid if you made the study less language specific. I have a feeling that it all hinges on the fact that a mutual exclusive choice must be made, and you are simply distinguishing a number of different contexts in which choices occur in CSP.

Kuiper: It is certainly true, that it will not be the same for every language. But I think that most of these things will simplify a lot if you have other languages. We especially have chosen CSP, because it was such a difficult language. We wanted to show that using temporal logic and using this approach we were able to even deal with quite complicated fairness assumptions. But it is language specific.

de Roever: If you look at the definitions, in fact the crux is whether a certain move is enabled or not. So in fact on a high level there is uniformity. This winter I tried to investigate with Pnueli whether we could formalize that. Every week I got another fairness principle. So we ended up in specifying, instead of these six (fairness) principles for CSP, the principles for Ada, which is again slightly different. That is, the one fairness principle for the concurrency section of Ada. It is again slightly different from the fairness principle for strong semaphores. Also it is different from the other principles for semaphores which I know. So at this moment technically speaking I see no uniformity insight except for this one clause: if a move is infinitely often enabled it will be infinitly often taken, in case of strong fairness.

Apt: In the first place I want to make a small comment on this example of the last transparency. You do not need any assignment in the style of Back - how do you call it, random assignment? - because the problem is not there. You passed after this assignment and the problem only began there. So I think we want to indicate that this problem of the necessity of using the rule has nothing to do with the random assignment, because this might be the conclusion which one has. But I now have two questions: One is that you use the next-time operator, and it was used neither in the formulation of the fairness assumptions nor, as far as I could see, in the formulation of the proof rules which you used. I just wondered to what extent it is necessary to use it. Because this gives a certain granularity to the description which seems to be too detailed. You do not need to know what is exactly true after one step. It seems to suffice to know that eventually something is true, doesn't it? That's one question. The other question is that in the syntax, which you have, the syntax is extremely poor. I now checked that you cannot even have in these bodies of these looping constructs 2 consecutive assignments, and it can be either an assignment or skip, that's it. So you say that you have some completeness proof. The principles which were expressed, do they not depend on the particular syntax on the language and to what extent was the choice of the syntax was motivated?

Kuiper: Well, I should start, I think, with the first remarks you made. I agree with that. It is indeed not needed to use this construct. But, I thought it was more or less illustrative of how to use the rule. I have not been wanting to put any emphasis on that further on. As to your first question, if you look at the proof system, you will see that I use, in axioms, the next-time operator. And we have tried, that was our first attempt, to make a proof system without having to say things like this. But it became clear, after some time, why we failed to do it, namely because in that case, you do not know where you are in the meantime. If you only know that you are eventually at some place, then there is no guarantee that in between you were not at some weird place, which might bring you in great trouble with respect to other processes.

Apt: There are only a few places where you are before and after an assignment. There is no third possibility and this is what I am bothered about.

de Roever: I can answer the question conclusively. So let me do it. (Loud laughter) We first started out with trying to use Lamport's approach. You know there are two seminal papers on fairness assumptions, one written by Lamport and Owicki, and one by Pnueli. Lamport's seems the more structured one. So we started with Lamport and we went to something like 17 different versions of trying to get at a complete proof system. In the end it boiled down that the sound basis is that of Pnueli. And you see here the proof system of Pnueli's. After we had understood that, we realized why we could in fact use Lamport's approach again and remove the next time operator. Because what is standing here is in fact nothing else than a safety semantics. You say: either you do a move, or you don't do a move. If you do the move, something happens at that moment. If you don't do it, you still remain there. But the strength of proof system using safety semantics, in which you need to replace this next-time operator, is that of the completeness proofs for safety semantics which record in auxiliary variables the sets of sequences of atomic actions which have been taken. That is not elegant! As this insight came after we had understood this system, we preferred not to present here the other system, oriented upon the Lamport-Owicki approach. Because this is the fundamental one. And this is nothing else than safety semantics in it's strongest form; the one you are familiar with, on top of which the various temporal principles expressing justness and weak or strong fairness have been built.

Apt: Sorry, but I have another question.

Kuiper: As I understood you said that (in the statements) after the guards there can only be one assignment statement.

Apt: Yes, not even two consecutive ones.

Kuiper: Maybe then I was not making myself clear about that.

Apt: It's in the paper.

Kuiper: It was not intended to be so.

Apt: Well, that's how it is written.

Kuiper: I then have clearly been giving an impression that I didn't wanted to. Maybe we should discuss it later. (Kuipers post-conference note: Apt was right, I inadvertently had left out a clause in a definition in the conference version of the proceedings.)

Apt: Yes, but the problem is rather not whether you can have 2 or 3 assignments one after the other, but to what extent is this choice important here?

Kuiper: It's not important at all. You can have everything after it. As long as you know it terminates, it's enough.

Apt: Aha! So that's the choice.

Kuiper: There can even be guards. As long as you can prove it terminates, it is OK. There is no need to maintain only assignments or only simple statements. But it is done to make things more clear. What you have in this version is something like the most stripped down thing I could think of, to emphasize only what I wanted to show.

Kröger: First I have a very short remark to this last rule. I am not quite sure, but I would like to claim that you could replace this rule by a more general rule really belonging to temporal logic itself.

Kuiper: Without referring to wellfoundedness?

Kröger: Without wellfoundedness or things like that. Well, I am not absolutely sure at this very moment, but I think it's true. Secondly, a question: Have you any idea how to express in this framework more quantified fairness propositions like for example, the proposition: if two actions are enabled, say equally often, will they be scheduled equally often?

Kuiper: In an analogous framework, you can express these things. You can even express a definite amount of times after which you want something to be enabled. So there is no reason to say that quantification should not be there; equally often can also be expressed. It doesn't change anything fundamental. There are even papers which will show you how you can extend your temporal logic to dealing with constructs like this.

Apt: So you have to extend the logic?

Kuiper: Yes, because now I have nothing which says that after so and so many moves something happens, or after having had an equal amount of moves. But you can do this. Well, I should say that temporal logic tells you something about the parameter of time and the operators which I have chosen were only those saying that after a finite number of moves, something happens. If you want to specify that finite number to a certain fixed number, you can do that.