# Trace Semantics for the Owicki-Gries Theory Integrated with the Progress Logic from UNITY

Brijesh Dongol
Ian J. Hayes

April 2007

Technical Report SSE-2007-02

Division of Systems and Software Engineering Research
School of Information Technology and Electrical Engineering
The University of Queensland
QLD, 4072, Australia

http://www.itee.uq.edu.au/∼sse

# Trace Semantics for the Owicki-Gries Theory Integrated with the Progress Logic from UNITY

Brijesh Dongol and Ian J. Hayes

ARC Center for Complex Systems
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, AUSTRALIA

**Abstract.** The theory of Owicki and Gries has been used as a platform for safety-based verification and derivation of concurrent programs. It has also been integrated with the progress logic of UNITY which has allowed newer techniques of progress-based verification and derivation to be developed. However, a theoretical basis for the integrated theory has thus far been missing. In this paper, we provide a theoretical background for the logic of Owicki and Gries integrated with the logic of progress from UNITY. An operational semantics for the new framework is provided which is used to prove soundness of the progress logic.

## 1 Introduction

The theory of Owicki and Gries [OG76] is a popular platform for the verification of concurrent programs. Feijen and van Gasteren have shown that the theory is also useful for program derivation [FvG99]. However, the theory lacks a logic of progress which means only safety properties can be formally considered. To allow progress properties to be considered, Dongol and Goldson [DG06] integrated it with the progress logic of UNITY [CM88]. This change preserves the original theory and maintains applicability of the safety-based techniques of Feijen and van Gasteren. Furthermore, the extended theory has been used to develop newer techniques for progress-based derivation of concurrent programs [GD05,DM06a,DM06b,Don06]. In this paper, we provide a theoretical background for the integrated theory in [DG06], and give it a more rigorous and formal treatment.

Partial correctness is enough to reason about safety [OG76,FvG99] and although required for progress, Dongol and Goldson [DG06] do not elaborate on total correctness. Thus, we start by defining total correctness which requires careful consideration of atomicity, non-termination, and blocking. Then, an operational semantics for the theory in [OG76,FvG99] is provided.

Owicki and Gries [OG76] offer a partial solution for reasoning about the control state via the use of auxiliary variables. Dongol and Goldson [DG06] provide a more robust solution by describing how program counters may be incorporated into the framework, thus providing a full description of control. We also provide an operational semantics for the framework in [DG06], with which we can see that [DG06] is a valid extension of [OG76,FvG99]. The operational semantics is used to define state traces which allows us to relate our programs to linear-time temporal logic [MP92] and help formalise reachability, fairness and leads-to properties. The rules for proving progress in [DG06] then become theorems (as opposed to definitions [CM88,DG06]) which proves their soundness with respect to temporal logic. Fairness requirements that are simply assumed in [CM88,DG06] become more explicit and new theorems for proving progress are devised.

This paper is organised as follows. Section 2 presents background to our work. Section 3 describes our extensions to the theory of Owicki and Gries where describe how to represent the control state of a program. In Section 4, we present semantics for the new system and in Section 5, we describe how the logic of progress from UNITY may be incorporated into our extended formalism.

## 2 Background

### 2.1 Syntax

Our programming language is based on Dijkstra's guarded command language [Dij76], except that in our concurrent context, a selection command with all guards false is considered to be blocking. To handle progress properties, we will later introduce labels on all statements, but we begin with unlabelled statements.

**Definition 1 (Unlabelled Statement).** *For unlabelled statements $S_1, \ldots, S_n$; boolean expressions $B_1, \ldots, B_n$; a vector of distinct variables $\overline{x} \mathrel{\widehat{=}} x_1 \ldots x_m$; and a vector of expressions $\overline{E} \mathrel{\widehat{=}} E_1 \ldots E_m$, the syntax of* unlabelled statements *is given by:*

$$\begin{array}{lcl} \mathit{UStmt} & \mathrel{\widehat{=}} & \textbf{abort} \mid \textbf{skip} \mid \overline{x} := \overline{E} \mid S_1;\ S_2 \mid \\ & & \textbf{if}\ B_1 \longrightarrow S_1 [\!] \ldots [\!] B_n \longrightarrow S_n\ \textbf{fi} \mid \textbf{do}\ B_1 \longrightarrow S_1 [\!] \ldots [\!] B_n \longrightarrow S_n\ \textbf{od} \end{array}$$

The following abbreviations are introduced for a two-branch *selection* and a single branch *iteration*.

$$\begin{array}{lcl} \mathit{IF} & \mathrel{\widehat{=}} & \textbf{if}\ B_1 \longrightarrow S_1 [\!]\ B_2 \longrightarrow S_2\ \textbf{fi} \\ \mathit{DO} & \mathrel{\widehat{=}} & \textbf{do}\ B \longrightarrow S\ \textbf{od} \end{array}$$

The statements *IF* and *DO* may be taken as representatives for the more general form of the selection and iteration, respectively. Thus, we only treat *IF* and *DO* formally as their extension to the full syntax is straightforward.

### 2.2 Semantics

The values of the variables in a program define its current data state. A *state space* $\Sigma$ has type $\Sigma \mathrel{\widehat{=}} \mathit{Var} \longrightarrow \mathit{Val}$ where *Var* is a set of variables and *Val* a set of values. A *state* is a member of $\Sigma$. A *predicate* is a member of the set

$\mathcal{P}\Sigma \mathrel{\widehat{=}} \Sigma \longrightarrow \mathit{Bool}$ that maps each state to *true* or *false*. For state spaces $\Sigma$ and $\Gamma$, a *predicate transformer* from $\Sigma$ to $\Gamma$ has type $\mathcal{P}\Gamma \longrightarrow \mathcal{P}\Sigma$, so is a function that maps predicates over $\Gamma$ to predicates over $\Sigma$. We use the naming convention where $\sigma$, $\sigma'$ are states, $s, t$ are sequences of states, $E_1, E_2, \ldots$ are expressions, $x_1, x_2, \ldots$ are variables, $V_1$, $V_2, \ldots$ are values, $i, j, \ldots$ are labels, and $u, v, \ldots$ are indices.

To formalise our operational understanding of the language, we provide an operational semantics. Expression evaluation is represented by the function

$$\xrightarrow{e} : (\mathit{Expr} \times \Sigma) \rightarrow \mathit{Val}$$

that takes an expression and a state, and returns the value of the expression in the given state. Execution of an unlabelled statement is represented by the relation

$$\xrightarrow{us} : (\mathit{UStmt} \times \Sigma) \leftrightarrow (\mathit{UStmt} \times \Sigma)$$

which is defined in Fig. 1 using a small-step semantics [Plo04] that expresses the desired atomicity.

We use $\xrightarrow{us}^{*}$ to denote the reflexive transitive closure of $\xrightarrow{us}$, $\oplus$ for the functional override operator, and for vectors $\overline{x}$, $\overline{V}$, we use $\{\overline{x} \mapsto \overline{V}\}$ to denote the mapping $\{x_1 \mapsto V_1, \ldots, x_m \mapsto V_m\}$.

$$\boxed{\mathsf{asgn}}\ \frac{(\forall_{u:1..m}\ (E_u, \sigma) \xrightarrow{e} V_u)}{(\overline{x} := \overline{E}, \sigma) \xrightarrow{us} (\mathbf{skip}, \sigma \oplus \{\overline{x} \mapsto \overline{v}\})} \qquad \boxed{\mathsf{abort}}\ \frac{}{(\mathbf{abort}, \sigma) \xrightarrow{us} (\mathbf{abort}, \sigma')}$$

$$\boxed{\mathsf{seq\text{-}I}}\ \frac{(S_1, \sigma) \xrightarrow{us} (S_1', \sigma')}{(S_1;\ S_2, \sigma) \xrightarrow{us} (S_1';\ S_2, \sigma')} \qquad \boxed{\mathsf{seq\text{-}II}}\ \frac{}{(\mathbf{skip};\ S, \sigma) \xrightarrow{us} (S, \sigma)}$$

$$\boxed{\mathsf{seq\text{-}III}}\ \frac{}{(\mathbf{abort};\ S, \sigma) \xrightarrow{us} (\mathbf{abort}, \sigma)}$$

$$\boxed{\mathsf{IF\text{-}L}}\ \frac{(B_1, \sigma) \xrightarrow{e} \mathit{true}}{(\mathit{IF}, \sigma) \xrightarrow{us} (S_1, \sigma)} \qquad \boxed{\mathsf{IF\text{-}R}}\ \frac{(B_2, \sigma) \xrightarrow{e} \mathit{true}}{(\mathit{IF}, \sigma) \xrightarrow{us} (S_2, \sigma)}$$

$$\boxed{\mathsf{DO\text{-}loop}}\ \frac{(B, \sigma) \xrightarrow{e} \mathit{true}}{(\mathit{DO}, \sigma) \xrightarrow{us} (S;\ \mathit{DO}, \sigma)} \qquad \boxed{\mathsf{DO\text{-}exit}}\ \frac{(B, \sigma) \xrightarrow{e} \mathit{false}}{(\mathit{DO}, \sigma) \xrightarrow{us} (\mathbf{skip}, \sigma)}$$

**Fig. 1.** Operational semantics of unlabelled statements

Assignment statement $\overline{x} := \overline{E}$ uses the asgn rule where given that each expression $E_u$ in state $\sigma$ evaluates to value $V_u$, the state after the assignment maps each variable $x_u$ to $V_u$. Execution of statement **abort** results in an arbitrary post state. Sequential composition $S_1;\ S_2$ evaluates $S_1$ first using rule seq-I, but if $S_1$ is **skip**, uses rule seq-II so that $S_2$ may be evaluated. The sequential composition **abort** ; $S$ may be evaluated using rule seq-III whereby the result is the same as evaluating **abort** by itself. For statement *IF*, we execute either $S_1$ or $S_2$ depending on whether $B_1$ or $B_2$ evaluates to *true* in state $\sigma$. If both $B_1$ and $B_2$ evaluate to *true*, then either of the rules IF-L or IF-R may be used. Notice that no rule has been defined for the case that both $B_1$ and $B_2$ evaluate to *false* because no transition take place in the system, i.e., the *IF* statement blocks. For a *DO* statement, if $B$ evaluates to *true* in $\sigma$, we evaluate $S$ followed by *DO*, otherwise the *DO* terminates.

The weakest liberal precondition (*wlp*) for unlabelled statements is defined inductively as follows. We use notation $(\overline{x} := \overline{E}).P$ to denote the textual substitution of each $E_u$ for all free occurrences of $x_u$ in $P$ and $[\,P\,]$ to denote "$P$ holds in all states", i.e., $[\,P\,] \mathrel{\widehat{=}} (\forall_{\sigma:\Sigma}\ P.\sigma)$. Notation $\nu X::[\,X \equiv f(X)\,]$ denotes the greatest fixed point of $f(X)$.

**Definition 2 (Weakest Liberal Precondition).** *The* weakest liberal precondition (*wlp*) *of an unlabelled statement $S$ and a predicate $P$ is the weakest predicate that needs to hold before executing $S$, so that every terminating execution of $S$ results in a state satisfying $P$.*

1. $[\,wlp.\mathbf{abort}.P \;\equiv\; true\,]$
2. $[\,wlp.\mathbf{skip}.P \;\equiv\; P\,]$
3. $[\,wlp.(\overline{x} := \overline{E}).P \;\equiv\; (\overline{x} := \overline{E}).P\,]$
4. $[\,wlp.(S_1;\ S_2).P \;\equiv\; wlp.S_1.(wlp.S_2.P)\,]$
5. $[\,wlp.IF.P \;\equiv\; [\,(B_1 \Rightarrow wlp.S_1.P) \wedge (B_2 \Rightarrow wlp.S_2.P)\,]\,]$
6. $[\,wlp.DO.P \;\equiv\; \nu Y::[\,Y \equiv (B \Rightarrow wlp.S.Y) \wedge (\neg B \Rightarrow P)]\,]$

When only considering safety properties, Feijen and van Gasteren [FvG99] have already demonstrated that knowledge of partial correctness alone is enough. However, when reasoning about progress, one is also required to reason about termination. Thus, we present the weakest precondition (*wp*) predicate transformer [Dij76,DS90] which allows us to describe the total correctness of statements. Our programs are blocking, thus the definition of *wp* follows Nelson [Nel89]. We use $\mu X::[\,X \equiv f(X)\,]$ to denote the least fixed point of $f(X)$.

**Definition 3 (Weakest precondition).** *The* weakest precondition (*wp*) *of a statement $S$ and a predicate $P$ is the weakest predicate that needs to hold before executing $S$, so that $S$ is guaranteed to terminate in a state satisfying $P$.*

1. $[\,wp.\mathbf{abort}.P \;\equiv\; false\,]$
2. $[\,wp.\mathbf{skip}.P \;\equiv\; P\,]$
3. $[\,wp.(\overline{x} := \overline{E}).P \;\equiv\; (\overline{x} := \overline{E}).P\,]$
4. $[\,wp.(S_1;\ S_2).P \;\equiv\; wp.S_1.(wp.S_2.P)\,]$
5. $[\,wp.IF.P \;\equiv\; (B_1 \Rightarrow wp.S_1.P) \wedge (B_2 \Rightarrow wp.S_2.P)\,]$
6. $[\,wp.DO.P \;\equiv\; \mu Y::[\,Y \;\equiv\; (B \Rightarrow wp.S.Y) \wedge (\neg B \Rightarrow P)\,]]$

Using the *wp* we define the following terms. Note that a statement that terminates is not guaranteed to be executed because it might be blocked.

**Definition 4 (Guard, Enabled, Blocked, Terminates).** *For an unlabelled statement $S$, $g.S$ is the* guard *of $S$ where*

$$g.S \quad \widehat{=} \quad \neg wp.S.false.$$

*In a state $\sigma$, $S$ is* enabled *if $g.S.\sigma$ holds and* blocked *if $\neg g.S.\sigma$ holds. Statement $S$* terminates *from state $\sigma$ if $t.S.\sigma$ holds where*

$$t.S \quad \widehat{=} \quad wp.S.\,true\,.$$

### 2.3 Theory of Owicki and Gries

A program in our model consists of a number of concurrently executing processes where each process is just a sequential program. We define *PROC* to be the set containing the process identifiers of all processes in the program. Programs are preceded by predicate INIT that describes the allowable initial states where we require that INIT $\not\equiv$ *false*.

An *annotation* of a program represents the program's proof outline and consists of a collection of *assertions* (predicates on the state) at various points of interference in the program. We use the theory of Owicki and Gries [OG76] to prove correctness of a program's annotation. The main difficulty of establishing correctness of assertions in a concurrent environment is interference from other processes. For this reason, Owicki and Gries required an interference freedom proof obligation to ensure that an assertion is correct against execution of other processes. Feijen and van Gasteren re-interpret this rule as the "global correctness requirement" [FvG99].

**Definition 5 (Locally correct).** *An assertion $\{P\}$ occuring in process $p$ is* locally correct *if,*

- *$\{P\}$ is a precondition of $p$ and $[\,\text{INIT} \Rightarrow P\,]$, or*
- *$\{P\}$ is textually preceded by $\{Q\}\,S$, where $\{Q\}$ is locally correct and $[\,Q \Rightarrow wlp.S.P\,]$.*

**Definition 6 (Globally correct).** *An assertion $\{P\}$ occuring in process $p$ is* globally correct *if for each $\{Q\}\,S$ executed by a process (other than $p$) where $\{Q\}$ is correct, $[\,P \wedge Q \Rightarrow wlp.S.P\,]$.*

**Definition 7 (Correct).** *An assertion $P$ occuring in process $p$ is* correct *if it is both locally and globally correct. An annotation is* correct *if all assertions in the annotation are correct.*

We do not elaborate the Owicki-Gries theory here, but refer the interested reader to [OG76,FvG99,DG06] instead.

## 3 Representing the control state

In this section we present an extension to the programming model to support reasoning about the control state of a program.

## 3.1 Atomicity brackets

To allow finer control over the atomicity of statements, we use pairs of *atomicity brackets* '$\langle$' and '$\rangle$'. That is, given any statement $S$, execution of statement $\langle S \rangle$ takes place atomically and eliminates all points of interference within $S$. We refer to such a statement as a *coarse-grained atomic statement.* Note that we assume that **skip** and assignment statements are atomic, and hence the following hold:

$$\begin{array}{lcl} \langle \mathbf{skip} \rangle & \equiv & \mathbf{skip} \\ \langle \overline{x} := \overline{E} \rangle & \equiv & \overline{x} := \overline{E} \\ \langle \langle S \rangle \rangle & \equiv & \langle S \rangle. \end{array}$$

We take the view that an atomic statement that blocks partway through its execution is semantically equivalent to one that blocks at the start. In practice, implementation of such an atomic statement is not possible, however statement of the form $\langle S_1;\ \mathbf{if}\ B \longrightarrow S_2\ \mathbf{fi} \rangle$, may be rewritten to move the guard to the front by rewriting the statement as $\langle \mathbf{if}\ \mathit{wp}.S_1.B \longrightarrow S_1;\ S_2\ \mathbf{fi} \rangle$ so that blocking takes place at the start. Note that blocking forever is different from non-termination (cf Definition 11).

Atomicity brackets may also be placed around guard evaluations, for example $\mathbf{if}\ \langle B_1 \longrightarrow S_1 \rangle\ [\!]\ \langle B_2 \longrightarrow S_2 \rangle\ \mathbf{fi}$. Here, the evaluation of guards $B_1$ and $B_2$, and execution of statement $S_1$ or $S_2$ (depending on which guard holds) takes place atomically. We point out the awkward nature of our notation as the pair of atomicity brackets suggest two atomic guard evaluations, however, it is important realise that this is not the case and guard evaluation takes place atomically. In particular, the statement $\mathbf{if}\ \langle B \longrightarrow S_1 \rangle\ [\!]\ \langle \neg B \longrightarrow S_2 \rangle\ \mathbf{fi}$ is non-blocking as long as both $S_1$ and $S_2$ are non-blocking. A more general form of statement $\mathit{IF}$ is:

$$\mathbf{if}\ \langle B_1 \longrightarrow S_1 \rangle\ T_1\ [\!]\ \langle B_2 \longrightarrow S_2 \rangle\ T_2\ \mathbf{fi}$$

where $B_1$ and $B_2$ are evaluated atomically, and depending on whether $B_1$ or $B_2$ holds, $S_1$ or $S_2$ is executed atomically with the guard evaluation. After execution of $S_1$ control is transferred just before $T_1$, and after $S_2$ is executed, control is transferred to just before $T_2$. Thus, there is no point of interleaving between evaluation of $B_1$ and execution of $S_1$ (similarly between $B_2$ and $S_2$), and furthermore $B_1$ and $B_2$ are evaluated atomically. Similarly, a more general form of statement $\mathit{DO}$ is:

$$\mathbf{do}\ \langle B \longrightarrow S \rangle\ T\ \mathbf{od}.$$

A further concern when using atomicity brackets is that the atomic statement defined may not terminate. We take the view that a non-terminating atomic statement is equivalent to **abort**. The implications of this are discussed throughout the paper.

## 3.2 Labelled statements

The first step towards describing the control state of a process requires being able to refer to the next atomic statement to be executed. We do this by assigning a unique label to each atomic statement in the process.

The label at the start of a statement is called the *initial label* of that statement. In addition, a label is assigned to the end of the statement which is called the *final label* of the statement. A final label of a statement always labels the initial atomic statement of the statement that follows it. However, if there is no following statement, then the final label does not refer to any atomic statement, but simply marks the end of the process. Effectively, this scheme labels all the points of interference in a process. We define the type of labels to be $PC$ and use $PC_p$ to denote the set of labels of process $p$. We assume that $\tau \notin PC_p$ for each process $p$.

**Definition 8 (Labelled statement).** *For unlabelled statements $S, S_1, \ldots, S_n$, labelled statements $T, T_1, \ldots, T_n$, boolean expressions $B, B_1, \ldots, B_n$, a vector of distinct variables $\overline{x} \mathrel{\widehat{=}} x_1 \ldots x_m$, and a vector of expressions $\overline{E} \mathrel{\widehat{=}} E_1 \ldots E_m$, the syntax of a* labelled statement *takes the following form:*

$$\begin{array}{lcl} \mathit{LStmt} & \mathrel{\widehat{=}} & i{:}\,\langle S\rangle\, j{:} \mid T_1;\ T_2 \\ & & \mid i{:}\,\mathbf{if}\ \langle B_1 \to S_1\rangle\ T_1 [\!]\ldots[\!]\langle B_n \to S_n\rangle\ T_n\ \mathbf{fi}\, k{:} \\ & & \mid i{:}\,\mathbf{do}\ \langle B_1 \to S_1\rangle\ T_1 [\!]\ldots[\!]\langle B_n \to S_n\rangle\ T_n\ \mathbf{od}\, k{:} \end{array}$$

When we write $i{:}\, S\, j{:}$, we mean that the initial and final labels of $S$ are $i$ and $j$, respectively. Notice **abort** does not have a final label, and that the label before and after **skip** are different. For the sequential composition $T_1;\ T_2$ of labelled statements $T_1$ and $T_2$, we require the final label of $T_1$ to be equal to the initial label of $T_2$, otherwise the sequential composition is not well-formed. Furthermore, we use $i{:}\, S_1\, ;\ j{:}\, S_2\, k{:}$ as shorthand for $i{:}\, S_1\, j{:}\ ;\ j{:}\, S_2\, k{:}$. We define the following statements to be representatives of a labelled selection and iteration statements, respectively.

$$\begin{array}{lcl} \mathit{IF}_L & \mathrel{\widehat{=}} & i{:}\,\mathbf{if}\ \langle B_1 \to S_1\rangle\ j_1{:}\ T_1\ [\!]\ \langle B_2 \to S_2\rangle\ j_2{:}\ T_2\ \mathbf{fi}\, k{:} \\ \mathit{DO}_L & \mathrel{\widehat{=}} & i{:}\,\mathbf{do}\ \langle B \to S\rangle\ j{:}\ T\ \mathbf{od}\, k{:}\,. \end{array}$$

We use $p_i$ to denote "the atomic statement with initial label $i$ in process $p$".

## 3.3 Modelling program counters

The desire to make a conservative extension to the theory of Owicki and Gries, prompted by the desire to retain the calculational nature of *wlp*, has led us to use auxiliary variables to reason about the control state. Consequently, we formalise a program's control state by introducing an auxiliary variable $pc_p$ for each process $p$ in a way that models its 'program counter', i.e., the value of this variable indicates the active point of interference in the process, which is just the label of the next atomic statement to be executed, or the final label of the process if no such statement exists. Program counter $pc_p$ must be updated at every atomic statement in $p$ in a way that assigns $pc_p$ the final label of that statement. This is done by superimposing an auxiliary assignment to $pc_p$ on every atomic statement in $p$ except **skip**.

Since every atomic statement in process $p$ updates $pc_p$, explicitly mentioning updates to $pc_p$ unnecessarily adds clutter to our programs. Hence, we follow the convention that execution of each statement in process $p$ implicitly updates $pc_p$ to reflect the change in control state. Furthermore, we add the restriction that $pc_p$ may not appear in any statement, although it may appear in the annotation.

## 4 Semantics of labelled statements

### 4.1 Operational semantics

Providing an operational semantics for our new programming model is complicated because we allow atomicity brackets. We re-iterate a previous point: if statement $\langle S \rangle$ blocks partway through its execution, it is regarded as being semantically equivalent to blocking at the start. In order to make implementation of the statement possible, we may move the blocking to the start of the statement. However, when writing $\langle S \rangle$ it is still possible for $S$ to abort, for example, if $S$ is a non-terminating loop.

When defining an operational semantics for labelled statements, because labelled statements are an extension of unlabelled statements, it is easiest to define an additional layer between the semantics for labelled and unlabelled statements. It is also useful to define an identity statement. However, we are unable to use **skip** as the identity because it has property of updating the program counter. Hence, we include statement **id** in our system with the following restrictions

- the label before and after **id** are the same,
- **id** may not appear in any program, and
- **id** is the identity of sequential composition, i.e., $\mathbf{id};\ S \;=\; S \;=\; S;\ \mathbf{id}$ for any labelled statement $S$.

We define relation

$$\xrightarrow{t} : (\mathit{UStmt} \times \Sigma) \leftrightarrow (\{\mathbf{id}, \mathbf{abort}\} \times \Sigma)$$

that evaluates unlabelled statements atomically. Recall that by Definition 4, an unlabelled statement is guaranteed to terminate if $t.S$ holds. Thus, given an unlabelled statement $S$ and state $\sigma$, if $t.S.\sigma$ holds, we return a state that follows from the reflexive, transitive closure of $\xrightarrow{us}$. If an atomic statement does not terminate, it is essentially equivalent to **abort**. Thus, we obtain the terminating semantics in Fig. 2.

Labelled statements are evaluated using the family of relations

$$\xrightarrow{ls} : \mathit{PROC} \to ((\mathit{LStmt} \times \Sigma) \leftrightarrow (\mathit{LStmt} \times \Sigma))$$

$$\boxed{\textsf{term}}\ \frac{t.S.\sigma \qquad (S,\sigma) \xrightarrow{us}{}^{*} (\mathbf{skip},\sigma')}{(S,\sigma) \xrightarrow{t} (\mathbf{id},\sigma')} \qquad \boxed{\textsf{abort}}\ \frac{\neg t.S.\sigma}{(S,\sigma) \xrightarrow{t} (\mathbf{abort},\sigma)}$$

**Fig. 2.** Terminating semantics

which for each process represents a single atomic step of execution in the process.

We provide the operational semantics for labelled atomic statements in Fig. 3. When evaluating the coarse-grained atomic statement $\langle S \rangle$, the unlabelled statement $S$ is evaluated according to the semantics in Fig. 2, whereby we obtain an $S'$ that is either **skip** (if $S$ terminates) or **abort** (if $S$ does not terminate). If $S$ blocks in state $\sigma$, none of the rules in Fig. 3 apply.

$$\boxed{\textsf{CG}}\ \frac{(S,\sigma) \xrightarrow{t} (S',\sigma')}{(\langle S \rangle\ j\colon,\sigma) \xrightarrow{ls}_p (S',\sigma' \oplus \{pc_p \mapsto j\})}$$

**Fig. 3.** Labelled atomic statements

The rules for non-atomic labelled statements are straightforward and are presented in Fig. 4. Once again, we recall that following a $\xrightarrow{t}$ evaluation, we either obtain an **id** or **abort** depending on whether or not the atomic statement terminates from state $\sigma$. Thus, in Fig. 4, $A \in \{\mathbf{id}, \mathbf{abort}\}$.

## 4.2 Weakest (liberal) precondition semantics

In this section, we define the *wp* and *wlp* of labelled statements. Since the update to $pc_p$ occurs implicitly, we must parameterise both the *wp* and *wlp* by the identifier of the executing process.

**Definition 9 (Weakest liberal precondition).** *The* weakest liberal precondition *(wlp) of a labelled statement in process p to establish predicate P is:*

1. $[\,wlp_p.(i\colon \mathbf{abort}).P \quad\equiv\quad true\,]$
2. $[\,wlp_p.(i\colon \mathbf{id}\ i\colon).P \quad\equiv\quad P\,]$
3. $[\,wlp_p.(i\colon \mathbf{skip}\, j\colon).P \quad\equiv\quad (pc_p := j).P\,]$
4. $[\,wlp_p.(i\colon \overline{x} := \overline{E}\, j\colon).P \quad\equiv\quad (\overline{x}, pc_p := \overline{E}, j).P\,]$
5. $[\,wlp_p.(i\colon \langle S \rangle\, j\colon).P \quad\equiv\quad wlp.(S;\ pc_p := j).P)\,]$
6. $[\,wlp_p.(i\colon S_1;\ j\colon S_2\, k\colon).P \quad\equiv\quad wlp_p.(i\colon S_1\, j\colon).(wlp_p.(j\colon S_2\, k\colon).P)\,]$
7. $[\,wlp_p.IF_L.P \quad\equiv\quad (B_1 \Rightarrow wlp.(S_1;\ pc_p := j_1).(wlp_p.(j_1\colon T_1\, k\colon).P))$
   $\qquad \wedge\ (B_2 \Rightarrow wlp.(S_2;\ pc_p := j_2).(wlp_p.(j_2\colon T_2\, k\colon).P))$
8. $[\,wlp_p.DO_L.P \quad\equiv\quad \nu Y \colon\colon [Y \equiv (B \Rightarrow wlp.(S;\ pc_p := j).(wlp_p.(j\colon T\ i\colon).Y))$
   $\qquad \wedge\ (\neg B \Rightarrow (pc_p := k).P)]\,]$

**Definition 10 (Weakest precondition).** *The* weakest precondition *(wp) of a labelled statement in process p to establish predicate P is defined as:*

$$\boxed{\textsf{seq-I}}\ \frac{(S_1,\sigma) \xrightarrow{ls}_p (S_1',\sigma')}{(S_1;\ S_2,\sigma) \xrightarrow{ls}_p (S_1';\ S_2,\sigma')} \qquad \boxed{\textsf{seq-II}}\ \frac{}{(\mathbf{id};\ S_2,\sigma) \xrightarrow{ls}_p (S_2,\sigma)}$$

$$\boxed{\textsf{abort}}\ \frac{}{(\mathbf{abort},\sigma) \xrightarrow{ls}_p (\mathbf{abort},\sigma')}$$

$$\boxed{\textsf{IF-L}}\ \frac{(B_1,\sigma) \xrightarrow{e} \mathit{true} \qquad (S_1,\sigma) \xrightarrow{t} (A,\sigma')}{(\mathit{IF}_L,\sigma) \xrightarrow{ls}_p (A;\ T_1,\sigma' \oplus \{pc_p \mapsto j_1\})}$$

$$\boxed{\textsf{IF-R}}\ \frac{(B_2,\sigma) \xrightarrow{e} \mathit{true} \qquad (S_2,\sigma) \xrightarrow{t} (A,\sigma')}{(\mathit{IF}_L,\sigma) \xrightarrow{ls}_p (A;\ T_2,\sigma' \oplus \{pc_p \mapsto j_2\})}$$

$$\boxed{\textsf{DO-true}}\ \frac{(B,\sigma) \xrightarrow{e} \mathit{true} \qquad (S,\sigma) \xrightarrow{t} (A,\sigma')}{(\mathit{DO}_L,\sigma) \xrightarrow{ls}_p (A;\ T;\ \mathit{DO}_L,\sigma' \oplus \{pc_p \mapsto j\})}$$

$$\boxed{\textsf{DO-exit}}\ \frac{(B,\sigma) \xrightarrow{e} \mathit{false}}{(\mathit{DO}_L,\sigma) \xrightarrow{ls}_p (\mathbf{id},\sigma' \oplus \{pc_p \mapsto k\})}$$

**Fig. 4.** Labelled non-atomic statements

1. $[\, wp_p.(i{:}\,\mathbf{abort}).P \quad\equiv\quad \mathit{false}\,]$
2. $[\, wp_p.(i{:}\,\mathbf{id}\ i{:}\,).P \quad\equiv\quad P\,]$
3. $[\, wp_p.(i{:}\,\mathbf{skip}\,j{:}\,).P \quad\equiv\quad (pc_p := j).P\,]$
4. $[\, wp_p.(i{:}\,\overline{x} := \overline{E}\,j{:}\,).P \quad\equiv\quad (\overline{x}, pc_p := \overline{E}, j).P\,]$
5. $[\, wp_p.(i{:}\,\langle S\rangle\,j{:}\,).P \quad\equiv\quad wp.(S;\ pc_p := j).P)\,]$
6. $[\, wp_p.(i{:}\,S_1;\ j{:}\,S_2\,k{:}\,).P \quad\equiv\quad wp_p.(i{:}\,S_1\,j{:}\,).(wp_p.(j{:}\,S_2\,k{:}\,).P)\,]$
7. $[\, wp_p.\mathit{IF}_L.P \quad\equiv\quad (B_1 \Rightarrow wp.(S_1;\ pc_p := j_1).(wp_p.(j_1{:}\ T_1\,k{:}\,).P))$
   $\wedge\ (B_2 \Rightarrow wp.(S_2;\ pc_p := j_2).(wp_p.(j_2{:}\ T_2\,k{:}\,).P))$
8. $[\, wp_p.\mathit{DO}_L.P \quad\equiv\quad \mu\, Y{:}{:}\,[Y \equiv (B \Rightarrow wp.(S;\ pc_p := j).(wp_p.(j{:}\ T\ i{:}\,).Y))$
   $\wedge\ (\neg B \Rightarrow (pc_p := k).P)]\,]$

Note that we are able to assert correctness of $pc_p = i$ as a precondition of statement $p_i$. Local correctness of $pc_p = i$ follows from the definition of $wlp_p$, and global correctness follows because $pc_p$ is a local variable of $p$. This means we are free to interpret predicate $pc_p = i$ to mean that 'control in $p$ is at $p_i$' because $pc_p = i$ is a correct precondition of $p_i$ and because labels are unique within a process.

**Definition 11 (Guard, Termination).** *For a labelled statement $S$ in process $p$, the* guard *of $S$ denoted $g_p.S$, is the predicate $\neg wp_p.S.\mathit{false}$ and the* termination *of $S$ denoted $t_p.S$ is the predicate $wp_p.S.\,\mathit{true}$.*

### 4.3 Relating the semantics

The relationship between the operational and weakest liberal precondition semantics is established via the following theorem which may be proved using induction on the structure of $S$.

**Theorem 1.** *For a labelled statement $S$ in process $p$ and predicate $Q$, the following holds:*

$$(\forall_{\sigma:\Sigma}\ wlp_p.S.Q.\sigma\ \equiv\ (\forall_{\sigma':\Sigma}\ (S,\sigma) \xrightarrow{ls}{}_p^{*} (\mathbf{id},\sigma') \Rightarrow Q.\sigma')).$$

Hence, if the *wlp* of $S$ to establish $Q$ holds in state $\sigma$ and the reflexive, transitive closure of $\xrightarrow{ls}_p$ results in $(\mathbf{skip},\sigma')$, i.e., a terminating execution of $S$ results in state $\sigma'$, then $Q$ must hold in $\sigma'$. The following theorem relates the termination of statement $S$ to its operational description. We use $(S,\sigma) \xrightarrow{ls}{}_p^{\infty}$ to denote that evaluation of $(S,\sigma)$ *diverges*, i.e., generates an infinite sequence.

**Theorem 2.** *For a labelled statement $S$ in process $p$, the following holds:*

$$(\forall_{\sigma:\Sigma}\ t_p.S.\sigma\ \equiv\ \neg(\exists_{\sigma':\Sigma}\ (S,\sigma) \xrightarrow{ls}{}_p^{*} (\mathbf{abort},\sigma'))\ \wedge\ \neg((S,\sigma) \xrightarrow{ls}{}_p^{\infty})).$$

Hence, if $t_p.S$ holds in state $\sigma$, no evaluation of $S$ in $\sigma$ is aborting or diverging. The weakest precondition may be related to the operational semantics using Theorems 1 and 2, together with the following equation from [Dij76]:

$$wp_p.S.P\ \equiv\ wlp_p.S.P \wedge t_p.S.$$

Note that we cannot guarantee that $S$ can be executed just because $wp_p.S.P$ holds in state $\sigma$. To ensure that $S$ can be executed, we use the next theorem.

**Theorem 3.** *For a labelled statement $S$ in process $p$, the following holds:*

$$(\forall_{\sigma:\Sigma}\ g_p.S.\sigma\ \equiv\ (\exists_{\sigma':\Sigma,S':LStmt}\ (S,\sigma) \xrightarrow{ls}_p (S',\sigma'))).$$

### 4.4 Execution semantics

Using operational semantics from the sequential part of our programming language, we are able to formalise the execution model of concurrent programs. Defining $\textsc{Prgm}$ to be the set of all possible programs, the *state transition* relation

$$\hookrightarrow : \textsc{Prgm} \to (\Sigma \leftrightarrow \Sigma)$$

represents a single step of execution of the program (Fig. 5).

According to rule par, a program makes an atomic transition from state $\sigma$ to $\sigma'$ if there is a process in the program that takes an atomic step from $\sigma$ to $\sigma'$. Using this, we define a trace of a program as follows.

**Definition 12 (Trace, Complete Trace).** *A possibly infinite sequence of states $s$ is a* trace *of program $\mathcal{A}$ if $\textsc{Init}_{\mathcal{A}}.s_0 \wedge (\forall_{u:dom(s)-\{0\}}\ s_{u-1} \hookrightarrow_{\mathcal{A}} s_u)$. Trace $s$ is* complete *if either $dom(s) = \mathbb{N} \vee \neg(\exists_{\sigma:\Sigma}\ last(s) \hookrightarrow_{\mathcal{A}} \sigma)$ holds.*

Thus, a complete trace represents either a terminating, deadlocked, or infinite execution of a program. For a program $\mathcal{A}$, we let $\mathsf{T}_{\mathcal{A}}$ denote the set of all complete traces of the program.

$$\boxed{\textsf{par}}\ \frac{p \in PROC_{\mathcal{A}} \quad (S, \sigma) \xrightarrow{ls}_p (S', \sigma')}{\sigma \hookrightarrow_{\mathcal{A}} \sigma'}$$

**Fig. 5.** Execution semantics

## 5 A logic of progress

As we now have the means to reason about the control state of a program, we are in a position to extend the theory to support reasoning about progress requirements.

### 5.1 Trace semantics

In this section, we present a trace semantics for our model which allows us to review the logic of Manna and Pnueli [MP92] and formalise concepts such as fairness. For a sequence of states $s$ and formula $\mathcal{F}$, we use $s \vdash \mathcal{F}$ to mean "$s$ satisfies $\mathcal{F}$".

**Definition 13 (Always, Eventually, Unless).** *[MP92] For predicates $P$ and $Q$ and sequence of states $s$, we define*

$$\begin{array}{lcl} s \vdash \Box P & \equiv & (\forall_{u:dom(s)}\ P.s_u) \\ s \vdash \Diamond P & \equiv & (\exists_{u:dom(s)}\ P.s_u) \\ s \vdash P\,\mathcal{W}\,Q & \equiv & (\exists_{v:dom(s)}\ Q.s_v \wedge (\forall_{u:0..v-1}\ P.s_u)) \ \vee\ (\forall_{u:dom(s)}\ P.s_u) \end{array}$$

Thus, $s \vdash \Box P$ iff *all* states in $s$ satisfy $P$, $s \vdash \Diamond P$ iff *some* state in $s$ satisfies $P$, and $s \vdash P\,\mathcal{W}\,Q$ iff there either exists a state in $s$ that satisfies $Q$ and $P$ holds until $Q$ does, or $P$ always holds, in which case $Q$ may never be established.

**Definition 14 (Satisfiable, Valid).** *Given a set $T \mathrel{\widehat{=}} \{s \mid s \in seq\Sigma\}$, a temporal formula $\mathcal{F}$ is* satisfiable *in $T$ iff $(\exists_{s:T}\ s \vdash \mathcal{F})$ holds and* valid *in $T$ iff $(\forall_{s:T}\ s \vdash \mathcal{F})$ holds.*

For a set of sequences $T$, we use notation $T \models \mathcal{F}$ to denote that formula $\mathcal{F}$ is valid in $T$.

**Definition 15 (Invariant, Reachable).** *For a set $T \mathrel{\widehat{=}} \{s \mid s \in seq\Sigma\}$ and predicate $P$, $P$ is* invariant *in $T$ iff $T \models \Box P$ and $P$ is* reachable *in $T$ iff $\Diamond P$ is satisfiable in $T$.*

Temporal logic makes it easy to specify progress properties, however, proving this specification can be difficult. In [DG06,CM88], progress is proved using the 'leads-to' relation which is defined without using temporal logic. Instead one proves 'leads-to' in a calculational manner without reasoning about state traces. However, it is not easy to be convinced that the definition of 'leads-to' indeed captures its intended temporal meaning. Furthermore, two of the required conditions in the definition of 'leads-to' (cf theorems 4 and 5) are actually theorems of temporal logic.

**Definition 16 (Leads-to).** *For predicates $P$ and $Q$, $P$* leads-to *$Q$ (written $P \leadsto Q$) iff $\Box(P \Rightarrow \Diamond Q)$.*

**Theorem 4 (Transitivity).** *For predicates $P$ and $Q$, $P \leadsto Q$ holds if for some predicate $R$, $(P \leadsto R) \wedge (R \leadsto Q)$.*

*Proof.* For an arbitrary sequence of states $s$, we show that if $s \vdash P \leadsto R$ and $s \vdash R \leadsto Q$ holds, then $s \vdash P \leadsto Q$ holds. The proof makes use of the following properties from [MP92].

$$\Box P \wedge \Box Q \quad \equiv \quad \Box(P \wedge Q) \qquad (T1)$$
$$(P \Rightarrow Q) \quad \Rightarrow \quad (\Diamond P \Rightarrow \Diamond Q) \qquad (T2)$$
$$\Diamond\Diamond P \quad \equiv \quad \Diamond P \qquad (T3)$$

We now have the following calculation:

$$\begin{array}{ll}
 & (s \vdash P \leadsto R) \wedge (s \vdash R \leadsto Q) \\
\equiv & \quad \{\text{definition of } \leadsto\} \\
 & (s \vdash \Box(P \Rightarrow \Diamond R)) \ \wedge \ (s \vdash \Box(R \Rightarrow \Diamond Q)) \\
\equiv & \quad \{\text{logic}\} \\
 & s \vdash \Box(P \Rightarrow \Diamond R) \ \wedge \ \Box(R \Rightarrow \Diamond Q) \\
\Rightarrow & \quad \{\text{properties } (T1) \text{ and } (T2)\} \\
 & s \vdash \Box((P \Rightarrow \Diamond R) \ \wedge \ (\Diamond R \Rightarrow \Diamond\Diamond Q)) \\
\Rightarrow & \quad \{\text{property } (T3)\}\{\text{transitivity of } \Rightarrow\} \\
 & s \vdash \Box(P \Rightarrow \Diamond Q) \\
\equiv & \quad \{\text{definition of } \leadsto\} \\
 & s \vdash P \leadsto Q
\end{array}$$

□

**Theorem 5 (Disjunction).** *For predicates $P$ and $Q$ if $P \equiv (\exists_{m:W}\, P.m)$, for any set $W$, given that $m$ does not occur free in $Q$, $P \leadsto Q$ if $(\forall_{m:W}\, P.m \leadsto Q)$.*

*Proof.* Assuming $P \equiv (\exists_{m:W}\ P.m)$ and $m$ is not free in $Q$, for an arbitrary sequence of states $s$, we show that $s \vdash P \leadsto Q$ holds provided $(\forall_{m:W}\ s \vdash P.m \leadsto Q)$ holds. In this proof, we use the following property from [MP92].

$$(\forall_{x:T}\ \Box P) \quad \equiv \quad \Box(\forall_{x:T}\ P) \qquad (D)$$

We have:

$$\begin{array}{ll}
 & (\forall_{m:W}\ s \vdash P.m \leadsto Q) \\
\equiv & \quad \{\text{definition of } \leadsto\} \\
 & (\forall_{m:W}\ s \vdash \Box(P.m \Rightarrow \Diamond Q)) \\
\equiv & \quad \{\text{trace property}\} \\
 & s \vdash (\forall_{m:W}\ \Box(P.m \Rightarrow \Diamond Q)) \\
\equiv & \quad \{\text{property } (D)\} \\
 & s \vdash \Box(\forall_{m:W}\ P.m \Rightarrow \Diamond Q) \\
\Rightarrow & \quad \{m \text{ not free in } Q\} \\
 & s \vdash \Box((\exists_{m:W}\ P.m) \Rightarrow \Diamond Q) \\
\equiv & \quad \{\text{definition of } \leadsto\}\{P \equiv (\exists_{m:W}\ P.m)\} \\
 & s \vdash P \leadsto Q
\end{array}$$

□

**Lemma 6 (Contradiction).** *For predicates $P$ and $Q$,*

$$P \rightsquigarrow Q \;\equiv\; (P \wedge \neg Q) \rightsquigarrow Q.$$

By incorporating temporal logic directly into the framework, we are able to formalise properties such as fairness. Our definitions are closely related to the formalisation of fairness in [Lam02], however, a stronger definition of strong fairness is provided in order to establish a more intuitive link between weak and strong fairness than Lamport [Lam02].

**Definition 17 (Weakly fair, Strongly fair).** *For a program $\mathcal{A}$, a trace $s \in \mathsf{T}_\mathcal{A}$ is* weakly fair *iff*
$(\forall_{p:PROC_\mathcal{A}} (\forall_{i:PC_{\mathcal{A},p}} s \vdash \Box\Diamond(pc_p \neq i \vee \neg g_p.p_i))).$
*Trace $s \in \mathsf{T}_\mathcal{A}$ is* strongly fair *iff*
$(\forall_{p:PROC_\mathcal{A}} (\forall_{i:PC_{\mathcal{A},p}} s \vdash \Box(\Box\Diamond(pc_p = i \wedge g_p.p_i) \Rightarrow \Diamond(pc_p \neq i)))).$

Thus, for a program $\mathcal{A}$, trace $s$ is weakly fair iff for each $s_u$, there is a future state $s_v$ for which $(pc_p \neq i \vee \neg g_p.p_i).s_v$ holds, i.e., if $g_p.p_i$ is continuously true, then $pc_p \neq i$ must hold. Note that by Lemma 6 (contradiction), $s$ is weakly fair iff $s \vdash (pc_p = i \wedge g_p.p_i) \rightsquigarrow (pc_p \neq i \vee \neg g_p.p_i)$. Trace $s$ is strongly fair iff for every $s_u$ it is always the case if $pc_p = i \wedge g_p.p_i$ always eventually becomes true, then eventually $pc_p \neq i$ holds. Note that because $\Box(P \Rightarrow Q) \Rightarrow (\Box P \Rightarrow \Box Q)$, if $s$ is strongly fair then $s \vdash \Box\Diamond(pc_p = i \wedge g_p.p_i) \Rightarrow \Diamond(pc_p \neq i)$ holds. Lemma 7 demonstrates why we do not use this weaker version as a definition.

We use notation $\mathsf{WF}_\mathcal{A}$ and $\mathsf{SF}_\mathcal{A}$ to respectively denote the weakly fair and strongly fair traces of program $\mathcal{A}$. The formal definitions of weak and strong fairness allows one to describe different conditions under which a progress property might hold. For example, given a formula $\mathcal{F}$ and program $\mathcal{A}$, proving $\mathsf{WF}_\mathcal{A} \models \mathcal{F}$ and $\mathsf{SF}_\mathcal{A} \models \mathcal{F}$ shows that $\mathcal{F}$ holds for weakly fair and strongly fair traces of $\mathcal{A}$, respectively. Note that $\mathsf{WF}_\mathcal{A} \subseteq \mathsf{T}_\mathcal{A}$ by the definition of $\mathsf{WF}_\mathcal{A}$.

**Lemma 7.** $\mathsf{SF}_\mathcal{A} \subseteq \mathsf{WF}_\mathcal{A}$.

*Proof.* We have the following calculation:

$$\begin{array}{ll}
 & \Box\Diamond(pc_p = i \wedge g_p.p_i) \Rightarrow \Diamond(pc_p \neq i) \\
\equiv & \quad \{\text{logic}\}\{\neg\Box P \equiv \Diamond\neg P\} \\
 & \Diamond\Box(pc_p \neq i \vee \neg g_p.p_i) \vee \Diamond(pc_p \neq i) \\
\equiv & \quad \{\Diamond(P \vee Q) \equiv \Diamond P \vee \Diamond Q\} \\
 & \Diamond(\Box(pc_p \neq i \vee \neg g_p.p_i) \vee pc_p \neq i) \\
\Rightarrow & \quad \{\Box P \Rightarrow P\}\{\text{logic}\} \\
 & \Diamond(pc_p \neq i \vee \neg g_p.p_i)
\end{array}$$

Thus, we get,

$$\begin{array}{ll}
 & \Box(\Box\Diamond(pc_p = i \wedge g_p.p_i) \Rightarrow \Diamond(pc_p \neq i)) \\
\Rightarrow & \quad \{\Box \text{ is monotonic}\} \\
 & \Box\Diamond(pc_p \neq i \vee \neg g_p.p_i)
\end{array}$$

which proves that a strongly fair trace is also weakly fair. $\Box$

## 5.2 The progress logic

We now present our progress logic and prove its soundness with respect to linear time temporal logic. The basis of the progress logic in [CM88,DG06] is the unless (**un**) relation which is defined as follows.

**Definition 18 (Unless).** *Given a program $\mathcal{A}$, predicates $P$ and $Q$, $P$ $\mathbf{un}_{\mathcal{A}}$ $Q$ holds iff*
$(\forall_{p:PROC_{\mathcal{A}}}\ (\forall_{i:PC_{\mathcal{A},p}}\ [\,P \wedge \neg Q \wedge pc_p = i \;\Rightarrow\; wp_p.p_i.(P \vee Q)\,])).$

Thus, a program satisfies $P$ $\mathbf{un}_{\mathcal{A}}$ $Q$ if for each atomic statement $S$ in the program, execution of $S$ from a state that satisfies $P \wedge \neg Q$ is guaranteed to terminate in a state that satisfies $P \vee Q$. Note that Definition 18 considers all states in the program, including those that are not reachable. Hence, a proof of $\mathsf{T}_{\mathcal{A}} \models P\,\mathcal{W}\,Q$ does not constitute a proof of $P$ $\mathbf{un}_{\mathcal{A}}$ $Q$ because the proof of $\mathsf{T}_{\mathcal{A}} \models P\,\mathcal{W}\,Q$ will only consider reachable states.

Furthermore, given a program $\mathcal{A}$, if $\mathsf{T}_{\mathcal{A}} \models P\,\mathcal{W}\,\mathit{false}$ holds, then $\mathsf{T}_{\mathcal{A}} \models \Box P$ must be true. However, if $P$ $\mathbf{un}_{\mathcal{A}}$ *false* holds, then we cannot conclude that $P$ holds initially. For $P$ to be an invariant of $\mathcal{A}$, we require that both $P$ $\mathbf{un}_{\mathcal{A}}$ *false* and $[\,\textsc{Init}_{\mathcal{A}} \Rightarrow P\,]$ be true. This difference is highlighted by the following lemma. Given sequences $s$ and $t$, we use notation $s \preceq t$ and $s \prec t$ to respectively denote that $s$ is a *prefix* and *proper prefix* of $t$.

**Lemma 8.** *Given a program $\mathcal{A}$, if $P$ and $Q$ are predicates such that*

$$[\,\textsc{Init}_{\mathcal{A}} \;\Rightarrow\; P \vee Q\,] \tag{1}$$
$$P\ \mathbf{un}_{\mathcal{A}}\ Q \tag{2}$$

*then $\mathsf{T}_{\mathcal{A}} \models P\,\mathcal{W}\,Q$.*

*Proof.* Given a program $\mathcal{A}$ that satisfies (1) and (2) and a trace $t \in \mathsf{T}_{\mathcal{A}}$, we show that $t \vdash P\,\mathcal{W}\,Q$ by induction over prefixes of $s$ of $t$.

For the base case we consider the prefix of length one, i.e., $s \preceq t$ such that $\mathit{size}(s) = 1$. Note that this gives us the state of the system after initialisation. We have the following calculation:

$$\begin{array}{ll}
 & s \vdash P\,\mathcal{W}\,Q \\
\equiv & \quad \{\text{Definition 13}\}\{\text{as } \mathit{size}(s) = 1\} \\
 & (Q.s_0 \wedge \mathit{true}) \vee P.s_0 \\
\equiv & \quad \{\text{logic}\} \\
 & (P \vee Q).s_0 \\
\Leftarrow & \quad \{\text{logic using (1)}\} \\
 & \textsc{Init}_{\mathcal{A}}.s_0 \\
\equiv & \quad \{s \preceq t\} \\
 & \textsc{Init}_{\mathcal{A}}.t_0 \\
\equiv & \quad \{t \in \mathsf{T}_{\mathcal{A}}\} \\
 & \mathit{true}
\end{array}$$

Our inductive hypothesis assumes that the result holds for trace $s$ such that $s \prec t \wedge \mathit{size}(s) = k$, i.e., we assume $(\exists_{v:\mathit{dom}(s)}\ Q.s_v \wedge (\forall_{u:0..v-1}\ P.s_u)) \vee (\forall_{v:\mathit{dom}(s)}\ P.s_v)$ holds. Note that $s$ does not represent a complete trace of $\mathcal{A}$ because $s \prec t$, hence, a state $\sigma$ such that $\mathit{last}(s) \hookrightarrow_{\mathcal{A}} \sigma$ exists. By logic, we may use the following equivalent assumption

$$(\exists_{v:\mathit{dom}(s)}\ Q.s_v \wedge (\forall_{u:0..v-1}\ P.s_u)) \vee (\forall_{v:\mathit{dom}(s)}\ (P \wedge \neg Q).s_v) \qquad (3)$$

Using this, we would like to show that for all traces $s'$ such that $s \prec s' \preceq t \wedge \mathit{size}(s') = k+1$ holds, the following is true:

$$(\exists_{v:\mathit{dom}(s')}\ Q.s'_v \wedge (\forall_{u:0..v-1}\ P.s'_u)) \vee (\forall_{v:\mathit{dom}(s')}\ P.s'_v) \qquad (4)$$

The implication is proved by case analysis on the disjuncts of (3). The first case

$$(\exists_{v:\mathit{dom}(s)}\ Q.s_v \wedge (\forall_{u:0..v-1}\ P.s_u)) \ \Rightarrow\ (4)$$

is trivial. For the second case, we have:

$$\begin{array}{ll} & (\forall_{v:\mathit{dom}(s)}\ (P \wedge \neg Q).s_v) \\ \Rightarrow & \quad \{\text{logic: } s \prec s'\} \\ & (\forall_{v:\mathit{dom}(s)}\ (P \wedge \neg Q).s'_v) \\ \Rightarrow & \quad \{\text{logic: as } P\ \mathbf{un}_{\mathcal{A}}\ Q\}\{s \prec s', \text{ thus } s'_{k+1} \text{ exists}\} \\ & (\forall_{v:\mathit{dom}(s)}\ (P \wedge \neg Q).s'_v) \wedge (P \vee Q).s'_{k+1} \\ \Rightarrow & \quad \{\text{by logic and sequence property}\} \\ & (\exists_{v:\mathit{dom}(s')}\ Q.s'_v \wedge (\forall_{v:0..k}\ P.s'_v)) \vee (\forall_{v:\mathit{dom}(s')}\ P.s'_v) \end{array}$$ □

Note that $P\ \mathbf{un}_{\mathcal{A}}\ Q$ does not guarantee that $Q$ will ever hold, for (an extreme) example, *true* $\mathbf{un}_{\mathcal{A}}\ Q$ holds for all $Q$, including *false*. To guarantee that a property is eventually established via the execution of a single statement, Chandy and Misra [CM88] define the *ensures* relation which appears as the immediate progress rule in [DG06]. In [CM88,DG06], this relation forms the base case for the definition of leads-to which we have defined using temporal logic (cf Definition 16).

Unlike transitivity and disjunction [CM88,DG06], the immediate progress rule (or ensures in UNITY) is bound to the program under consideration, as opposed to being a property of temporal logic. Thus, we present immediate progress as a theorem where $P \leadsto Q$ holds whenever the conditions required by the theorem hold. Our treatment turns out to be more favourable than [CM88,DG06] because conditions hidden away in [CM88,DG06] such as the weak-fairness requirement are more visible, and furthermore, we are able to prove that the conditions are sound. Later, we present a version of immediate progress that holds for strongly fair traces (cf Theorem 10), and a version that does not require any fairness conditions (cf Theorem 11).

**Theorem 9 (Immediate progress).** *Given a program $\mathcal{A}$, for predicates $P$ and $Q$, $\mathsf{WF}_{\mathcal{A}} \models P \leadsto Q$ holds if $P\ \mathbf{un}_{\mathcal{A}}\ Q$ holds, and*

$$(\exists_{p:\mathit{PROC}_{\mathcal{A}}}\ (\exists_{i:\mathit{PC}_{\mathcal{A},p}}\ [\,P \wedge \neg Q \ \Rightarrow\ \mathit{pc}_p = i \wedge g_p.p_i \wedge \mathit{wp}_p.p_i.Q\,])). \qquad (5)$$

To make sense of the Theorem 9 (immediate progress) we provide these interpretative notes. $\mathsf{WF}_{\mathcal{A}} \models P \rightsquigarrow Q$ is justified on the basis of being able to execute a continually enabled atomic statement that establishes $Q$. To see how the theorem formalises this, since $P\ \mathbf{un}_{\mathcal{A}}\ Q$ must hold, we can be assured that $P$ remains true as long as $\neg Q$ is true. Second, we establish that control of process $p$ is at an atomic statement $p_i$, that $p_i$ is enabled when $P \wedge \neg Q$ is true, and that execution of $p_i$ makes $Q$ true. It follows from $P\ \mathbf{un}_{\mathcal{A}}\ Q$ that $p_i$ is continually enabled as long as $\neg Q$ is true and because we are assuming weak fairness, that $p_i$ must eventually be executed whereby $Q$ is established.

*Proof.* [1] We prove that $\mathsf{WF}_{\mathcal{A}} \models P \rightsquigarrow Q$ holds if the conditions for immediate progress hold. By Lemma 6 (contradiction), we may equivalently prove $\mathsf{WF}_{\mathcal{A}} \models P \wedge \neg Q \rightsquigarrow Q$.

Suppose $s \in \mathsf{WF}_{\mathcal{A}}$ and for some $k$, $(P \wedge \neg Q).s_k$ holds. By (5) there exists $p \in \mathit{PROC}_{\mathcal{A}}$ and there exists an $i \in \mathit{PC}_{\mathcal{A},p}$ such that $(pc_p = i \wedge g_p.i \wedge wp_p.p_i.Q).s_k$ holds. Note that by Theorem 3, because $s$ is a complete trace the transition $s_k \hookrightarrow_{\mathcal{A}} s_{k+1}$ exists.

Now consider process $q$ different from $p$. If transition $s_k \hookrightarrow_{\mathcal{A}} s_{k+1}$ follows from $(S, s_k) \xrightarrow{ls}_q (S', s_{k+1})$, because $P\ \mathbf{un}_{\mathcal{A}}\ Q$ holds, so does $(P \vee Q).s_{k+1}$. If $Q.s_{k+1}$ holds we are done. So, we assume that no $q$ transition establishes $Q$ which means for all $v > k$ that follows from a $q$ transition, $(P \wedge \neg Q).s_v$ holds. We now have

$$
\begin{array}{ll}
 & (P \wedge \neg Q).s_k \\
\Rightarrow & \quad \{s \in \mathsf{WF}_{\mathcal{A}}\} \\
 & (\exists_{v:dom(s)}\ v > k \wedge (pc_p \neq i \vee \neg g_p.p_i).s_v) \\
\Rightarrow & \quad \{\text{assumption: every } q \text{ transition establishes } P \wedge \neg Q\} \\
 & (\exists_{v:dom(s)}\ v > k \wedge (s[k..v-1] \vdash \Box(P \wedge \neg Q)) \wedge (pc_p \neq i \vee \neg g_p.p_i).s_v) \\
\equiv & \quad \{P \wedge \neg Q \Rightarrow pc_p = i \wedge wp_p.p_i.Q\} \\
 & (\exists_{v:dom(s)}\ v > k \wedge Q.s_v)
\end{array}
$$
□

Having formalised fairness, in addition to previously known theorems, we are able to present a version of immediate progress that holds for strongly fair traces (Theorem 10), and a version that does not require any fairness conditions (Theorem 11).

**Theorem 10 (Immediate progress under strong fairness).** *Given a program $\mathcal{A}$, for predicates $P$ and $Q$, $\mathsf{SF}_{\mathcal{A}} \models P \rightsquigarrow Q$ if $P\ \mathbf{un}_{\mathcal{A}}\ Q$ holds, and*

$$[\,P \wedge \neg Q \ \Rightarrow\ (\exists_{p:PROC_{\mathcal{A}}} (\exists_{i:PC_{\mathcal{A},p}}\ pc_p = i \wedge g_p.p_i \wedge wp_p.p_i.Q))\,]. \qquad (6)$$

[1] On proving Theorem 9, we discover an error in the definition of **un** in [DG06]. In [DG06], $P\ \mathbf{un}_{\mathcal{A}}\ Q$ holds if $P \wedge \neg Q \Rightarrow wlp.S.(P \vee Q)$ holds for all statements $S$ in $\mathcal{A}$. However, partial correctness provided by *wlp* is not enough to guarantee that $P \rightsquigarrow Q$ holds. For Theorem 9, until $p_i$ is executed, all processes $q$ different from $p$ must establish $P \vee Q$. However if $P\ \mathbf{un}_{\mathcal{A}}\ Q$ is defined using the *wlp*, then a statement in $q$ might not terminate whereby $P \rightsquigarrow Q$ will not hold.

The theorem states that $P \wedge \neg Q$ needs to imply that there is a enabled statement that establishes $Q$ and that control is currently at that statement. Thus, execution of a process $q$ different from $p$ may disable $p_i$ as long as it enables some other statement that can establish $Q$. Condition (6) required by Theorem 10 is weaker than condition (5) required by Theorem 9, however, by weakening this condition, Theorem 10 only holds for strongly fair traces of the program.

**Theorem 11 (Immediate progress under no fairness).** *Given a program $\mathcal{A}$, for predicates $P$ and $Q$, $\mathsf{T}_{\mathcal{A}} \models P \rightsquigarrow Q$ if*

$$\begin{aligned}[\, P \wedge \neg Q \;\; \Rightarrow \;\; & (\forall_{p:PROC_{\mathcal{A}}} (\forall_{i:PC_{\mathcal{A},p}} \; pc_p = i \wedge g_p.p_i \Rightarrow wp_p.p_i.Q)) \\ & \wedge (\exists_{p:PROC_{\mathcal{A}}} (\exists_{i:PC_{\mathcal{A},p}} \; pc_p = i \wedge g_p.p_i))\,].\end{aligned} \tag{7}$$

Thus, $P \rightsquigarrow Q$ holds for all traces in the program if $P \wedge \neg Q$ implies all enabled processes establish $Q$ and one of these processes is enabled. Condition (7) is stronger than (5), but Theorem 11 does not impose any fairness requirements. Note that (7) may equivalently be written as

$$[\, P \wedge \neg Q \;\; \Rightarrow \;\; (\forall_{p:PROC_{\mathcal{A}}} \; g_p.p_{pc_p} \Rightarrow wp_p.p_{pc_p}.Q) \wedge (\exists_{p:PROC_{\mathcal{A}}} \; g_p.p_{pc_p})\,].$$

## 6 Conclusion and related work

The framework used in [OG76,FvG99,DG06] provides an approach where concurrent programs are modelled as a number of sequential processes executing in parallel. In [DG06], the safety logic of Owicki and Gries [OG76] is integrated with the progress logic from UNITY [CM88] where immediate progress, transitivity and disjunction are presented as the definition of $\rightsquigarrow$. Techniques for the progress-based derivation of concurrent programs that use the progress logic from [DG06] have already been developed [GD05,DM06a,DM06b], however, a theoretical backing for the logic has thus far not been provided.

Although $\rightsquigarrow$ is a liveness property, the presentation in [CM88,DG06] does not refer to temporal logic [MP92]. Jutla et al [JKR89] describe the weakest leads-to predicate transformer which is related to the progress logic of UNITY and to branching-time temporal logic [EH86], however, the relationship between UNITY and branching-time temporal logic is not established. Furthermore, their proofs assume $wp = wlp$. Gerth and Pnueli [GP89] show how UNITY could have been obtained as a specialisation of transition logic and linear-time temporal logic [MP92], thus providing a theoretical backing for UNITY.

This paper expands on [GP89] and proves soundness of the UNITY logic integrated into a fundamentally different programming model. An operational description of the programming model from [DG06] is provided, which allows temporal logic to be encoded directly into the framework. Progress properties are then described at a temporal level and the definitions from [DG06] are reformulated as theorems which validate their soundness. The new presentation separates theorems of temporal logic from theorems of the progress logic more clearly and requirements such as weak fairness that are implicit in [CM88,DG06] are now explicitly stated within the theorems. The usefulness of this is demonstrated by our ability to devise new theorems (Theorems 10 and 11) that describe the conditions necessary for $P \rightsquigarrow Q$ to hold under various fairness assumptions.

Owicki and Lamport [OL82] present a proof system in which the temporal operators $\Box$ and $\Diamond$ have been incorporated into the Owicki-Gries formalism. One of the drawbacks of their system is that blocking statements have not been described, and one must simulate these using the looping construct. Their logic is also missing 'unless' operator. Furthermore, keywords, 'at', 'after' and 'in' are used to describe the control state of the program, and temporal logic has been encoded directly (as opposed to axiomatically) into their logic. This has meant that the method needs to stay within the realms of logical reasoning, as opposed to algebraic calculation, which Feijen and van Gasteren [FvG99] point out, is not suitable in the context of program derivation. Lamport [Lam02] describes a framework that encodes temporal logic into a logic of actions, however, the framework is only suitable for describing specifications not programs.

As for further work, Chandy and Misra [CM88] suggest that ensures, transitivity, and disjunction are complete, in that, any proof of $P \rightsquigarrow Q$ can be proved via a finite number of applications of ensures, transitivity, and disjunction. Completeness of the UNITY logic is addressed in [Pac92] and in [GP89], UNITY is proved to be relatively complete to transition logic with linear-time temporal logic. We have not yet managed to obtain a completeness result and leave a more rigorous proof of completeness as a topic for further investigation.

## References

[CM88] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley Longman Publishing Co., Inc., 1988.

[DG06] B. Dongol and D. Goldson. Extending the theory of Owicki and Gries with a logic of progress. *Logical Methods in Computer Science*, 2(6):1–25, March 2006.

[Dij76] E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[DM06a] B. Dongol and A. J. Mooij. Progress in deriving concurrent programs: Emphasizing the role of stable guards. In Tarmo Uustalu, editor, *8th International Conference on Mathematics of Program Construction*, volume 4014 of *LNCS*, pages 140–161. Springer, 2006.

[DM06b] B. Dongol and A. J. Mooij. Streamlining progress-based derivations of concurrent programs. Technical Report SSE-2006-06, University of Queensland, Australia, 2006.

[Don06] B. Dongol. Derivation of Java monitors. In *Australian Software Engineering Conference (ASWEC)*, pages 211–220. IEEE Computer Society, 2006.

[DS90] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics.* Springer-Verlag, 1990.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching time versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[FvG99] W. H. J. Feijen and A. J. M. van Gasteren. *On a Method of Multiprogramming.* Springer Verlag, 1999.

[GD05] D. Goldson and B. Dongol. Concurrent program design in the extended theory of Owicki and Gries. In M. Atkinson and F. Dehne, editors, *CATS*, volume 41 of *CRPIT*, pages 41–50. Australian Computer Society, 2005.

[GP89] R. Gerth and A. Pnueli. Rooting unity. In *Proceedings of the 5th International Workshop on Software Specification and Design*, pages 11–19, Pittsburgh, Pensylvania, USA, 1989. ACM Press.

[JKR89] C. S. Jutla, E. Knapp, and J. R. Rao. A predicate transformer approach to semantics of parallel programs. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, pages 249–263. ACM Press, 1989.

[Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[MP92] Z. Manna and A. Pnueli. *Temporal Verification of Reactive and Concurrent Systems: Specification.* Springer-Verlag New York, Inc., 1992.

[Nel89] G. Nelson. A generalization of Dijkstra's calculus. *ACM Trans. Program. Lang. Syst.*, 11(4):517–561, 1989.

[OG76] S. Owicki and D. Gries. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976.

[OL82] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982.

[Pac92] Jan K. Pachl. A simple proof of a completeness result for leads-to in the UNITY logic. *Inf. Process. Lett.*, 41(1):35–38, 1992.

[Plo04] Gordon D. Plotkin. The origins of structural operational semantics. *J. Log. Algebr. Program.*, 60-61:3–15, 2004.