Math. Struct. in Comp. Science (2010), *vol.* 20, *pp.* 865–914. © Cambridge University Press 2010
doi:10.1017/S0960129510000241

Axiomatic semantics of projection temporal logic programs[†]

XIAOXIAO YANG[‡], ZHENHUA DUAN[§] and QIAN MA[¶]

[‡]*Institute of Computing Theory and Technology, and ISN Laboratory, Xidian University, Xi'an, 710071 China and Laboratory for Computer Science, Institute of Software, Chinese Academy of Sciences, P.O. Box 8718, Beijing 100080 China*
Email: xxyang@ios.ac.cn
[§]*Institute of Computing Theory and Technology, and ISN Laboratory, Xidian University, Xi'an, 710071 China*
Email: zhhduan@mail.xidian.edu.cn
[¶]*Institute of Computing Theory and Technology, and ISN Laboratory, Xidian University, Xi'an, 710071 China*
Email: qma@mail.xidian.edu.cn

Received 5 November 2009; revised 1 May 2010

In this paper, we investigate the axiomatic semantics of the projection temporal logic programming language MSVL. To this end, we employ Propositional Projection Temporal Logic (PPTL) as an assertion language to specify the desired properties. We give a set of state axioms and state inference rules. In order to deduce a program over an interval, we also formalise a set of rules in terms of a Hoare logic-like triple. These rules enable us to deduce a program into its normal form and from the current state to the next one. They also enable us to verify properties over intervals. In this way, an axiom system for proving the correctness of MSVL programs is established. The axiom system is proved to be sound and relatively complete with respect to an operational model of MSVL, and give an example showing how the axiom system works. Finally, we employ a recently developed prototype verifier based on PVS as an example of semi-automatic verification using MSVL.

1. Introduction

Proving the correctness of software has long been a major challenge in computer science. Several approaches to formal verification have been proposed over the years (Clarke *et al.* 1999). However, verification has suffered from the disadvantage that different languages have been used for writing programs, for writing about their properties, and for writing about whether and how a program satisfies a given property (Lamport 1994). The transformation between different notations is a burden, so one way to simplify the problem is to use the same language, as far as possible.

Amir Pnueli was the first to suggest that temporal logic could be used for specifying and developing concurrent programs (Pnueli 1977; Manna and Pnueli 1992). Subsequently,

[†] This research is supported by the NSFC Grant Nos. 60433010, 60873018, 60910004, National Program on Key Basic Research Project (973 Program) Grant No. 2010CB328102 and the CAS Innovation Programs.
[‡] Xiaoxiao Yang is the corresponding author for this paper.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

several researchers proposed using temporal logic (or a subset of it) as the foundational basis for a programming language (Tang 1983; Lamport 1994; Fujita *et al.* 1986; Moszkowski 1986). Thus, writing programs, specifying their properties, and verifying those properties, could all be treated within the same notation.

Over the past two decades, a number of programming languages based on temporal logics have emerged. For instance:

— Tempura (Moszkowski 1986; Hale 1987) and Tokio (Fujita *et al.* 1986) are based on interval temporal logic;
— Cactus (Rondogiannis *et al.* 1997; 1998) is based on branching-time temporal logic;
— THLP (Wadge 1988), Chronolog (Orgun and Wadge 1988; 1992), Templog (Abadi and Manna 1989) and Temporal Prolog (Gabbay 1987) are based on linear-time temporal logic;
— (a different) Temporal Prolog (Hrycej 1988; 1993) is based on reified temporal logic;
— Brzoska has also proposed temporal logic programming based on metric temporal logic (Brzoska 1993; 1998).

In the field of interval temporal logic (ITL), Duan *et al.* (1994) introduced a new projection construct, $(p_1, \ldots, p_m)\,\mathsf{prj}\,q$, and generalised ITL to infinite time intervals. This new interval-based temporal logic was called Projection Temporal Logic (PTL) (Duan *et al.* 2008b; Duan *et al.* 1994). Tian and Duan (2008) proved that the expressiveness of propositional PTL is more powerful than propositional linear temporal logic (PLTL). Using PTL as a basis, Duan further developed an executable subset, called the Modelling Simulation and Verification Language (MSVL) (Duan *et al.* 2008b; Duan and Koutny 2004; Duan *et al.* 1994), which extends Tempura by adding infinite models, projection construct and framing technique.

The origins of most temporal logic programming languages lie in logic programming, so they have adopted a Prolog-like syntax and inherited the Prolog programming style. However, an interval temporal logic programming language, such as MSVL, has certain features in common with imperative programming. One can derive operators naturally from PTL that are very like the constructs of imperative programming. For instance, in MSVL, a repetition statement can be defined as

$$\mathit{while}\ b\ \mathit{do}\ p \stackrel{\text{def}}{=} (p \wedge b)^* \wedge \Box(\varepsilon \rightarrow \neg b)$$

where ε (empty), $\Box$ (always), $*$ (chop star) are temporal operators in PTL. All the PTL formulas (programs) are interpreted over intervals. Detailed explanations of these temporal operators will be given in Section 2. Other constructs such as sequence $(p\ ;\ q)$, selection ($\mathit{if}\ b\ \mathit{then}\ p\ \mathit{else}\ q$), concurrency $(p \wedge q)$ and parallel computation $(p \parallel q)$ can also be defined in MSVL. In addition, armed with the framing technique (Duan *et al.* 2008b), it becomes easy to define the communication synchronisation construct $\mathsf{await}(c)$ in MSVL. These constructs reduce the gap, at least to some degree, between temporal logic programming and imperative programming, and enable us to write temporal logic programs in a structured style.

The projection construct, $(p_1, \ldots, p_m)\,\mathsf{prj}\,q$, is another distinct feature of MSVL, which may have potential applications for compositional reasoning about concurrent systems.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

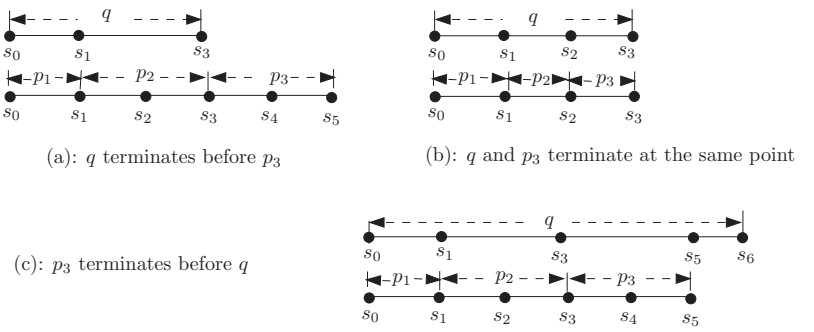


Fig. 1. Possible executions of (p_1, p_2, p_3) prj q

It can be thought of as a combination of the parallel and projection operators (p *proj* q) in Tempura. Intuitively, it means that q is executed in parallel with p_1 ; … ; p_m over an interval obtained by taking the endpoints of the intervals over which $p_1, \ldots, p_m$ are executed. The projection construct permits the processes $p_1, \ldots, p_m$, q to be autonomous, each process having the right to specify the interval over which it is executed. In particular, the sequence of processes $p_1, \ldots, p_m$ and the process q may terminate at different time points, as shown in Figure 1.

MSVL provides an executable PTL framework with a more succinct description and immediate practical applications. This then yields the possibility that specification and implementation can be realised in one notation, which will avoid time consuming and complex transformations between different formal languages. The goal of developing an approach to program verification using a single unified logic provided us with the motivation to investigate the axiomatic semantics of MSVL in this paper. Using this method, a system is first modelled using the MSVL language, and a property of the system is then specified using Propositional PTL (PPTL). In this way, the model and property of the system are both written using the same PTL logic.

The axiomatic semantics for MSVL can be formalised in two parts: one part for state deduction and the other for interval deduction. For state deductions, we give state axioms and inference rules for each basic statement so that we can reduce a program to its normal form. For interval deduction, we propose a variation of Hoare's triples, that is, $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$, to be used for correctness assertions. Here:

— A and σ_k are considered to be *left-conditions*, where A is a PPTL formula, and $\sigma_k = \langle s_0, \ldots, s_k \rangle$ $(k \geqslant 0)$ represents an interval over which some program q has been deduced for k states $s_0, \ldots, s_{k-1}$ $(k > 0)$ and indicates that the successor of q, program p, is being deduced at state s_k.

— B and σ_h are regarded as *right-conditions*, where B is a PPTL formula, and $\sigma_h = \langle s_0, \ldots, s_h \rangle$ presents the whole interval over which the original program q has been deduced for $h + 1$ states $s_0, \ldots, s_h$ $(h \geqslant 0)$, while program p has been deduced over $\langle s_k, \ldots, s_h \rangle$.

In fact, σ_k is a prefix of σ_h. During the deduction, if $\sigma_{(k \ldots h)}$ satisfies program p, then it should satisfy condition A and the final state s_h should satisfy condition B. The relationships

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

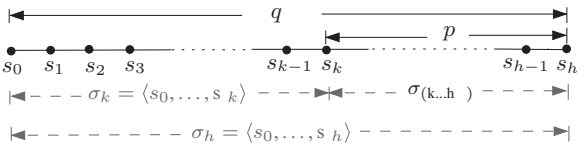


Fig. 2. The relationships between the components in triple $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$

between the components in the triple are illustrated in Figure 2. If p terminates, σ_h is a finite interval; otherwise, it is an infinite interval, denoted by σ_ω ($h = \omega$). When $k = 0$, we say $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$ is an *initial triple*, abbreviated to $\{A\}\ p\ \{B\}$, which means that program p is deduced over the interval σ_h from the initial state s_0. We say $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$ is a *general triple*, which means that program p is deduced over the interval $\sigma_{(k..h)}$ starting from some state s_k ($0 \leqslant k \leqslant h$)†. Using the general triple, we give a formal set of inference rules over intervals for deducing programs from one state to another.

The intuition lying behind Hoare's triple $\{A\}\ p\ \{B\}$ is 'If the assertion A is true before initiation of a program p, then the assertion B will be true on its completion' (Hoare 1969). However, within our proof system, triple $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$ means that if model $\sigma_{(k...h)}$ satisfies program p, then it satisfies property A (run-time properties), and when p terminates, the final state of model σ_h satisfies property B. We adopt an *on-the-fly* method for verifying the properties of programs. That is, the property is verified as programs are reduced to their normal forms by state axioms and state inferences. Equipped with the new triple, we can deal with the parallel ($p \parallel q$) and concurrent ($p \wedge q$) programs, and verify not only the partial correctness, but also run-time properties such as deadlock freeness and mutual exclusion. In addition, we will give a formal proof of the soundness and relative completeness of the system with respect to an operational model of MSVL. In this way, we show that the minimal model-theoretic (Duan *et al.* 2008b), axiomatic and operational semantics (Yang and Duan 2008) of MSVL programs are consistent.

The idea of encoding the semantics of a programming language in a logic in order to reduce verification to theorem proving was first formalised in Hoare (1969). Since then, various efforts have been made to generalise the classical Hoare verification approach to parallel programs. For instance, Owicki and Gries introduced a classical non-compositional method for parallel systems (Owicki 1975; Owicki and Gries 1976). Compositional methods based on the rely–guarantee approach (Misra and Chandy 1981) for concurrency verification have been worked on by a number of researchers, including C. B. Jones, Q. Xu and W. de Roever – see Jones (1981), Roever (1985) and Xu *et al.* (1997). However, these methods have been proposed for conventional imperative interleaving languages only, and not for the temporal logic programming languages that we are particularly interested in. Furthermore, compared with other interval-based logics, such as the Duration Calculus (*DC*) (Hansen and Zhao 1997), which is a tense interval temporal logic, PTL is a discrete interval temporal logic, and, more importantly, PTL can be executed.

The rest of this paper is organised as follows. In the next section, we present the syntax and semantics of PTL and give a brief introduction to the MSVL language. In

† $\sigma_{(0..h)}$ is identical to σ_h.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Section 3, we investigate an axiomatic semantics for MSVL, and then, in Section 4, prove the soundness and relative completeness of the system. In Section 5, we introduce a proof assistant developed in PVS for the axiom system, and present a mutual exclusion example. Finally, we present our conclusions in Section 6.

2. Preliminaries

2.1. *Projection temporal logic*

Syntax

Projection temporal logic (PTL) is a variation of interval temporal logic (ITL). The terms e and formulas p of PTL are as follows:

$$e ::= v \mid \bigcirc e \mid \ominus e \mid f(e_1, \ldots, e_n)$$

$$p ::= \pi \mid e_1 = e_2 \mid F(e_1, \ldots, e_n) \mid \neg p \mid p_1 \wedge p_2 \mid \exists v : p \mid \bigcirc p \mid (p_1, \ldots, p_m)\ \mathsf{prj}\ q \mid p^+$$

where v is a variable and π is a proposition. In $f(e_1, \ldots, e_n)$ and $F(e_1, \ldots, e_n)$, f is a function and F a predicate. $\bigcirc$ (*next*) and $\ominus$ (*previous*) are unitary temporal operators denoting future and past time, respectively. A formula (term) is said to be a state formula (term) if it does not contain any temporal operators (that is, $\bigcirc$, $\ominus$, prj); otherwise it is a temporal formula (term).

Semantics

Let $Prop$ be a set of propositions and V be a set of variables. $B = \{\mathsf{true}, \mathsf{false}\}$ represents the boolean domain. D denotes all data including integers, lists, sets, and so on. A *state* s is a pair of assignments (I_{var}, I_{prop}) that for each variable $v \in V$ defines $s[v] = I_{var}[v]$, and for each proposition $\pi \in Prop$ defines $s[\pi] = I_{prop}[\pi]$. Here $I_{var}[v]$ is a value within a data domain D or *nil* (undefined), while $I_{prop}[\pi] \in \{\mathsf{true}, \mathsf{false}\}$. An *interval* σ is a, possibly infinite, sequence of states. We define $|\sigma|$, the length of σ, to be ω if σ is infinite; otherwise it is the number of states in σ minus one. To ensure a uniform notation for both finite and infinite intervals, we will use *extended integers* as indices. That is, we consider the set N_ω consisting of the set N_0 of non-negative integers and ω, so $N_\omega = N_0 \cup \{\omega\}$, and extend the comparison operators, $=, <, \leqslant$, to N_ω by defining $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Moreover, we define $\preceq$ as $\leqslant - \{(\omega, \omega)\}$.

To define the semantics of the projection construct, we need an auxiliary operator. Let $\sigma = \langle s_0, s_1, \ldots \rangle$ be an interval and $r_1, \ldots, r_h$ be integers ($h \geqslant 1$) such that $0 \leqslant r_1 \leqslant \ldots \leqslant r_h \preceq |\sigma|$. Then

$$\sigma \downarrow (r_1, \ldots, r_h) \stackrel{\text{def}}{=} \langle s_{t_1}, s_{t_2}, \ldots, s_{t_l} \rangle$$

The *projection* of σ onto $r_1, \ldots, r_h$ is the interval (called the projected interval), where $t_1, \ldots, t_l$ are obtained from $r_1, \ldots, r_h$ by deleting all duplicates. In other words, $t_1, \ldots, t_l$ is the longest strictly increasing subsequence of $r_1, \ldots, r_h$. For example,

$$\langle s_0, s_1, s_2, s_3 \rangle \downarrow (0, 2, 2, 2, 3) = \langle s_0, s_2, s_3 \rangle.$$

So a new interval can be obtained in a convenient way using the $\downarrow$ operator by taking the endpoints of a projected interval over which $p_1, \ldots, p_m$ are interpreted in the projection

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

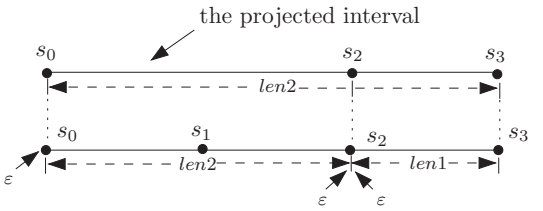


Fig. 3. A projected interval

$$
\begin{array}{lllll}
\textsc{Ivar} & \mathcal{I}[v] & = & s_k[v] = I^k_{var}[v] & \\
\textsc{Ifun} & \mathcal{I}[f(e_1,\ldots,e_m)] & = & \begin{cases} f(\mathcal{I}[e_1],\ldots,\mathcal{I}[e_m]) & \text{if } \mathcal{I}[e_h] \neq nil \text{ for all } h \\ nil & \text{otherwise} \end{cases} & \\
\textsc{Ienext} & \mathcal{I}[\bigcirc e] & = & \begin{cases} (\sigma,i,k+1,j)[e] & \text{if } k < j \\ nil & \text{otherwise} \end{cases} & \\
\textsc{Ieprev} & \mathcal{I}[\ominus e] & = & \begin{cases} (\sigma,i,k-1,j)[e] & \text{if } i < k \\ nil & \text{otherwise} \end{cases} &
\end{array}
$$

Fig. 4. Interpretation of PTL terms

construct. Figure 3 shows an example, where $\mathsf{len}(i)$ denotes the fact that the length of an interval is i and ε means the length of an interval is 0. The projected interval $\langle s_0, s_2, s_3\rangle$ can be obtained using the $\downarrow$ operator to take the endpoints of each formula $\varepsilon, \mathsf{len}(2), \varepsilon, \varepsilon, \mathsf{len}(1)$.

For a variable v, we will write $\sigma' \overset{v}{=} \sigma$ when σ' is an interval that is the same as σ except that different values can be assigned to v, and we will then say that σ and σ' are v-equivalent. The concatenation $(\cdot)$ and fusion $(\circ)$ operators are defined as follows:

— Concatenation $(\cdot)$:

$$
\sigma \cdot \sigma' = \begin{cases} \sigma & \text{if } |\sigma| = \omega \text{ or } \sigma' = \epsilon \\ \sigma' & \text{if } \sigma = \epsilon \\ \langle s_0,\ldots,s_i,s_{i+1},\ldots\rangle & \text{if } \sigma = \langle s_0,\ldots,s_i\rangle \text{ and } \\ & \sigma' = \langle s_{i+1},\ldots\rangle \end{cases}
$$

— Fusion $(\circ)$:

$$
\sigma \circ \sigma' = \begin{cases} \sigma & \text{if } |\sigma| = \omega \text{ or } \sigma' = \epsilon \\ \sigma' & \text{if } \sigma = \epsilon \\ \langle s_0,\ldots,s_i,\ldots\rangle & \text{if } \sigma = \langle s_0,\ldots,s_i\rangle \text{ and } \sigma' = \langle s_i,\ldots\rangle \end{cases}
$$

An *interpretation* for a PTL term or formula is a tuple $\mathcal{I} = (\sigma, i, k, j)$, where $\sigma = \langle s_0, s_1, \ldots\rangle$ is an interval, i and k are non-negative integers and j is an integer or ω such that $i \leqslant k \preceq j \leqslant |\sigma|$. We write (σ, i, k, j) to mean that a term or formula is interpreted over a subinterval $\sigma_{(i..j)}$ with the current state being s_k. We use I^k_{var} and I^k_{prop} to denote the state interpretation at state s_k. Each m-place function symbol f has an interpretation $\mathcal{I}[f]$, which is a function mapping m elements in D'^m to a single value in D'. Interpretations of predicate symbols are similar, but map to truth values. We assume that $\mathcal{I}$ gives standard interpretations to operators such as $+, -, *$ and $<, >, \leqslant, \geqslant, =$, and so on. The evaluation of e relative to interpretation $\mathcal{I} = (\sigma, i, k, j)$ is defined as $\mathcal{I}[e]$ in Figure 4. The satisfaction relation for formulas $\models$ is given in Figure 5.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

IPROP	$\mathcal{I}$	$\models$	π	if $s_k[\pi] = I^k_{prop}[\pi] = \mathsf{true}$
IPRED	$\mathcal{I}$	$\models$	$F(e_1, \ldots, e_m)$	if $F(\mathcal{I}[e_1], \ldots, \mathcal{I}[e_m]) = \mathsf{true}$, and $\mathcal{I}[e_h] \neq nil$ for all $1 \leq h \leq m$
IEQUAL	$\mathcal{I}$	$\models$	$e_1 = e_2$	if $\mathcal{I}[e_1] = \mathcal{I}[e_2]$
INEG	$\mathcal{I}$	$\models$	$\neg p$	if $\mathcal{I} \not\models p$
IAND	$\mathcal{I}$	$\models$	$p \wedge q$	if $\mathcal{I} \models p$ and $\mathcal{I} \models q$
INEXT	$\mathcal{I}$	$\models$	$\bigcirc p$	if $k < j$ and $(\sigma, i, k+1, j) \models p$
IEXISTS	$\mathcal{I}$	$\models$	$\exists v : p$	if $(\sigma', i, k, j) \models p$ for some $\sigma' \stackrel{v}{=} \sigma$
IPRJ	$\mathcal{I}$	$\models$	$(p_1, \ldots, p_m)$ prj q	if there are $k = r_0 \leq r_1 \leq \ldots \leq r_m \leq j$ such that $(\sigma, i, r_0, r_1) \models p_1$ and $(\sigma, r_{l-1}, r_{l-1}, r_l) \models p_l$ for all $1 < l \leq m$ and $(\sigma', 0, 0, \lvert\sigma'\rvert) \models q$ for σ' given by : (1) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_m+1..j)}$ (2) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_h)$ for some $0 \leq h \leq m$.
IPLUS	$\mathcal{I}$	$\models$	p^+	if there are finitely many $r_0, \ldots, r_n$ and $k = r_0 \leq r_1 \leq \ldots \leq r_{n-1} \preceq r_n = j$ $(n \geq 1)$ such that $(\sigma, i, r_0, r_1) \models p$ and $(\sigma, r_{l-1}, r_{l-1}, r_l) \models p$ for all $1 < l \leq n$; or $j = \omega$ and there are infinitely many integers $k = r_0 \leq r_1 \leq r_2 \leq \ldots$ such that $\lim_{i \to \infty} r_i = \omega$ and $(\sigma, i, r_0, r_1) \models p$ and for all $l > 1$, $(\sigma, r_{l-1}, r_{l-1}, r_l) \models p$.

Fig. 5. Interpretation of PTL formulas

If there is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models p$, then a formula p is *satisfiable*. We also define the satisfaction relation for an interval σ and formula p, by writing $\sigma \models p$ if $(\sigma, 0, 0, |\sigma|) \models p$. Furthermore, we write $\models p$ if $\sigma \models p$, for all intervals σ.

2.2. *Derived formulas*

Figure 6 defines the following useful formulas in terms of elementary PTL formulas:

— ε represents the final state;
— more specifies that the current state is a non-final state;
— $\Diamond p$ (pronounced *sometimes p*) means that p holds eventually in the future, which may include the current state;
— $\Box p$ (pronounced *always p*) means that p always holds in the future, from now on;
— $\bigodot p$ (*weak next*) means that either the current state is the final one or p holds at the next state of the present interval;
— p^* means that p is repeatedly executed a finite or infinite number of times;
— Prj$(p_1, \ldots, p_m)$ represents a *sequential* computation of $p_1, \ldots, p_m$ since the projected interval is a singleton;
— p ; q (p *chop* q) represents a computation of p followed by q, where the intervals for p and q share a common state. That is, p holds from now until some point in the future

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

$$
\begin{array}{lcl}
\varepsilon & \stackrel{\text{def}}{=} & \neg \bigcirc \mathsf{true} \\
\mathsf{len}(n) & \stackrel{\text{def}}{=} & \begin{cases} \varepsilon & \text{if } n = 0 \\ \bigcirc \mathsf{len}(n-1) & \text{if } n > 1 \end{cases} \\
p^* & \stackrel{\text{def}}{=} & p^+ \vee \varepsilon \\
\mathsf{skip} & \stackrel{\text{def}}{=} & \mathsf{len}(1) \\
\mathsf{Prj}(p_1, \ldots, p_m) & \stackrel{\text{def}}{=} & (p_1, \ldots, p_m)\ \mathsf{prj}\ \varepsilon \\
\mathsf{fin}(p) & \stackrel{\text{def}}{=} & \Box(\varepsilon \rightarrow p) \\
p\ ;\ q & \stackrel{\text{def}}{=} & \mathsf{Prj}(p, q) \\
\mathsf{keep}(p) & \stackrel{\text{def}}{=} & \Box(\neg\varepsilon \rightarrow p) \\
\mathsf{more} & \stackrel{\text{def}}{=} & \neg\varepsilon \\
\mathsf{halt}(p) & \stackrel{\text{def}}{=} & \Box(\varepsilon \leftrightarrow p) \\
\Diamond p & \stackrel{\text{def}}{=} & \mathsf{Prj}(\mathsf{true}, p) \\
\texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ q & \stackrel{\text{def}}{=} & (b \rightarrow p) \wedge (\neg b \rightarrow q) \\
\bigodot p & \stackrel{\text{def}}{=} & \varepsilon \vee \bigcirc p \\
\texttt{while}\ b\ \texttt{do}\ p & \stackrel{\text{def}}{=} & (p \wedge b)^* \wedge \Box(\varepsilon \rightarrow \neg b) \\
\Box p & \stackrel{\text{def}}{=} & \neg\Diamond\neg p \\
p \mid\mid q & \stackrel{\text{def}}{=} & (p \wedge (q\ ;\ \mathsf{true})) \vee (q \wedge (p\ ;\ \mathsf{true}))
\end{array}
$$

Fig. 6. Derived PTL formulas

and from then on q holds. Note that $p\ ;\ q$ is a strong chop, which always requires that p be true on some finite subinterval;

— $\mathsf{len}(n)$ specifies the distance n from the current state to the final state of an interval;

— skip means that the length of the interval is one unit of time;

— $\mathsf{fin}(p)$ is true provided p is true at the final state;

— $\mathsf{keep}(p)$ is true if p is true at every state except the final one;

— the formula $\mathsf{halt}(p)$ holds if and only if formula p is true in the final state.

The definitions of $\texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ q$, $\texttt{while}\ b\ \texttt{do}\ p$ and $p \mid\mid q$, with the obvious meanings, are also given in terms of our underlying logic in Figure 6.

We will sometimes use $p \equiv q$ to denote $\models \Box(p \leftrightarrow q)$ and $p \supset q$ to denote $\models \Box(p \rightarrow q)$. We call the former *strong equivalence* and the latter *strong implication*. In fact, $p \equiv q$ means that p and q have the same truth value in all states in every model. Similarly, $p \supset q$ means that $p \rightarrow q$ is true in all states in every model.

Theorem 1. Let σ be an interval. If $\sigma \models p$ and $\sigma \models p \rightarrow q$, then $\sigma \models q$.

The proof is straightforward – see Duan (1996). Theorem 1 states that for any interval σ satisfying p, that is, $\sigma \models p$, if we also have $\sigma \models p \rightarrow q$, then we can infer that the interval satisfies q.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

2.3. *The temporal logic programming language MSVL*

2.3.1. *Framing*. The introduction of a framing technique to temporal logic programming is motivated by both practical and theoretical considerations, by improving the efficiency of a program and synchronising communication for parallel processes. The meaning of the framing operator, denoted by frame, can be stated as follows: $\mathsf{frame}(x)$ *means that variable* x *always keeps its old value over an interval if no assignment to* x *is encountered.*

The crux of this is that it specifies how we perceive the assignment of values to variables. To identify an occurrence of an assignment to a variable, say x, we make use of a flag called the assignment flag, denoted by the predicate $\mathsf{af}(x)$, which is true whenever an assignment of a value to x is encountered, and false otherwise. Note that $\mathsf{af}(x)$ may not be used freely in a program but is only concerned with an assignment and a framing operator. Predicate $\mathsf{af}(x)$ is easy to understand, but difficult to formalise in a logic framework. The problem is that a program provides only positive information, that is, some explicit assignments from which we know those variables that are assigned explicitly within the program. However, what we need is negative information, that is, those variables that do not encounter assignments in the current state.

To do this, we first define a new assignment, which is required by framing, then we define framing operators, and, finally, we present a minimal model-based approach for framing.

Let $S_p = \{x_1, \ldots, x_n\} (S_p \subset V)$ be a set of state (dynamic) variables within a program p and $\Phi_p = \{p_{x_1}, \ldots, p_{x_n}\}$ be the set of propositions associated with state variables. We assume that a program p does not involve propositions other than Φ_p. A new assignment called *positive immediate assignment* is defined by

$$x \Leftarrow e \stackrel{\text{def}}{=} x = e \wedge p_x$$

where p_x is an atomic proposition connected with variable x and cannot be used for any other purpose. To identify an occurrence of an assignment to a variable, say x, we make use of a flag called *the assignment flag*, denoted by a predicate $\mathsf{af}(x)$, which is defined by

$$\mathsf{af}(x) \stackrel{\text{def}}{=} p_x$$

The predicate $\mathsf{af}(x)$ is associated with some assignment operator and can be used to determine whether or not such an assignment has taken place to x in the execution of a program. Whenever such an assignment is encountered, $\mathsf{af}(x)$ should be true. Conversely, when $\mathsf{af}(x)$ is true, such an assignment should have been perceived in the execution of the program. As expected, when $x \Leftarrow e$ is encountered, p_x is set to true, hence $\mathsf{af}(x)$ is true; whereas if no assignment to x takes place, p_x is unspecified. In this case, we will use a minimal model to force it to be false.

Armed with the assignment flag, we can define state framing and interval framing operators. Intuitively, when a variable is framed at a state, its value remains unchanged if no assignment is encountered at that state. A variable is framed over an interval if it is framed at every state over the interval. We formalise this idea in Definition 1.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

idempotent	$\mathsf{frame}(x) \wedge \mathsf{frame}(x)$	$\equiv$	$\mathsf{frame}(x)$
distributive	$\mathsf{frame}(x) \wedge p \vee \mathsf{frame}(x) \wedge q$	$\equiv$	$\mathsf{frame}(x) \wedge (p \vee q)$
absorptive	$\mathsf{frame}(x) \wedge (\mathsf{frame}(x) \wedge p \vee q)$	$\equiv$	$\mathsf{frame}(x) \wedge (p \wedge q)$
	$\mathsf{frame}(x) \wedge (p \vee \mathsf{frame}(x) \wedge q)$	$\equiv$	$\mathsf{frame}(x) \wedge (p \vee q)$
equivalence	$\mathsf{frame}(x) \wedge \mathsf{more}$	$\equiv$	$\bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x))$
	$\mathsf{frame}(x) \wedge \varepsilon$	$\equiv$	ε

Table 1. *Some algebraic properties of framing operators*

Definition 1 (Looking back framing).

$$\mathsf{lbf}(x) \stackrel{\text{def}}{=} \neg\mathsf{af}(x) \rightarrow \exists b : (\ominus x = b \wedge x = b)$$
$$\mathsf{frame}(x) \stackrel{\text{def}}{=} \Box(\mathsf{more} \rightarrow \bigcirc\mathsf{lbf}(x))$$

where b is a static variable.

Theorem 2. The following laws hold:
(1) $x = e \equiv (p_x \wedge x = e) \vee (\neg p_x \wedge x = e)$.
(2) $\mathsf{lbf}(x) \equiv p_x \vee (\neg p_x \wedge (x = \ominus x))$.
(3) $\mathsf{lbf}(x) \wedge x = e \equiv x \Leftarrow e$, where $\ominus x \neq e$.
(4) $\mathsf{lbf}(x) \wedge x \Leftarrow e \equiv x \Leftarrow e$.

Proof. We will only prove (3); the other proofs can be found in Duan *et al.* (2008b) and Duan (1996).

$$\begin{aligned}
\mathsf{lbf}(x) \wedge x = e &\equiv \mathsf{lbf}(x) \wedge (x = e \wedge (p_x \vee \neg p_x)) \\
&\equiv (p_x \vee (\neg p_x \wedge x = \ominus x)) \wedge ((x = e \wedge p_x) \vee (x = e \wedge \neg p_x)) \\
&\equiv (p_x \wedge x = e) \vee (p_x \wedge (x = e \wedge \neg p_x)) \vee \\
&\quad ((\neg p_x \wedge x = \ominus x) \wedge (x = e \wedge p_x)) \vee \\
&\quad ((\neg p_x \wedge x = \ominus x) \wedge (x = e \wedge \neg p_x)) \\
&\equiv (p_x \wedge x = e) \vee ((\neg p_x \wedge x = \ominus x) \wedge (x = e \wedge \neg p_x)) \\
&\equiv p_x \wedge x = e \qquad (\text{since } \ominus x \neq e) \\
&\equiv x \Leftarrow e. \qquad \Box
\end{aligned}$$

The equivalent laws in Theorem 2 are useful to reduce assignment statements. Table 1 lists some interesting algebraic properties of framing operators, including the idempotent, distributive, absorptive and equivalence laws.

Introducing the framing operator enables us not only to write concise programs but also to define an await construct within the underlying logic. Hence, synchronised communication can be handled within MSVL. The await statement can be defined as follows:

$$\mathsf{await}(c) \stackrel{\text{def}}{=} \mathsf{frame}(x_1) \wedge \ldots \wedge \mathsf{frame}(x_h) \wedge \mathsf{halt}(c)$$

where c is a condition, that is, a boolean expression, and $x_1, \ldots, x_h$ are dynamic variables contained in c.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Termination:	ε
Assignment:	$x = e$
Positive Immediate Assignment:	$x \Leftarrow e$
State Frame:	$\mathsf{lbf}(x)$
Interval Frame:	$\mathsf{frame}(x)$
Conjunction:	$p \wedge q$
Selection:	$p \vee q$
Next statement:	$\bigcirc p$
Always statement:	$\Box p$
Conditional statement:	`if` b `then` p `else` q
Existential quantification:	$\exists x : p(x)$
Sequential statement:	p `;` q
While statement:	`while` b `do` p
Parallel statement:	$p \parallel q$
Projection statement:	$(p_1, \ldots, p_m)$ prj q
Synchronised Communication:	$\mathsf{await}(c)$

Fig. 7. MSVL programs

2.3.2. *Modelling, Simulation and Verification Language.* MSVL provides an executable PTL framework with a succinct description that can be used for modelling simulation and verification of hardware and software systems. In MSVL, expressions can be regarded as PTL terms and statements as PTL formulas. Arithmetic expressions e and boolean expressions b in MSVL are defined by the grammar

$$e ::= n \mid x \mid \bigcirc x \mid \ominus x \mid e_0 \; op \; e_1 \quad (op ::= + \mid - \mid * \mid mod)$$
$$b ::= \mathsf{true} \mid \mathsf{false} \mid \neg b \mid b_0 \wedge b_1 \mid e_0 = e_1 \mid e_0 < e_1$$

where n is an integer and x is a variable. A dynamic variable x is said to be framed in program p if $\mathsf{frame}(x)$ or $\mathsf{lbf}(x)$ is contained in p.

A framed program in MSVL can be defined inductively using Figure 7, where x denotes a variable, e denotes an arbitrary arithmetic expression, b denotes a boolean expression, and $p, p_1, \ldots, p_m$ and q are programs.

When proving a program property inductively, the assignment $x = e$, positive immediate assignment $x \Leftarrow e$, state frame $\mathsf{lbf}(x)$ and ε can be thought of as basic statements and the others can be treated as composite statements:

— The termination statement ε simply means that the current state is the final state of the interval over which a program is executed.
— The 'next' statement $\bigcirc p$ means that p holds at the next state.
— $\Box p$ means that p holds in all the states from now on.
— The sequential statement p `;` q means that p holds from now until some point in the future, and from that point on, q holds.
— The conditional statement `if` b `then` p `else` q first evaluates the boolean expression, and if b is true, the process p is executed, otherwise q is executed.
— The iteration `while` b `do` p allows process p to be repeatedly executed a finite (or infinite) number of times over a finite (respectively, infinite) interval as long as the condition b holds at the beginning of each execution, but if b becomes false, the `while` statement terminates.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

— The conjunction statement $p \wedge q$ means that the processes p and q are executed concurrently, sharing all their states and variables during the mutual execution.
— The parallel construction $p \parallel q$ presents an alternative form of concurrent computation. The difference between $p \parallel q$ and $p \wedge q$ is that the former allows both processes p and q to specify their own intervals, while the latter does not. For example, $\mathsf{len}(2) \parallel \mathsf{len}(3)$ holds, but $\mathsf{len}(2) \wedge \mathsf{len}(3)$ is obviously false.
— The selection statement $p \vee q$ means p or q are executed.
— Projection can be thought of as a special parallel computation that is executed on different time scales. The projection $(p_1, \ldots, p_m)$ prj q means that q is executed in parallel with $p_1, \ldots, p_m$ over an interval obtained by taking the endpoints of the intervals over which the p_i's are executed. In particular, the sequence of p_i's and q may terminate at different points.
— The synchronisation construct $\mathsf{await}(c)$ is used to synchronise communication between parallel processes in a concurrent program with the shared variable model. $\mathsf{await}(c)$ does not change any variable, but waits until the condition c becomes true, at which point it terminates.

We use a renaming method to reduce a program that has existential quantification. Given a formula $\exists x : p(x)$ with a bound variable x, we can remove the existential quantification $(\exists x)$ from $\exists x : p(x)$ to obtain a formula $p(y)$ with a free variable y by renaming x as y. To do this, we require that:

(1) y does not occur (free or bound) in the whole program such as $(q \wedge \exists x : p(x))$;
(2) y and x are either both dynamic or static;
(3) y substitutes for x only within the bound scope of x in $\exists x : p(x)$.

In this case, we say $p(y)$ is a *renamed formula* of $\exists x : p(x)$. We will now discuss some facts about renamed formulas.

Lemma 3. Let $p(y)$ be a renamed formula of $\exists x : p(x)$. Then $\exists x : p(x)$ is satisfiable if and only if $p(y)$ is satisfiable. Furthermore, any model of $p(y)$ is a model of $\exists x : p(x)$.

The importance of Lemma 3 is that it guarantees the soundness of the renaming method for reducing programs involving existential quantifications. From Lemma 3, it is clear that $\exists x : p(x)$ and $p(y)$ are equivalent in satisfiability. To check whether or not $\exists x : p(x)$ is satisfiable amounts to checking whether or not $p(y)$ is satisfiable. Once we find a model for $p(y)$, this model is also a model for $\exists x : p(x)$. Therefore, to reduce $\exists x : p(x)$, it is sufficient to reduce $p(y)$.

Definition 2. For a variable x, we say $\exists x : p(x)$ and $p(x)$ are x-equivalent, denoted by $\exists x : p(x) \overset{x}{\equiv} p(x)$, if for each model $\sigma \models \exists x : p(x)$, there exists some σ', such that $\sigma \overset{x}{=} \sigma'$ and $\sigma' \models p(x)$ and *vice versa.*

Definition 2 tells us that the models of $\exists x : p(x)$ and $p(x)$ are the same if we ignore the variable x. Thus, we denote this special equivalence relation between programs by $\overset{x}{\equiv}$, which is the counterpart of the equivalence relation $\overset{x}{=}$ over the intervals. Notice that $\exists x : p(x) \equiv \exists y : p(y)$, and by Definition 2, $\exists y : p(y) \overset{y}{\equiv} p(y)$. So we say that $\exists x : p(x) \overset{y}{\equiv} p(y)$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

if $p(y)$ is a renamed program of $\exists x : p(x)$, and the model of $\exists x : p(x)$ can be obtained by hiding the variable y in the model of $p(y)$.

A program can be terminating or non-terminating. If a program is terminating (non-terminating), it can be executed over a finite (infinite) interval. The language provides some statements to specify the length of an interval. In fact, the primitive statement ε and the derived constructs $\mathsf{len}(k)$, skip and $\mathsf{halt}(k)$ are used for this purpose.

2.4. *Minimal model semantics of framed programs*

We use canonical models to capture the semantics of non-framed programs in Tempura (Moszkowski 1986). In the following, we will give a brief introduction to canonical models. Suppose a program p contains a finite set S_p of variables and a finite set Φ_p of propositions. A canonical interpretation on propositions is a subset $\mathcal{I}_{prop} \subset \Phi_p$. Implicitly, propositions not in $\mathcal{I}_{prop}$ are false. Let $\sigma = \langle (I^0_{var}, I^0_{prop}), \ldots \rangle$ be a model. We denote the sequence of interpretation on propositions of σ by $\sigma_{prop} = \langle I^0_{prop}, \ldots \rangle$. If there exists a model σ with σ_{prop} canonical and $\sigma \models p$ as in the logic, then the program p is *satisfiable* under the canonical interpretation on propositions. We denote this by $\sigma \models_c p$, and call σ_{prop} a canonical interpretation sequence (on propositions) of p. If $\sigma \models p$ for all σ with canonical σ_{prop}, then p is *valid* under the canonical interpretation on propositions. We denote this by $\models_c p$. Note that the definition of a canonical interpretation of a program is independent of its syntax in the sense that it does not refer to the program's structure. Hence it can also be applied to temporal formulas.

Definition 3. Let p be a framed program in MSVL and $\Sigma_p = \{\sigma | \sigma \models_c p\}$. Let $\sigma_1, \sigma_2 \in \Sigma_p$. We define

— $\sigma_{prop} \sqsubseteq \sigma'_{prop}$ if and only if $|\sigma| = |\sigma'|$ and $I^i_{prop} \subseteq I'^i_{prop}$, for all $0 \leqslant i \leqslant |\sigma|$.
— $\sigma \sqsubseteq \sigma'$ if and only if $\sigma_{prop} \sqsubseteq \sigma'_{prop}$.
— $\sigma \sqsubset \sigma'$ if and only if $\sigma \sqsubseteq \sigma'$ and $\sigma' \not\sqsubseteq \sigma$.
— $\sigma_1 \doteq \sigma_2$ if and only if $\sigma_1 \sqsubseteq \sigma_2$ and $\sigma_2 \sqsubseteq \sigma_1$.

For instance, $\langle (\varnothing, \{x{:}1\}) \rangle \sqsubset \langle (\{p_x\}, \varnothing) \rangle$, $\langle (\{p_x\}, \{x{:}1\}) \rangle \doteq \langle (\{p_x\}, \{x{:}2\}) \rangle$.

Since a program can be satisfied by several different canonical models, we need to choose a model carefully so that it reflects its intended meaning. We now formulate the following definition.

Definition 4. Let p be an MSVL program and $\mathcal{I} = (\sigma, i, k, j)$ be a canonical interpretation. Then the minimal satisfaction relation $\models_m$ is defined as
$(\sigma, i, k, j) \models_m p$ if and only if $(\sigma, i, k, j) \models_c p$ and there is no σ' such that $\sigma' \sqsubset \sigma$ and $(\sigma', i, k, j) \models_c p$.

A *minimal model* of a program p is a canonical model σ such that $(\sigma, 0, 0, |\sigma|) \models_m p$. We denote this by $\sigma \models_m p$. Moreover, the equivalence relations $\equiv_m$ as well as the strong implication relation $\supset_m$ can be defined similarly to the relations $\equiv$ and $\supset$.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Definition 5. An MSVL program q is in *normal form* if

$$q \stackrel{\text{def}}{=} \left(\bigvee_{i=1}^{k} q_{ei} \wedge \varepsilon\right) \vee \left(\bigvee_{j=1}^{h} q_{cj} \wedge \bigcirc q_{fj}\right)$$

where $k + h \geqslant 1$ and the following hold:

— q_{fj} is an *internal program*, that is, one in which variables may refer to the previous states but not beyond the first state of the current interval over which the program is executed;

— each q_{cj} and q_{ei} is either true or a state formula of the form $p_1 \wedge \ldots \wedge p_m$ $(m \geqslant 1)$ such that each p_l $(1 \leqslant l \leqslant m)$ is either $(x = e)$ with $e \in D, x \in V$, or p_x or $\neg p_x$.

A key conclusion is that any MSVL program can be reduced to its normal form. Therefore, one way to execute programs in MSVL is to transform them so that they are logically equivalent to their normal forms (Duan *et al.* 2008b).

Theorem 4. For any MSVL program *prog* there is an MSVL program *prog*′ in the normal form such that

$$prog \equiv prog'.$$

The proof can be found in Duan *et al.* (2008b).

Example 1. Consider the program

$$\textsc{Prog} \stackrel{\text{def}}{=} \mathsf{frame}(x) \wedge (x = 1) \wedge \mathsf{len}(1).$$

We reduce it as follows:

$$\begin{aligned}
\mathsf{frame}(x) \wedge (x = 1) \wedge \mathsf{len}(1) &\equiv \mathsf{frame}(x) \wedge (x = 1) \wedge \bigcirc\varepsilon \\
&\equiv \mathsf{frame}(x) \wedge (x = 1) \wedge \bigcirc\varepsilon \wedge \mathsf{more} \\
&\equiv \bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x)) \wedge (x = 1) \wedge \bigcirc\varepsilon \qquad \text{(Table 1)} \\
&\equiv x = 1 \wedge \bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x) \wedge \varepsilon).
\end{aligned}$$

Then the normal form of this program is

$$\textsc{Prog} \equiv q_{c1} \wedge \bigcirc q_{f1}$$

where

$$\begin{aligned}
q_{c1} &\equiv (x = 1) \\
&\equiv (x = 1 \wedge p_x) \vee (x = 1 \wedge \neg p_x) \qquad \text{(Theorem 2)}
\end{aligned}$$

$$\begin{aligned}
q_{f1} &\equiv \mathsf{lbf}(x) \wedge \mathsf{frame}(x) \wedge \varepsilon \\
&\equiv \mathsf{lbf}(x) \wedge \varepsilon \qquad \text{(Table 1)} \\
&\equiv ((x = \ominus x \wedge \neg p_x) \vee p_x) \wedge \varepsilon \qquad \text{(Theorem 2)} \\
&\equiv (x = 1 \wedge \neg p_x \wedge \varepsilon) \vee (p_x \wedge \varepsilon).
\end{aligned}$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

This program is executed over an interval with two states, $\sigma = \langle s_0, s_1 \rangle$. Conventionally, at each state, we use the pair $(\{x : e\}, \{p_x\})$ to express the state formula $x = e \wedge p_x$. And when there is no variable or proposition at a state, it is denoted by $\varnothing$. Hence, four canonical models can satisfy the program PROG:

$$\sigma_1 = \langle (\{x{:}1\}, \{p_x\}), (\varnothing, \{p_x\}) \rangle$$
$$\sigma_2 = \langle (\{x{:}1\}, \{p_x\}), (\{x{:}1\}, \varnothing) \rangle$$
$$\sigma_3 = \langle (\{x{:}1\}, \varnothing), (\varnothing, \{p_x\}) \rangle$$
$$\sigma_4 = \langle (\{x{:}1\}, \varnothing), (\{x{:}1\}, \varnothing) \rangle.$$

However, the intended meaning of this program is captured by its minimal model σ_4. For this model, x is defined as 1 in both states of the interval.

3. Axiomatic semantics

In this section, we investigate the axiomatic semantics of MSVL, by:

(1) Using the temporal logic language Propositional PTL (PPTL) as an assertion language for specifying the desired properties (Duan *et al.* 2008a).
(2) Giving state axioms and state inference rules to reduce an MSVL program to normal form.
(3) Formalising a set of interval axioms and inference rules to deduce a program over an interval – these rules enable us to transform a program from the current state to the next one, and simultaneously verify properties over intervals.

In this way, we will establish an axiom system for MSVL, which will be denoted by Π_{MSVL}.

Typically, a Hoare proof system uses extended boolean expressions with quantifiers to describe the pre- and post-conditions for sequential programs. However, in order to state or prove the properties of concurrent programs, it is often necessary to deal with the entire execution sequence. Therefore, to specify run-time properties like *eventually* or *never*, we will employ PPTL as the assertion language, which means, for example, that we will be able to specify *eventually* or *never* using the temporal operators $\diamond$ and $\Box$, respectively. In addition, PPTL has greater expressive power when compared with propositional linear temporal logic (PLTL). For instance, the property that proposition P (a PPTL formula) must hold at even states regardless of the odd states can be specified as $P \wedge (\bigcirc \bigcirc (P \wedge \varepsilon))^*$ in PPTL, but cannot be expressed in PLTL (Wolper 1983).

Note that, in our axiom system, we are only concerned with the verification of a satisfiable program, and do not consider the case of $p \equiv \mathsf{false}$. Hence, we claim that all of the programs appearing in the following context are satisfiable.

3.1. *Axioms and inference rules within a state*

We will first define a normal form for MSVL programs in the axiom system Π_{MSVL}, and then establish the fundamental property that each MSVL program can be transformed into its normal form by state axioms and state inference rules. For convenience, we will use $p \cong q$ to denote $\vdash \Box(p \leftrightarrow q)$.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

(A1)	$\mathsf{lbf}(x) \wedge x = e \cong x = e \wedge p_x$ (where $\ominus x \neq e$)
(A2)	$\mathsf{lbf}(x) \wedge x \Leftarrow e \cong x = e \wedge p_x$
(A3)	$\Box p \wedge \mathsf{more} \cong p \wedge \bigcirc \Box p$
(A4)	$\Box p \wedge \varepsilon \cong p \wedge \varepsilon$
(A5)	$\mathsf{frame}(x) \wedge \mathsf{more} \cong \bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x))$
(A6)	$\mathsf{frame}(x) \wedge \varepsilon \cong \varepsilon$
(A7)	$\bigcirc p \,;\, q \cong \bigcirc(p \,;\, q)$
(A8)	$\varepsilon \,;\, q \cong q$
(A9)	$(w \wedge p) \,;\, q \cong w \wedge (p \,;\, q)$
(A10)	$p \parallel q \cong ((p \,;\, \mathsf{true}) \wedge q) \vee ((q \,;\, \mathsf{true}) \wedge p)$
(A11)	$\bigcirc p \wedge q \cong \bigcirc p \wedge q \wedge \mathsf{more}$
(A12)	$\varepsilon \,\mathsf{prj}\, q \cong q$
(A13)	$(p_1, \ldots, p_m)\,\mathsf{prj}\, \varepsilon \cong p_1 \,;\, \ldots \,;\, p_m$
(A14)	$(p_1, \ldots, p_{i-1}, w \wedge \varepsilon, p_{i+1}, \ldots, p_m)\,\mathsf{prj}\, q$ $\cong (p_1, \ldots, p_{i-1}, w \wedge p_{i+1}, \ldots, p_m)\,\mathsf{prj}\, q$
(A15)	$(w \wedge p_1, p_2, \ldots, p_m)\,\mathsf{prj}\, q \cong w \wedge (p_1, \ldots, p_m)\,\mathsf{prj}\, q$
(A16)	$(p_1, \ldots, p_m)\,\mathsf{prj}\, (w \wedge q) \cong w \wedge (p_1, \ldots, p_m)\,\mathsf{prj}\, q$
(A17)	$(\bigcirc p_1, \ldots, p_m)\,\mathsf{prj}\, \bigcirc q \cong \bigcirc(p_1 \,;\, (p_2, \ldots, p_m)\,\mathsf{prj}\, q)$
(A18)	$\mathtt{if}\ b\ \mathtt{then}\ p\ \mathtt{else}\ q \cong (b \wedge p) \vee (\neg b \wedge q)$
(A19)	$\mathtt{while}\ b\ \mathtt{do}\ p \cong \mathtt{if}\ b\ \mathtt{then}\ (p \wedge \mathsf{more} \,;\, \mathtt{while}\ b\ \mathtt{do}\ p)\ \mathtt{else}\ \varepsilon$
(A20)	$\vdash P$, where P is a substitution instance of all valid formulas.

Table 2. *State axioms: in axiom (A8), q is a lec-formula; in axioms (A7, A9, A19), p is required to be a terminating program; in axioms (A9, A14, A15, A16), w is a present component*

Definition 6. Let q be a program in MSVL. The normal form of the program q in Π_{MSVL} is defined as

$$q \cong \bigvee_{i=1}^{k} (q_{ei} \wedge \varepsilon) \vee \bigvee_{j=1}^{h} (q_{cj} \wedge \bigcirc q_{fj})$$

where $k + h \geqslant 1$ and q_{fj} is a general program in MSVL, while q_{ei} and q_{cj} are either true or *state formulas* of the form

$$(x_1 = e_1) \wedge \ldots \wedge (x_l = e_l) \wedge \dot{p}_{x_1} \wedge \ldots \wedge \dot{p}_{x_l},$$

where $e_k \in D$ $(1 \leqslant k \leqslant D)$ and $\dot{p}_x$ denotes p_x or $\neg p_x$.

The conjuncts $q_{ei} \wedge \varepsilon$ and $q_{cj} \wedge \bigcirc q_{fj}$ are called *basic terminal products* and *basic future products*, respectively. Also, q_{cj} and q_{ei} are called *present components* and $\bigcirc q_{fj}$ are called *future components*. We write $q_{fj} \ll q$ to mean q_{fj} is an *immediate successor* of program q. If $q_1 \ll q_2 \ll \ldots \ll q_n \ll q$, we say q_i $(1 \leqslant i < n)$ is a *successor* of q.

Normal forms play a significant role for deducing programs in MSVL. Tables 2 and 3 list the set of state axioms and state inference rules, respectively, for deducing a program to its normal form. In Table 2:

— Axioms (A1) and (A2) deal with the *assignment statements* $x = e$ and $x \Leftarrow e$. When they are conjuncted with the *state frame* $\mathsf{lbf}(x)$, they are deduced to $x \Leftarrow e$. Note that in (A1), e is a new value different from the previous value x.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

(R1)	$p \cong q \Longrightarrow prog[p] \cong prog[q/p]$
(R2)	$p(x) \cong (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x))$ $\Longrightarrow \exists x : p(x) \cong (\exists x : p_e(x) \wedge \varepsilon) \vee (\exists x : p_c(x) \wedge \bigcirc \exists x : p_f(x))$ where $p_e(x)$ and $p_c(x)$ are present components.
(R3)	$\vdash P \Longrightarrow \vdash \Box P$, where P is a substitution instance of all valid formulas.

Table 3. *State inference rules*

— Axioms (A3) and (A4) deal with $\Box p$, and axioms (A5) and (A6) deal with the *interval frame* $\mathsf{frame}(x)$. These are described in terms of the interval length: ε (that is, $\mathsf{len}(0)$) or more (that is, $\mathsf{len}(n), n > 0$).
— Axioms (A7–A9) deal with the sequential statement p ; q.
— Axiom (A10) deals with the parallel statement $p \parallel q$ in terms of its definition.
— Axiom (A11) deals with $\bigcirc p$.
— Axioms (A12–A17) deal with the projection statement $(p_1, \ldots, p_m)\ \mathsf{prj}\ q$.
— Axioms (A18) and (A19) deal with the conditional and while statements, respectively.
— Finally, axiom (A20) is a basic axiom that describes the substitution instances of all valid classical first-order formulas into temporal contexts.

In addition, we will also consider some algebraic properties of temporal operators as axioms for use with (A20). For instance, temporal operators ($\bigcirc$) and (;) have the *distributive* property over $\wedge$ and $\vee$, that is, p ; $(t \wedge r) \cong (p$; $t) \wedge (p$; $r)$; $\bigcirc p \wedge \bigcirc q \cong \bigcirc(p \wedge q)$.

In Table 3:

— Rule (R1) is a substitution rule, where $prog[p]$ is a program in MSVL involving a subprogram p, and $prog[q/p]$ denotes the program given by replacing some occurrences of p in $prog$ by q.
— Rule (R2) deals with the existential quantifier. Provided $p(x)$ has a normal form, $\exists x : p(x)$ can be reduced to normal form.
— Rule (R3) says that if $\vdash P$, then P always holds over intervals, that is, $\vdash \Box P$.

Rule (R3) in association with axiom (A20) can be a great help when deducing programs. For instance, $\vdash \psi \vee \neg\psi \leftrightarrow \mathsf{true}$, where ψ is a formula in classical first-order logic. By axiom (A20), we have $\vdash \varepsilon \vee \neg\varepsilon \leftrightarrow \mathsf{true}$ replacing ψ by temporal formula ε. Then, by rule (R3), $\vdash \Box(\varepsilon \vee \neg\varepsilon \leftrightarrow \mathsf{true})$, that is, $\varepsilon \vee \neg\varepsilon \cong \mathsf{true}$.

The following simple example illustrates the use of state axioms and state inference rules – rule (R1) is used by default.

Example 2.

$$
\begin{aligned}
&(x = 1 \wedge \mathsf{len}(1))\ ;\ (x \Leftarrow 5 \wedge y = 1 \wedge \mathsf{skip}) \\
&\quad \cong x = 1 \wedge (\mathsf{len}(1)\ ;\ (x \Leftarrow 5 \wedge y = 1 \wedge \mathsf{skip})) && \text{(A9)} \\
&\quad \cong x = 1 \wedge (\bigcirc\varepsilon\ ;\ (x \Leftarrow 5 \wedge y = 1 \wedge \mathsf{skip})) && (\mathsf{len}(n) \text{ is defined in Section 2.2}) \\
&\quad \cong x = 1 \wedge \bigcirc(\varepsilon\ ;\ (x \Leftarrow 5 \wedge y = 1 \wedge \mathsf{skip})) && \text{(A7)} \\
&\quad \cong x = 1 \wedge \bigcirc(x \Leftarrow 5 \wedge y = 1 \wedge \mathsf{skip}) && \text{(A8)}
\end{aligned}
$$

It is now in normal form.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Theorem 5. Any MSVL program p can be reduced to its normal form using the state axioms and state inference rules in Tables 2 and 3.

Theorem 5 is one of our most important results as it means MSVL programs can be handled in normal form. In order to state the correctness of the state axioms and state inference rules, we will need to prove two more important theorems: soundness and completeness. Recall that, in the model theory of MSVL, programs p and q are strong equivalent if $\models \Box(p \leftrightarrow q)$, which is denoted by $p \equiv q$.

Theorem 6 (Soundness of the state axioms and state inference rules). Let p and q be MSVL programs. Then $p \cong q \Longrightarrow p \equiv q$.

Theorem 7 (Completeness of state axioms and state inference rules). Let p and q be MSVL programs. Then $p \equiv q \Longrightarrow p \cong q$.

Theorem 6 says that if one can deduce $p \cong q$, then this is also true semantically, namely, $p \equiv q$. Theorem 7 ensures that, if two programs p and q are strong equivalent ($p \equiv q$) in the model theory, then they are also provably equivalent ($p \cong q$) in the axiomatic system. Note that in order to prove completeness, we will need to confine variables to a finite domain, *viz*., $D = \{d_1, \ldots, d_n\} (n \geqslant 1)$.

These theorems will be proved in Section 4 and Appendix A.

3.2. *Axioms and inference rules over intervals*

This section is devoted to interval deduction. To this end, we define a modification of the Hoare triple as a correctness assertion, and formalise a set of axioms and inference rules over intervals to transform a program in normal form from the current state to the next one, and simultaneously verify the properties over intervals.

Recall that an interval σ is a non-empty sequence of states $\langle s_0, s_1, \ldots \rangle$, over which programs are deduced. In fact, the implementation of programs going from the current state, say s_k, to the next state, s_{k+1}, depends strongly on the temporal operators '*next*' ($\bigcirc$) and '*empty*' (ε). In order to capture the semantics of $\bigcirc p$ and ε by means of our axiomatic system, the following auxiliary notation is required. Let Σ be a set of intervals and σ_k denote $\langle s_0, \ldots, s_k \rangle$, which records the states passed through in going from the initial state s_0 to the current state s_k, at which a program is being deduced. The length of σ_k is $|\sigma_k| = k$, where k is a counter counting the number of states in σ_k. The correctness assertion for specifying and verifying properties is then given as follows.

Definition 7. The correctness assertion can be defined as

$$\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$$

where p is an MSVL program, σ_k and σ_h are intervals, with σ_k a prefix of σ_h ($k, h \in N_\omega, 0 \leqslant k \leqslant h$), and A and B are PPTL formulas that we assume are used in the following manner:

(1) For partial correctness properties, A and B are required to be PPTL *state formulas.*
(2) For run-time properties, such as the *eventually* ($\Diamond$) and *always* ($\Box$) properties, A is a PPTL *temporal formula*, and B is *a state formula.*
(3) If p is non-terminating, B is required to be true.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Here:

— A and σ_k are considered to be *left-conditions*, where A is a PPTL formula, and $\sigma_k = \langle s_0, \ldots, s_k \rangle$ represents an interval over which some program q has been deduced for k states $s_0, \ldots, s_{k-1}$ $(k > 1)$, and also indicates that the successor of q, program p, is being deduced at state s_k.

— B and σ_h are regarded as *right-conditions*, where B is a PPTL formula, and $\sigma_h = \langle s_0, \ldots, s_h \rangle$ presents the whole interval over which the original program q has been deduced for $h + 1$ states $s_0, \ldots, s_h$ $(h \geqslant 0)$, while program p has been deduced over $\langle s_k, \ldots, s_h \rangle$.

In fact, σ_k is a prefix of σ_h. During the deduction, if $\sigma_{(k \ldots h)}$ satisfies program p, then it should satisfy condition A and the final state s_h should satisfy condition B. If p terminates, then σ_h is a finite interval; otherwise, it is an infinite interval, denoted by σ_ω $(h = \omega)$. We often refer to $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$ as a *general triple*. However, when $k = 0$, we say $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$ is an *initial triple*, where program p is deduced over the interval σ_h starting from the initial state s_0. For convenience, in the axiomatic system, we abbreviate $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$ to $\{A\}\ p\ \{B\}$ and use (σ_h, k) to denote $\sigma_{(h..k)}$.

Definition 8. Let Σ be a set of all intervals, and $(\sigma_h, k) = \langle s_k, \ldots, s_h \rangle$ be a subinterval of σ_h starting from state s_k $(0 \leqslant k \leqslant h)$, where $\sigma_h = \sigma_k \circ (\sigma_h, k)$. We say that:

— $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$ is *satisfied* over σ_h starting from s_k, denoted

$$(\sigma_h, k) \models \{\sigma_k, A\}\ p\ \{\sigma_h, B\},$$

if given $(\sigma_h, 0, k, h) \models p$, we have $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$.

— $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$ is *valid*, denoted

$$\models \{\sigma_k, A\}\ p\ \{\sigma_h, B\},$$

if for all $\sigma_h \in \Sigma$, we have $(\sigma_h, k) \models \{\sigma_k, A\}\ p\ \{\sigma_h, B\}$.

— $\{A\}\ p\ \{B\}$ is *satisfiable*, denoted

$$(\sigma_h, 0) \models \{A\}\ p\ \{B\},$$

if $(\sigma_h, 0) \models \{\sigma_0, A\}\ p\ \{\sigma_h, B\}$.

— $\{A\}\ p\ \{B\}$ is *valid*, denoted

$$\models \{A\}\ p\ \{B\},$$

if for all $\sigma_h \in \Sigma$, we have $(\sigma_h, 0) \models \{A\}\ p\ \{B\}$.

— $\{\sigma_k, A\}\ p\ \{\sigma_h, B\}$ implies $\{\sigma_k, A\}\ q\ \{\sigma_h, B\}$ if

$$(\sigma_h, k) \models \{\sigma_k, A\}\ p\ \{\sigma_h, B\} \Longrightarrow (\sigma_h, k) \models \{\sigma_k, A\}\ q\ \{\sigma_h, B\}.$$

From Definition 8, we say that $\{A\}\ p\ \{B\}$ is satisfiable if there exists an interval σ_h satisfying program p, such that it satisfies property A and the final state over σ_h satisfies property B. Furthermore, we say that $\{A\}\ p\ \{B\}$ is valid if all the intervals σ_h satisfying program p satisfy property A and the final state satisfies property B. We require that program p be satisfiable, that is, at least a minimal model can satisfy program p.

In Table 4, axioms (AEM) and (APC) are concerned with the *terminal* and assignment statements, respectively (that is, present components). From the validity of correctness

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

(AEM)	$\{\sigma_k, \varepsilon\}\ \varepsilon\ \{\sigma_k, \varepsilon\}$
(APC)	$\{\sigma_k, A\}\ p\ \{\sigma_k, A\}$ if $p \to A$
	where A and p are present components.

Table 4. *Axioms over intervals*

(ISR)	$\{\sigma_k, A\} prog[p] \{\sigma_h, B\}$ and $p \cong q$
	$\Longleftrightarrow \{\sigma_k, A\} prog[q/p] \{\sigma_h, B\}$ and $p \cong q$

Table 5. *Interval substitution rule*

assertion in Definition 8, it is easy to see that both of them are valid. The detailed proof is given in Section 4.

In Table 5, rule (ISR) tells us that in the triple $\{\sigma_k, A\}\ prog[p]\ \{\sigma_h, B\}$, if $p \cong q$ can be deduced by state axioms and state inference rules, then we can replace sub-program p by q in program $prog$.

For convenience, we will now define *state components*, $ps(x)$, by

$$ps(x) ::= x = e \mid x \Leftarrow e \mid b.$$

So $ps(x)$ consists of the assignment statement $x = e$, the positive immediate assignment $x \Leftarrow e$ and boolean expressions b. A *state program* is a conjunction of state components such as $ps(x_1) \wedge ps(x_2) \wedge \ldots \wedge ps(x_n)$. In fact, state programs correspond to boolean expressions and *present components* q_{e_i} and q_{c_j} in the normal form. To formalise the inference rules for state programs, we need some more notation.

Definition 9.

— $s_k[(m, \dot{p_x})/x](y) \stackrel{\text{def}}{=} \begin{cases} (m, \dot{p_x})\ y = x \\ s_k(y)\ \ y \neq x. \end{cases}$

— $\sigma_k(x) \stackrel{\text{def}}{=} \Pi_1(s_k(x))$,
where Π_i is defined in the usual way to obtain the components of the pair.

— $s_i[w] \stackrel{\text{def}}{=} s_i[(m_1, \dot{p_{x_1}})/x_1][(m_2, \dot{p_{x_2}})/x_2] \ldots [(m_n, \dot{p_{x_n}})/x_n]$,
where w is

$$(x_1 = m_1 \wedge \dot{p_{x_1}}) \wedge (x_2 = m_2 \wedge \dot{p_{x_2}}) \ldots \wedge (x_n = m_n \wedge \dot{p_{x_n}}).$$

— $\sigma_k[w] \stackrel{\text{def}}{=} \langle s_0, \ldots, s_k[w] \rangle$,
where w is a present component.

— $\sigma_{k+1} \stackrel{\text{def}}{=} \sigma_k \cdot \langle s_{k+1} \rangle$.

We write $e[x \mapsto e_0]$ for the arithmetic (or boolean) expression obtained by replacing each occurrence of a variable x in an arithmetic expression e (or boolean expression b) with another expression e_0 (or b_0). This is called *substitution*. We define the following notation for substitution.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

(AAS)	$\{\sigma_k, A\}\ x = m \wedge ps(y) \wedge p\ \{\sigma_h, B\}$
$\Longleftrightarrow$	$\{\sigma_k, A\}\ x = m \wedge ps(y)[x \mapsto m] \wedge p\ \{\sigma_h, B\}$
	where $\mathsf{lbf}(y)$ and $\mathsf{lbf}(x)$ does not occur in p.
(LBF)	$\{\sigma_k, A\}\ \mathsf{lbf}(x) \wedge p\ \{\sigma_h, B\}$
$\Longleftrightarrow$	$\{\sigma_k, A\}\ x = \sigma_{k-1}(x) \wedge p\ \{\sigma_h, B\}\ (k \geqslant 1)$
	where $x = e$ and $x \Leftarrow e$ does not occur in p.

Table 6. *Rules for assignment statements*

Definition 10.

$$ps(y)[x \mapsto m] \stackrel{\text{def}}{=} \begin{cases} b(x)[x \mapsto m] & \text{if } ps(y) \text{ is } b(x) \\ y = e(x)[x \mapsto m] & \text{if } ps(y) \text{ is } y = e(x) \\ y = e(x)[x \mapsto m] \wedge p_y & \text{if } ps(y) \text{ is } y \Leftarrow e(x) \\ ps(y) & \text{if } x \text{ does not occur in } ps(y). \end{cases}$$

In Table 6, rules (AAS) and (LBF) are concerned with evaluating the state component $ps(x)$ and the state frame $\mathsf{lbf}(x)$. In rule (AAS), we apply *substitution* to state components to evaluate arithmetic and boolean expressions. Note that we do not allow $\mathsf{lbf}(x)$ (or $\mathsf{lbf}(y)$) to occur in p because $\mathsf{lbf}(x) \wedge x = m \cong x \Leftarrow m$ by axiom (A1). In rule (LBF), the condition '$x = e$ and $x \Leftarrow e$ does not occur in p', ensures that the value of variable x at state s_k is equivalent to the previous value of x, that is, the value of x at state s_{k-1}, denoted by $\sigma_{k-1}(x)$.

In practice, we often obtain the normal form of a program p after several deduction steps using the state axioms and state inference rules in Tables 2 and 3. We assume that the normal form of program p is $p \cong p_c \wedge \bigcirc p_f$ or $p \cong p_e \wedge \varepsilon$. By rule (ISR) in Table 5, we have triples like $\{\sigma_k, A\}\ p_c \wedge \bigcirc p_f\ \{\sigma_h, B\}$ or $\{\sigma_k, A\}\ p_e \wedge \varepsilon\ \{\sigma_h, B\}$. Rules (ANext) and (AEmpty) in Table 7 are formalised using these triples. These rules enable us to deduce a program p from the current state s_k to another state s_{k+1}. More precisely, the state program p_c (or p_e) is deduced at state s_k, and the '*next*' program p_f is to be deduced over an interval starting from state s_{k+1}, while the '*empty*' program ε denotes the fact that the program terminates at current state s_k.

In Table 7, the normal form of A in the model theory of PPTL is denoted by $A \equiv (A_e \wedge \varepsilon) \vee (A_c \wedge \bigcirc A_f)^{\dagger}$.

For rule (ANext):

— If $p_c \rightarrow A_c$, by axiom (APC), we have $\vdash \{\sigma_k, A_c\}\ p_c\ \{\sigma_k, A_c\}$. This means that the state program p_c satisfies the property A_c, which is a present component decomposed by A at state s_k. We then further transform the '*next*' program p_f to state s_{k+1}, and replace the assertion A by its *next* formula A_f. Thus, we can obtain the triple $\{\sigma_{k+1}, A_f\}\ p_f\ \{\sigma_h, B\}$ to continue to deduce program p_f over the remaining interval $\sigma_{(k+1\ldots h)} = \langle s_{k+1}, \ldots, s_h \rangle$.

[†] We have already proved in Duan *et al.* (2008a) that in the model theory of PPTL, any PPTL formulas can be reduced to normal form.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

(ANext)	$\{\sigma_k, A\}\ p_c \wedge \bigcirc p_f\ \{\sigma_h, B\}$
	$\Longleftrightarrow \begin{cases} \{\sigma_k[p_c] \cdot \langle s_{k+1} \rangle, A_f\}\ p_f\ \{\sigma_h, B\} \\ \text{and } \{\sigma_k, A_c\}\ p_c\ \{\sigma_k, A_c\} & \text{if } p_c \to A_c \\ \{\sigma_k, \mathsf{false}\}\ p_c \wedge \bigcirc p_f\ \{\sigma_h, B\} & \text{otherwise} \end{cases}$
	where $A \equiv (A_c \wedge \bigcirc A_f) \vee (A_e \wedge \varepsilon)$
(AEmpty)	$\{\sigma_k, A\}\ p_e \wedge \varepsilon\ \{\sigma_h, B\}$
	$\Longleftrightarrow \begin{cases} \{\sigma_k[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_k[p_e], \varepsilon\} \\ \text{and } \{\sigma_k, A_e \wedge B\}\ p_e\{\sigma_k, A_e \wedge B\} & \text{if } p_e \to A_e \wedge B \\ \{\sigma_k, \mathsf{false}\}\ p_e \wedge \varepsilon\ \{\sigma_k, \mathsf{false}\} & \text{otherwise} \end{cases}$
	where $A \equiv (A_c \wedge \bigcirc A_f) \vee (A_e \wedge \varepsilon)$

Table 7. *Temporal 'Next' and 'Empty' Rules, where p_c, p_e, A_c and A_e are present components*

(SSR)	$\{\sigma_k, A\}\ q_i\ \{\sigma_h, B\}$ $\Longrightarrow \{\sigma_k, A\}\ q_1 \vee q_2\ \{\sigma_h, B\} (i = 1, 2)$
(EQR1)	$\{\sigma_k, A\}\ p_c(y) \wedge \bigcirc p\ \{\sigma_h, B\}$ $\Longrightarrow \{\sigma_k, A\}\ \exists x : p_c(x) \wedge \bigcirc p\ \{\sigma_h, B\}$
(EQR2)	$\{\sigma_k, A\}\ p_e(y) \wedge \varepsilon\ \{\sigma_h, B\}$ $\Longrightarrow \{\sigma_k, A\}\ \exists x : p_e(x) \wedge \varepsilon\ \{\sigma_h, B\}$

Table 8. *Selection statement and existential quantification rules*

If a state program at each state s_k $(0 \leqslant k \leqslant h)$ satisfies a present component property, we say that the program satisfies property A over the whole interval over which the program is deduced.

— If $p_c \to A_c$ is not true, we say that the program does not satisfy the given property, which is denoted by false, and the program stops with an error.

Rule (AEmpty) is concerned with the *terminal* statement ε, which means that program p terminates at state s_k. Note that we have assumed that $p \equiv p_c \wedge \bigcirc p_f$ is satisfiable. Therefore, when $p_c \to A_c$ is wrong, we can affirm that p cannot satisfy A. However, this is not the case for an unsatisfiable program since program p always satisfies A for $p \equiv \mathsf{false}$.

Table 8 lists rule (SSR) for the selection statement $q_1 \vee q_2$ and rules (EQR1) and (EQR2) for the existential quantification statement $\exists x : p(x)$. In (SSR), if the correctness assertion $\{\sigma_k, A\}\ q_i\ \{\sigma_h, B\}$ holds for q_i $(i = 1\ or\ 2)$, then it is obvious that $\{\sigma_k, A\}\ q_1 \vee q_2\ \{\sigma_h, B\}$ holds. Rules (EQR1) and (EQR2) enable us to eliminate quantifiers in programs, where $p(y)$ is a renamed formula, and $p_e(y)$ and $p_c(y)$ are present components (that is, state programs).

We have now formalised the whole axiom system of the MSVL language, including the state axioms, the state inference rules, the interval axioms and the interval inference rules.

4. Soundness and completeness

In this section we will prove the soundness and relative completeness of our axiom system with regard to terminating programs. Cook introduced the notion of relative completeness

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

for partial correctness formulas in Hoare logic (Cook 1978). In our logic, the arithmetic expressions are involved, so by the Gödel Incompleteness Theorem, no axiomatic system for elementary arithmetic can be complete. Therefore, our system inherits incompleteness. However, by separating the incompleteness of arithmetic from any incompleteness due to inadequacies in the axioms and rules for the programming language constructs, we can obtain relative completeness in the sense of Cook.

In order to prove the soundness and relative completeness, two useful lemmas are required.

Lemma 8. Let $p \equiv p_c \wedge \bigcirc p_f$ be a satisfiable program, and $A \equiv A_c \wedge \bigcirc A_f$ be a PPTL formula. We have $(p \rightarrow A) \equiv (p_c \rightarrow A_c) \wedge (\bigcirc p_f \rightarrow \bigcirc A_f)$.

Proof. Suppose program p is satisfied by an interval σ and $0 \leqslant k \leqslant |\sigma|$. First, we prove $(p \rightarrow A) \supset (p_c \rightarrow A_c) \wedge (\bigcirc p_f \rightarrow \bigcirc A_f)$.

$$
\begin{aligned}
& rclr(\sigma, 0, k, |\sigma|) \models p \rightarrow A \\
& \qquad \Longrightarrow (\sigma, 0, k, |\sigma|) \models A \qquad \text{(by Theorem 1, since } (\sigma, 0, k, |\sigma|) \models p) \\
& \qquad \Longleftrightarrow (\sigma, 0, k, |\sigma|) \models A_c \wedge \bigcirc A_f \\
& \qquad \Longrightarrow (\sigma, 0, k, |\sigma|) \models (\neg p_c \vee A_c) \wedge (\neg \bigcirc p_f \vee \bigcirc A_f) \\
& \qquad \Longleftrightarrow (\sigma, 0, k, |\sigma|) \models (p_c \rightarrow A_c) \wedge (\bigcirc p_f \rightarrow \bigcirc A_f).
\end{aligned}
$$

On the other hand, we prove $(p_c \rightarrow A_c) \wedge (\bigcirc p_f \rightarrow \bigcirc A_f) \supset (p \rightarrow A)$ as follows.

$$
\begin{aligned}
& (\sigma, 0, k, |\sigma|) \models (p_c \rightarrow A_c) \wedge (\bigcirc p_f \rightarrow \bigcirc A_f) \\
& \Longleftrightarrow (\sigma, 0, k, |\sigma|) \models (\neg p_c \vee A_c) \wedge (\neg \bigcirc p_f \vee \bigcirc A_f) \\
& \Longleftrightarrow (\sigma, 0, k, |\sigma|) \models (\neg p_c \wedge \neg \bigcirc p_f) \vee (\neg p_c \wedge \bigcirc A_f) \vee (A_c \wedge \neg \bigcirc p_f) \vee (A_c \wedge \bigcirc A_f) \\
& \Longrightarrow (\sigma, 0, k, |\sigma|) \models (\neg p_c \vee \neg \bigcirc p_f) \vee (A_c \wedge \bigcirc A_f) \\
& \Longleftrightarrow (\sigma, 0, k, |\sigma|) \models \neg (p_c \wedge \bigcirc p_f) \vee (A_c \wedge \bigcirc A_f) \\
& \Longleftrightarrow (\sigma, 0, k, |\sigma|) \models \neg p \vee A \\
& \Longleftrightarrow (\sigma, 0, k, |\sigma|) \models p \rightarrow A.
\end{aligned}
$$

Hence, $(p \rightarrow A) \equiv (p_c \rightarrow A_c) \wedge (\bigcirc p_f \rightarrow \bigcirc A_f)$. □

Lemma 9. The axioms and inference rules over intervals in Tables (4)–(8) are sound.

Proof. The proof proceeds by induction on the axioms and inference rules over intervals, based on the validity of the correctness assertion defined in Definition 8:

Axiom (AEM): We need to prove $\models \{\sigma_k, \varepsilon\}\ \varepsilon\ \{\sigma_k, \varepsilon\}$, where $h = k$. By Definition 8, we have $\models \{\sigma_k, \varepsilon\}\ \varepsilon\ \{\sigma_k, \varepsilon\} \Longleftrightarrow \forall \sigma_k \in \Sigma$ if $(\sigma_k, 0, k, k) \models \varepsilon$, then $(\sigma_k, 0, k, k) \models \varepsilon \Longleftrightarrow \varepsilon \rightarrow \varepsilon \Longleftrightarrow \mathsf{true}$.

Axiom (APC): We need to prove $\models \{\sigma_k, A\}\ p\ \{\sigma_k, A\}$, where $h = k$. Under the condition of $p \rightarrow A$, by Definition 8, $\models \{\sigma_k, A\}\ p\ \{\sigma_k, A\} \Longleftrightarrow \forall \sigma_k \in \Sigma$ if $(\sigma_k, 0, k, k) \models p$, then $(\sigma_k, 0, k, k) \models A \Longleftrightarrow p \rightarrow A \Longleftrightarrow \mathsf{true}$.

Rule (ISR): We need to prove

$$\models \{\sigma_k, A\}\ prog[p]\ \{\sigma_h, B\} \text{ and } p \equiv q \Longleftrightarrow \models \{\sigma_k, A\}\ prog[q/p]\ \{\sigma_h, B\} \text{ and } p \equiv q.$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

By Definition 8, $\models \{\sigma_k, A\}\ prog[p]\ \{\sigma_h, B\}$ and $p \equiv q \iff \forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models prog[p]$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$ and $p \equiv q$. By a congruence rule in model theory, we can replace some occurrences of p in $prog$ by q if $p \equiv q$. Thus, $\forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models prog[q/p]$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$ and $p \equiv q$. Hence $\models \{\sigma_k, A\}\ prog[q/p]\ \{\sigma_h, B\}$ and $p \equiv q$.

Rule (AAS): We need to prove

$$\models \{\sigma_k, A\}\ x = m \wedge ps(y) \wedge p\ \{\sigma_h, B\} \iff \models \{\sigma_k, A\}\ x = m \wedge ps(y)[m/x] \wedge p\ \{\sigma_h, B\}.$$

By Definition 8, $\models \{\sigma_k, A\}\ x = m \wedge ps(y) \wedge p\ \{\sigma_h, B\} \iff \forall \sigma_h \in \Sigma$, if $(\sigma_h, 0, k, h) \models x = m \wedge ps(y) \wedge p$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$. By the substitution rule for expressions, we have $\models \{\sigma_k, A\}\ x = m \wedge ps(y)[m/x] \wedge p\ \{\sigma_h, B\}$.

Rule (PRE): We need to prove

$$\models \{\sigma_k, A\}\ \mathsf{lbf}(x) \wedge p\ \{\sigma_h, B\} \iff \models \{\sigma_k, A\}\ x = \sigma_{k-1}(x) \wedge p\ \{\sigma_h, B\}.$$

By Definition 8, $\models \{\sigma_k, A\}\ \mathsf{lbf}(x) \wedge p\ \{\sigma_h, B\} \iff \forall \sigma_h \in \Sigma$, if $(\sigma_h, 0, k, h) \models \mathsf{lbf}(x) \wedge p$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$. By the minimal model semantics of MSVL, we have $\forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models x = \ominus x \wedge p$, then $(\sigma_h, 0, k, h) \models A$and $(\sigma_h, 0, h, h) \models B$. By Definition 8, we have $\{\sigma_k, A\}\ x = \ominus x \wedge p\ \{\sigma_h, B\}$. Note that $\ominus x$ can be captured by $\sigma_{k-1}(x)$ defined in Definition 10. Hence, $\models \{\sigma_k, A\}\ x = \sigma_{k-1}(x) \wedge p\ \{\sigma_h, B\}$.

Rule (ANext): We need to prove

$$\models \{\sigma_k, A\}\ p_c \wedge \bigcirc p_f\ \{\sigma_h, B\} \iff \models \{\sigma_k[p_c] \cdot \langle s_{k+1}\rangle, A_f\}\ p_f\ \{\sigma_h, B\} \text{ and } \models \{\sigma_k, A_c\}\ p_c\ \{\sigma_k, A_c\},$$

where

$$A \equiv (A_e \wedge \varepsilon) \vee (A_c \wedge \bigcirc A_f) \text{ and } \sigma_{k+1} = \sigma_k[p_c] \cdot \langle s_{k+1}\rangle.$$

By Definition 8, we have

$\models \{\sigma_k[p_c] \cdot \langle s_{k+1}\rangle, A_f\}\ p_f\ \{\sigma_h, B\}$ and $\models \{\sigma_k, A_c\}\ p_c\ \{\sigma_k, A_c\}$

$\iff$ for all $\sigma_h \in \Sigma$ if $(\sigma_h, 0, k+1, h) \models p_f$,
then $(\sigma_h, 0, k+1, h) \models A_f$ and $(\sigma_h, 0, h, h) \models B$, and
for all $\sigma_h \in \Sigma$ if $(\sigma_h, 0, k, k) \models p_c$, then $(\sigma_h, 0, k, k) \models A_c$

$\iff$ for all $\sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models \bigcirc p_f$, then $(\sigma_h, 0, k, h) \models \bigcirc(A_f \wedge \mathsf{fin}(B))$
and if $(\sigma_h, 0, k, k) \models p_c$ then $(\sigma_h, 0, k, k) \models A_c$

$\iff (\sigma_h, 0, k, h) \models (\bigcirc p_f \rightarrow \bigcirc(A_f \wedge \mathsf{fin}(B)) \wedge (p_c \rightarrow A_c)$

$\iff (\sigma_h, 0, k, h) \models (p_c \wedge \bigcirc p_f) \rightarrow (A_c \wedge \bigcirc(A_f \wedge \mathsf{fin}(B))$
(by Lemma 8 since $p_c \wedge \bigcirc p_f$ is satisfiable)

$\iff$ for all $\sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models p_c \wedge \bigcirc p_f$, then $(\sigma_h, 0, k, h) \models A_c \wedge \bigcirc A_f$
and $(\sigma_h, 0, h, h) \models B$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

$$\Longleftrightarrow \text{for all } \sigma_h \in \Sigma \text{ if } (\sigma_h, 0, k, h) \models p_c \wedge \bigcirc p_f, \text{ then } (\sigma_h, 0, k, h) \models A \text{ and } (\sigma_h, 0, h, h) \models B$$

$$\Longleftrightarrow \models \{\sigma_k, A\}\ p_c \wedge \bigcirc p_f\ \{\sigma_h, B\}.$$

Rule (AEmpty): The proof of rule (AEmpty) is similar to the proof of rule (ANext).

Rule (SSR): We need to prove

$$\models \{\sigma_k, A\}\ q_i\ \{\sigma_h, B\} \Longrightarrow\ \models \{\sigma_k, A\}\ q_1 \vee q_2\ \{\sigma_h, B\} (i = 1, 2)$$

By Definition 8,
$\models \{\sigma_k, A\}\ q_1\ \{\sigma_h, B\} \Longleftrightarrow \forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models q_1$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$,
or
$\models \{\sigma_k, A\}\ q_2\ \{\sigma_h, B\} \Longleftrightarrow \forall \sigma_h \in \Sigma$, if $(\sigma_h, 0, k, h) \models q_2$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$.
Hence, we can infer that $\forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models q_1 \vee q_2$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$. By Definition 8, $\models \{\sigma_k, A\}\ q_1 \vee q_2\ \{\sigma_h, B\}$.

Rule (EQR1): We need to prove

$$\models \{\sigma_k, A\}\ p_c(y) \wedge \bigcirc p\ \{\sigma_h, B\} \Longrightarrow\ \models \{\sigma_k, A\}\ \exists x : p_c(x) \wedge \bigcirc p\ \{\sigma_h, B\}$$

By Definition 8, $\models \{\sigma_k, A\}\ p_c(y) \wedge \bigcirc p\ \{\sigma_h, B\} \Longleftrightarrow \forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models p_c(y) \wedge \bigcirc p$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B \Longleftrightarrow \forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, k) \models p_c(y)$ and $(\sigma_h, 0, k+1, h) \models p$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$. Because $p_c(y)$ is a renamed formula of $\exists x : p_c(x)$, by Definition 2, $p_c(y) \overset{y}{\equiv} \exists x : p(x)$. Therefore, $\forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, k) \models \exists x : p(x)$ and $(\sigma_h, 0, k+1, h) \models p$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B \Longleftrightarrow \forall \sigma_h \in \Sigma$ if $(\sigma_h, 0, k, h) \models \exists x : p(x) \wedge \bigcirc p$, then $(\sigma_h, 0, k, h) \models A$ and $(\sigma_h, 0, h, h) \models B$. Therefore, $\models \{\sigma_k, A\}\ \exists x : p_c(x) \wedge \bigcirc p\ \{\sigma_h, B\}$.

Rule (EQR2): The proof of rule (EQR2) is similar to the proof of rule (EQR1). □

4.1. *Deduction tree and outline of the proofs*

From the deduction of $\{A\}\ p\ \{B\}$, we can obtain the *deduction tree* shown in Figure 8, where d denotes the depth of the deduction tree. We will first introduce the deduction tree and give an outline of the proofs of soundness and relative completeness for $\{A\}\ p\ \{B\}$. We will then prove soundness and relative completeness in detail.

For any given MSVL program p, we can use the state axioms and state inference rules in Tables 2 and 3 to reduce p in its initial state s_0 to its normal form as $p \cong p_{c_1} \wedge \bigcirc p_{f_1}$ (if p does not terminate at s_0). Suppose $A \equiv (A_{c_1} \wedge \bigcirc A_{f_1}) \vee (A_e \wedge \varepsilon)$. Then, in the initial triple $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$, by rule (ISR), we have $\{\sigma_0, A\}\ p_{c_1} \wedge \bigcirc p_{f_1}\ \{\sigma_h, B\}$. Hence the initial triple can be decomposed using rule (ANext) into two triples of the form $\{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\}$ and $\{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\}$, where the former holds by axiom (APC) and the latter will be deduced at the next state s_1 over the subinterval $\langle s_1, \ldots, s_h \rangle$ of σ_h. Continuing in this way, we let p_{f_k} $(1 \leqslant k \leqslant h)$ be a successor of p, and $A_{f_k} \equiv (A_{c_{k+1}} \wedge \bigcirc A_{f_{k+1}}) \vee (A_e \wedge \varepsilon)$. At each

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

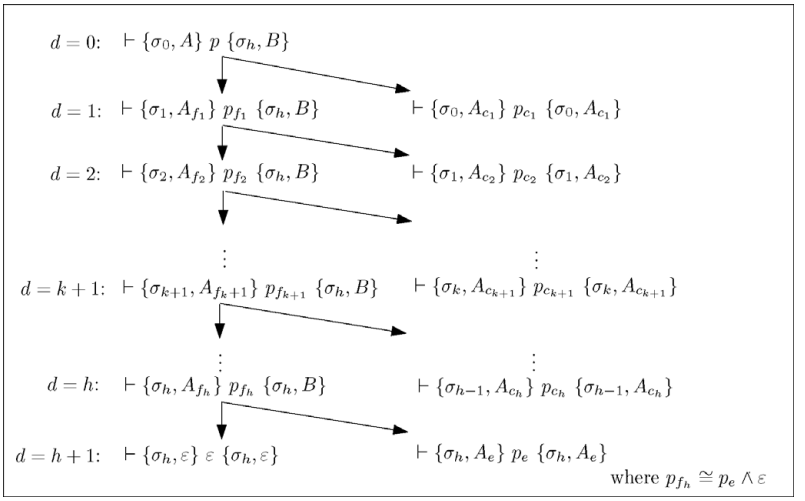


Fig. 8. Deduction tree for program p

state s_k, we decompose

$$\{\sigma_k, A_{f_k}\}\ p_{f_k}\ \{\sigma_h, B\}\ (k < h, p_{f_k} \cong p_{c_{k+1}} \wedge \bigcirc p_{f_{k+1}})$$

into

$$\{\sigma_k, A_{c_{k+1}}\}\ p_{c_{k+1}}\ \{\sigma_k, A_{c_{k+1}}\}$$

and

$$\{\sigma_{k+1}, A_{f_{k+1}}\}\ p_{f_{k+1}}\ \{\sigma_h, B\},$$

until

$$\{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\},$$

where $p_{f_h} = p_e \wedge \varepsilon$. By rule (AEmpty), $\{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\}$ can then be deduced to

$$\{\sigma_h, \varepsilon\}\ \varepsilon\ \{\sigma_h, \varepsilon\}$$

and

$$\{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\}.$$

By axioms (APC) and (AEM), both of these hold, so the initial triple holds. That is, program p satisfies properties A and B. Otherwise, if we cannot deduce an initial triple into axioms, p cannot satisfy A and B. Armed with the deduction tree, soundness and relative completeness can then be proved by considering two cases:

(i) For $h = 0$, we have $\{\sigma_0, A\}\ p\ \{\sigma_0, B\}$, and the normal form of p is $p \equiv p_e \wedge \varepsilon$.

(ii) For $h > 0$, we have $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$, and the normal form of p is $p \equiv p_c \wedge \bigcirc p_f$. From Figure 8, we can see that to prove $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$, we need to prove that every component $\{\sigma_{k-1}, A_{c_k}\}\ p_{c_k}\ \{\sigma_{k-1}, A_{c_k}\}$ and $\{\sigma_h, A_e\}\ p_e\ \{\sigma_h, A_e\}$ is true for $1 \leqslant k \leqslant h+1$. Therefore, the proof proceeds by induction on the depth of the deduction tree d.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Theorem 10. The proof system Π_{MSVL} for the correctness assertion is sound, that is, for any correctness assertion $\{A\}\ p\ \{B\}$, where p is terminating,

$$\vdash \{A\}\ p\ \{B\} \Longrightarrow \models \{A\}\ p\ \{B\}$$

Proof. We consider the following two cases:

(i) $h = 0$, that is, $\{A, \sigma_0\}\ p\ \{B, \sigma_0\}$.
In this case, $p \cong p_e \wedge \varepsilon$. By rules (ISR), (AAS), (LBF) and (AEmpty), we have

$$\begin{aligned}
&\vdash \{\sigma_0, A\}\ p\ \{\sigma_0, B\} \text{ and } p \cong p_e \wedge \varepsilon \\
&\quad \Longleftrightarrow \vdash \{\sigma_0, A\}\ p_e \wedge \varepsilon\ \{\sigma_0, B\} && \text{(ISR, AAS, LBF)} \\
&\qquad \text{If } p_e \rightarrow A_e \wedge B, \text{ then} \\
&\quad \Longleftrightarrow \vdash \{\sigma_0[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_0[p_e], \varepsilon\} \text{ and} \\
&\qquad \vdash \{\sigma_0, A_e \wedge B\}\ p_e\ \{\sigma_0, A_e \wedge B\} && \text{(AEmpty)}
\end{aligned}$$

where rules (AAS) and (LBF) are used implicitly to evaluate expressions in p_e. By Theorem 6, $p \cong p_e \wedge \varepsilon \Longrightarrow p \equiv p_e \wedge \varepsilon$. By Lemma 9, rules (ISR), (AAS) and (LBF) are sound. Hence, $\models \{\sigma_0, A\}\ p\ \{\sigma_0, B\}$ and $p \equiv p_e \wedge \varepsilon \Longleftrightarrow \models \{\sigma_0, A\}\ p_e \wedge \varepsilon\ \{\sigma_0, B\}$. Furthermore, by Lemma 9, rule (AEmpty) is sound. Thus, if $p_e \rightarrow A_e \wedge B$, we have

$$\begin{aligned}
\models \{\sigma_0, A\}\ p_e \wedge \varepsilon\ \{\sigma_0, B\} \Longleftrightarrow &\models \{\sigma_0[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_0[p_e], \varepsilon\} \text{ and} \\
&\models \{\sigma_0, A_e \wedge B\}\ p_e\ \{\sigma_0, A_e \wedge B\}
\end{aligned}$$

By Lemma 9, axioms (AEM) and (APC) are sound, so we have

$$\vdash \{\sigma_0[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_0[p_e], \varepsilon\} \Longrightarrow \models \{\sigma_0[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_0[p_e], \varepsilon\},$$

and under the condition of $p_e \rightarrow A_e \wedge B$,

$$\vdash \{\sigma_0, A_e \wedge B\}\ p_e\ \{\sigma_0, A_e \wedge B\} \Longrightarrow \models \{\sigma_0, A_e \wedge B\}\ p_e\ \{\sigma_0, A_e \wedge B\}.$$

Hence,

$$\vdash \{\sigma_0, A\}\ p\ \{\sigma_0, B\} \Longrightarrow \models \{\sigma_0, A\}\ p\ \{\sigma_0, B\}.$$

By Definition 8, we can conclude that

$$\models \{A\}\ p\ \{B\} \Longrightarrow \vdash \{A\}\ p\ \{B\}$$

for $h = 0$.

(ii) $h > 0$, that is, $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$.
In this case, the normal form of p is $p \cong p_c \wedge \bigcirc p_f$. From the deduction of $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$, we can obtain a deduction tree, and the proof then proceeds by induction on the depth of deduction tree d in Figure 8:

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Base case: For $d = 1$, by rules (ISR), (AAS), (LBF) and (ANext), we have

$$\begin{aligned}
&\vdash \{\sigma_0, A\}\ p\ \{\sigma_h, B\} \text{ and } p \cong p_{c_1} \wedge \bigcirc p_{f_1} \\
&\iff \vdash \{\sigma_0, A\}\ p_{c_1} \wedge \bigcirc p_{f_1}\ \{\sigma_h, B\} && \text{(ISR, AAS, LBF)} \\
&\quad \text{If } p_{c_1} \to A_{c_1}, \text{ then} \\
&\iff \vdash \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\} \text{ and} \\
&\quad \vdash \{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\} && \text{(ANext, } \sigma_1 = \sigma_0[p_{c_1}] \cdot \langle s_1 \rangle)
\end{aligned}$$

where rules (AAS) and (LBF) are used implicitly to evaluate expressions in p_{c_1}. By Theorem 6, we have $p \cong p_{c_1} \wedge \bigcirc p_{f_1} \Longrightarrow p \equiv p_{c_1} \wedge \bigcirc p_{f_1}$. By Lemma 9, rules (ISR), (AAS) and (LBF) are sound. Hence, $\models \{\sigma_0, A\}\ p\ \{\sigma_h, B\}$ and

$$p \equiv p_{c_1} \wedge \bigcirc p_{f_1} \iff \models \{\sigma_0, A\}\ p_{c_1} \wedge \bigcirc p_{f_1}\ \{\sigma_h, B\}.$$

Furthermore, by Lemma 9, rule (ANext) is sound. Thus, if $p_{c_1} \to A_{c_1}$, we have

$$\begin{aligned}
\models \{\sigma_0, A\}\ p_{c_1} \wedge \bigcirc p_{f_1}\ \{\sigma_h, B\} \Longleftrightarrow &\models \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\} \text{ and} \\
&\models \{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\}.
\end{aligned}$$

By Lemma 9, axiom (APC) is sound. So we can conclude that

$$\vdash \{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\} \Longrightarrow \models \{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\}.$$

Thus, to prove

$$\vdash \{\sigma_0, A\}\ p\ \{\sigma_h, B\} \Longrightarrow \models \{\sigma_0, A\}\ p\ \{\sigma_h, B\},$$

we need to prove

$$\vdash \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\} \Longrightarrow \models \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\}.$$

Induction step: By Figure 8, for any depth $d = k, 1 < k \leqslant h$, we assume

$$\vdash \{\sigma_{k-1}, A_{c_k}\}\ p_{c_k}\ \{\sigma_{k-1}, A_{c_k}\} \Longrightarrow \models \{\sigma_{k-1}, A_{c_k}\}\ p_{c_k}\ \{\sigma_{k-1}, A_{c_k}\}$$

and then prove for $d = h + 1$, that

$$\vdash \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\} \Longrightarrow \models \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\}.$$

From Figure 8, $p_{f_h} \cong p_e \wedge \varepsilon$. By rule (AEmpty), we assume

$$A_{f_h} \equiv (A'_c \wedge \bigcirc A'_f) \vee (A_e \wedge \varepsilon).$$

Then,

$$\begin{aligned}
&\vdash \{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\} \text{ and } p_{f_h} \cong p_e \wedge \varepsilon \\
&\quad \Longleftrightarrow \vdash \{\sigma_h, A_{f_h}\}\ p_e \wedge \varepsilon\ \{\sigma_h, B\} && \text{(ISR, AAS, LBF)} \\
&\qquad \text{If } p_e \to A_e \wedge B, \text{ then} \\
&\quad \Longleftrightarrow \vdash \{\sigma_h[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_h[p_e], \varepsilon\} \text{ and} \\
&\qquad \vdash \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\} && \text{(AEmpty)}
\end{aligned}$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

By Theorem 6, $p_{f_h} \cong p_e \wedge \varepsilon \Longrightarrow p_{f_h} \equiv p_e \wedge \varepsilon$. By Lemma 9, rules (ISR) and (AEmpty) are sound. Thus, if $p_e \rightarrow A_e \wedge B$, we have

$$\models \{\sigma_h, A_{f_h}\}\ p_e \wedge \varepsilon\ \{\sigma_h, B\} \Longleftrightarrow \models \{\sigma_h[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_h[p_e], \varepsilon\} \text{ and } \models \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\}$$

By Lemma 9, axioms (AEM) and (APC) are sound. Hence, for depth $d = h + 1$, we have,

$$\vdash \{\sigma_h[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_h[p_e], \varepsilon\} \Longrightarrow \models \{\sigma_h[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_h[p_e], \varepsilon\}$$

and

$$\vdash \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\} \Longrightarrow \models \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\}$$

Hence,

$$\vdash \{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\} \Longrightarrow \models \{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\}.$$

Therefore, for $h > 0$,

$$\vdash \{\sigma_0, A\}\ p\ \{\sigma_h, B\} \Longrightarrow \models \{\sigma_0, A\}\ p\ \{\sigma_h, B\},$$

and, by Definition 8, we can conclude that $\vdash \{A\}\ p\ \{B\} \Longrightarrow \models \{A\}\ p\ \{B\}$. □

Theorem 11. The proof system Π_{MSVL} for the correctness assertion is relatively complete, that is, for any correctness assertion $\{A\}\ p\ \{B\}$, where p is terminating,

$$\models \{A\}\ p\ \{B\} \Longrightarrow \vdash \{A\}\ p\ \{B\}.$$

Proof. We will first prove relative completeness for programs without the selection statement $p_1 \vee p_2$ and existential quantification $\exists x : p(x)$. We consider two cases:

(i) $h = 0$, that is, $\{\sigma_0, A\}\ p\ \{\sigma_0, B\}$.

In this case, $p \equiv p_e \wedge \varepsilon$. We assume $A \equiv A_e \wedge \varepsilon$. By Definition 8, we have

$$\begin{aligned}
&\models \{\sigma_0, A\}\ p\ \{\sigma_0, B\} \text{ and } p \equiv p_e \wedge \varepsilon \\
&\quad \Longleftrightarrow \models \{\sigma_0, A\}\ p_e \wedge \varepsilon\ \{\sigma_0, B\} \\
&\quad \Longleftrightarrow \forall \sigma_0 \in \Sigma, \text{ if } (\sigma_0, 0, 0, 0) \models p_e \wedge \varepsilon, \text{ then} \\
&\qquad (\sigma_0, 0, 0, 0) \models A \text{ and } (\sigma_0, 0, 0, 0) \models B \\
&\quad \Longleftrightarrow \forall \sigma_0 \in \Sigma, \text{ if } (\sigma_0, 0, 0, 0) \models p_e \wedge \varepsilon, \text{ then} \\
&\qquad (\sigma_0, 0, 0, 0) \models A_e \wedge \varepsilon \text{ and } (\sigma_0, 0, 0, 0) \models B \\
&\quad \Longleftrightarrow \forall \sigma_0 \in \Sigma, \text{ if } (\sigma_0, 0, 0, 0) \models p_e \text{ and } (\sigma_0, 0, 0, 0) \models \varepsilon, \text{ then} \\
&\qquad (\sigma_0, 0, 0, 0) \models A_e \text{ and } (\sigma_0, 0, 0, 0) \models \varepsilon \text{ and } (\sigma_0, 0, 0, 0) \models B \\
&\quad \Longleftrightarrow p_e \wedge \varepsilon \rightarrow A_e \wedge B \wedge \varepsilon \\
&\quad \Longrightarrow p_e \rightarrow A_e \wedge B \qquad (1)
\end{aligned}$$

By Theorem 7, $p \equiv p_e \wedge \varepsilon \Longrightarrow p \cong p_e \wedge \varepsilon$. Thus, by rules (ISR), (AAS) and (LBF), $\vdash \{\sigma_0, A\}\ p\ \{\sigma_h, B\}$ and $p \cong p_e \wedge \varepsilon \Longleftrightarrow \vdash \{\sigma_0, A\}\ p_e \wedge \varepsilon\ \{\sigma_h, B\}$, where the expressions in p_e can be evaluated by rules (AAS) and (LBF). From (1), we know $p_e \rightarrow A_e \wedge B$. So, by rule (AEmpty),

$$\vdash \{\sigma_0, A\}\ p_e \wedge \varepsilon\ \{\sigma_0, B\} \Longleftrightarrow \vdash \{\sigma_0, \varepsilon\}\ \varepsilon\ \{\sigma_0, \varepsilon\} \text{ and } \vdash \{\sigma_0, A_e \wedge B\}\ p_e\ \{\sigma_0, A_e \wedge B\}$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

By axiom (AEM), $\vdash \{\sigma_0, \varepsilon\}\ \varepsilon\ \{\sigma_0, \varepsilon\}$. Under the condition $p_e \to A_e \wedge B$, by axiom (APC), we have

$$\vdash \{\sigma_0, A_e \wedge B\}\ p_e\ \{\sigma_0, A_e \wedge B\}.$$

Thus,

$$\models \{\sigma_0, A\}\ p\ \{\sigma_0, B\} \Longrightarrow\ \vdash \{\sigma_0, A\}\ p\ \{\sigma_0, B\},$$

and by Definition 8, we can conclude that

$$\models \{A\}\ p\ \{B\} \Longrightarrow\ \vdash \{A\}\ p\ \{B\}$$

for $h = 0$.

(ii) $h > 0$, that is, $\{\sigma_0, A\}\ p\ \{\sigma_h, B\}$.

In this case, $p \equiv p_c \wedge \bigcirc p_f$. The proof proceeds by induction on the depth d of the deduction tree in Figure 8:

Base case: For $d = 1$.

Suppose $p \equiv p_{c_1} \wedge \bigcirc p_{f_1}$, $A \equiv A_{c_1} \wedge \bigcirc A_{f_1}$. Then, by Definition 8,

$$\begin{aligned}
&\models \{\sigma_0, A\}\ p\ \{\sigma_h, B\}\\
&\quad\Longleftrightarrow\quad \models \{\sigma_0, A\}\ p_{c_1} \wedge \bigcirc p_{f_1}\ \{\sigma_h, B\}\\
&\quad\Longleftrightarrow\quad \forall \sigma_h \in \Sigma,\ \text{if } (\sigma_h, 0, 0, h) \models p_{c_1} \wedge \bigcirc p_{f_1},\ \text{then}\\
&\qquad\qquad (\sigma_h, 0, 0, h) \models A \text{ and } (\sigma_h, 0, h, h) \models B\\
&\quad\Longleftrightarrow\quad \forall \sigma_h \in \Sigma,\ \text{if } (\sigma_h, 0, 0, h) \models p_{c_1} \text{ and } (\sigma_h, 0, 0, h) \models \bigcirc p_{f_1},\ \text{then}\\
&\qquad\qquad (\sigma_h, 0, 0, h) \models A_{c_1} \text{ and } (\sigma_h, 0, 0, h) \models \bigcirc(A_{f_1} \wedge fin(B))\\
&\quad\Longleftrightarrow\quad \forall \sigma_h \in \Sigma,\ \text{if } (\sigma_h, 0, 0, h) \models p_{c_1} \text{ then } (\sigma_h, 0, 0, h) \models A_{c_1},\\
&\qquad\qquad \text{and if } (\sigma_h, 0, 0, h) \models \bigcirc p_{f_1},\ \text{then}\\
&\qquad\qquad (\sigma_h, 0, 0, h) \models \bigcirc(A_{f_1} \wedge fin(B)) \qquad\qquad \text{(Lemma 8)}\\
&\quad\Longleftrightarrow\quad \forall \sigma_h \in \Sigma,\ \text{if } (\sigma_h, 0, 0, h) \models p_{c_1} \text{ then}\\
&\qquad\qquad (\sigma_h, 0, 0, h) \models A_{c_1} \text{ and if } (\sigma_h, 0, 1, h) \models p_{f_1},\ \text{then}\\
&\qquad\qquad (\sigma_h, 0, 1, h) \models A_{f_1} \text{ and } (\sigma_h, 0, h, h) \models B\\
&\quad\Longleftrightarrow\quad p_{c_1} \to A_{c_1} \text{ and } \models \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\} \qquad (2)
\end{aligned}$$

By Theorem 7, $p \equiv p_{c_1} \wedge \bigcirc p_{f_1} \Longrightarrow p \cong p_{c_1} \wedge \bigcirc p_{f_1}$. Hence, by rules (ISR), (AAS) and (LBF), we have

$$\vdash \{\sigma_0, A\}\ p\ \{\sigma_h, B\} \text{ and } p \cong p_{c_1} \wedge \bigcirc p_{f_1} \Longleftrightarrow \vdash \{\sigma_0, A\}\ p_{c_1} \wedge \bigcirc p_{f_1}\ \{\sigma_h, B\},$$

where the expressions in p_e are computed by rules (AAS) and (LBF). From (2), we have $p_{c_1} \to A_{c_1}$. Thus, by rule (ANext),

$$\vdash \{\sigma_0, A\}\ p_{c_1} \wedge \bigcirc p_{f_1}\ \{\sigma_h, B\} \Longleftrightarrow \vdash \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\} \text{ and } \vdash \{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\}$$

where $\sigma_1 = \sigma_0[p_{c_1}] \cdot \langle s_1 \rangle$. By axiom (APC), we can conclude that

$$\models \{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\}\ \Longrightarrow\ \vdash \{\sigma_0, A_{c_1}\}\ p_{c_1}\ \{\sigma_0, A_{c_1}\}.$$

Thus, proving

$$\models \{\sigma_0, A\}\ p\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_0, A\}\ p\ \{\sigma_h, B\}$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

is equivalent to proving

$$\models \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_1, A_{f_1}\}\ p_{f_1}\ \{\sigma_h, B\}.$$

Induction step: For any depth $d = k$, $(0 < k \leqslant h)$, we assume that

$$\models \{\sigma_{k-1}, A_{c_k}\}\ p_{c_k}\ \{\sigma_{k-1}, A_{c_k}\} \Longrightarrow \vdash \{\sigma_{k-1}, A_{c_k}\}\ p_{c_k}\ \{\sigma_{k-1}, A_{c_k}\},$$

and prove the case for $d = h + 1$. We assume that $p_{f_h} \equiv p_e \wedge \varepsilon$ and $A_{f_h} \equiv (A'_c \wedge \bigcirc A'_f) \vee (A_e \wedge \varepsilon)$. By Definition 8, we have

$$\begin{aligned}
&\models \{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\}\\
&\Longleftrightarrow\ \models \{\sigma_h, A_{f_h}\}\ p_e \wedge \varepsilon\ \{\sigma_h, B\}\\
&\Longleftrightarrow\ \forall \sigma_h \in \Sigma \text{ if } (\sigma_h, 0, h, h) \models p_e \wedge \varepsilon, \text{ then}\\
&\qquad (\sigma_h, 0, h, h) \models A_{f_h} \text{ and } (\sigma_h, 0, h, h) \models B\\
&\Longleftrightarrow\ \models p_e \wedge \varepsilon \rightarrow A_{f_h} \wedge B\\
&\Longleftrightarrow\ \models p_e \wedge \varepsilon \rightarrow ((A'_c \wedge \bigcirc A'_f) \vee (A_e \wedge \varepsilon)) \wedge B\\
&\Longleftrightarrow\ \models p_e \wedge \varepsilon \rightarrow (A'_c \wedge \bigcirc A'_f \wedge B) \text{ or}\\
&\qquad \models p_e \wedge \varepsilon \rightarrow (A_e \wedge \varepsilon \wedge B)\\
&\Longleftrightarrow\ \models p_e \wedge \varepsilon \rightarrow (A_e \wedge \varepsilon \wedge B)\\
&\Longleftrightarrow\ \models p_e \rightarrow A_e \wedge B \qquad\qquad (\text{since } p_e, A_e, B \text{ are state formulas})
\end{aligned}$$

By Theorem 7, $p_{f_h} \equiv p_e \wedge \varepsilon \Longrightarrow p_{f_h} \cong p_e \wedge \varepsilon$. From the above, $p_e \rightarrow A_e \wedge B$. So, by rule (ANext),

$$\begin{aligned}
\vdash \{\sigma_h, A\}\ p_e \wedge \varepsilon\ \{\sigma_h, B\} \Longleftrightarrow &\vdash \{\sigma_h[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_h[p_e], \varepsilon\} \text{ and}\\
&\vdash \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\}
\end{aligned}$$

Thus, for depth $d = h + 1$, by axioms (AEM) and (APC), respectively,

$$\vdash \{\sigma_h[p_e], \varepsilon\}\ \varepsilon\ \{\sigma_h[p_e], \varepsilon\}$$

and

$$\vdash \{\sigma_h, A_e \wedge B\}\ p_e\ \{\sigma_h, A_e \wedge B\},$$

both hold. Hence,

$$\models \{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_h, A_{f_h}\}\ p_{f_h}\ \{\sigma_h, B\},$$

so

$$\models \{\sigma_0, A\}\ p\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_0, A\}\ p\ \{\sigma_h, B\},$$

and, by Definition 8, we can conclude that

$$\models \{A\}\ p\ \{B\} \Longrightarrow\ \vdash \{A\}\ p\ \{B\}.$$

We now prove the theorem for the constructs $p_1 \vee p_2$ and $\exists x : p(x)$ as follows:

(i) $\models \{\sigma_0, A\}\ p_1 \vee p_2\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_0, A\}\ p_1 \vee p_2\ \{\sigma_h, B\}.$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

By the model theory and Theorem 8, it is easy to see that

$$\models \{\sigma_0, A\}\ p_1 \vee p_2\ \{\sigma_h, B\} \Longrightarrow\ \models \{\sigma_0, A\}\ p_i\ \{\sigma_h, B\}$$

for ($i = 1$ or 2). Suppose

$$\models \{\sigma_0, A\}\ p_i\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_0, A\}\ p_i\ \{\sigma_h, B\}\ (i = 1\ or\ 2).$$

By rule (SSR), we know

$$\vdash \{\sigma_k, A\}\ p_i\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_k, A\}\ p_1 \vee p_2\ \{\sigma_h, B\}.$$

Hence,

$$\models \{\sigma_0, A\}\ p_1 \vee p_2\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_k, A\}\ p_1 \vee p_2\ \{\sigma_h, B\}.$$

(ii) $\models \{\sigma_0, A\}\ \exists x : p(x)\ \{\sigma_h, B\} \Longrightarrow\ \vdash \{\sigma_0, A\}\ \exists x : p(x)\ \{\sigma_h, B\}$.
Suppose $p(x) \cong (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x))$. By rule (R2) in Table 2, we have

$$\begin{aligned} p(x) &\cong (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x)) \\ &\Longrightarrow \exists x : p(x) \cong (\exists x : p_e(x) \wedge \varepsilon) \vee (\exists x : p_c(x) \wedge \bigcirc \exists x : p_f(x)). \end{aligned}$$

Since $\exists x : p_e(x) \wedge \varepsilon$ and $\exists x : p_c(x) \wedge \bigcirc \exists x : p_f(x)$ are exclusive, we will discuss them separately:

— $\exists x : p(x) \cong \exists x : p_e(x) \wedge \varepsilon$:
Let $p_e(y)$ be a renamed formula of $\exists x : p_e(x)$ (see Lemma 3). By Definition 2, we have $p_e(y) \overset{y}{\equiv} \exists x : p_e(x)$ (which is called y-equivalence – see Duan *et al.* (2008)). Here we use $\overset{y}{\Longleftrightarrow}$ to denote y-equivalence between triples. So we have

$$\models \{\sigma_0, A\}\ \exists x : p_e(x) \wedge \varepsilon\ \{\sigma_h, B\}\ \overset{y}{\Longleftrightarrow} \models \{\sigma_0, A\}\ p_e(y) \wedge \varepsilon\ \{\sigma_h, B\}.$$

Therefore,

$$\begin{aligned} &\models \{\sigma_0, A\}\ \exists x : p_e(x) \wedge \varepsilon\ \{\sigma_h, B\} \\ &\quad \overset{y}{\Longleftrightarrow} \models \{\sigma_0, A\}\ p_e(y) \wedge \varepsilon\ \{\sigma_h, B\} && (y\text{—equivalent}) \\ &\quad \Longrightarrow \vdash \{\sigma_0, A\}\ p_e(y) \wedge \varepsilon\ \{\sigma_h, B\} && (\text{hypothesis}) \\ &\quad \Longrightarrow \vdash \{\sigma_0, A\}\ \exists x : p_e(x) \wedge \varepsilon\ \{\sigma_h, B\} && (\text{EQR2}) \end{aligned}$$

Similarly, by rule (EQR1), we can prove

$$\begin{aligned} &\models \{\sigma_0, A\}\ \exists x : p_c(x) \wedge \bigcirc p_f(x)\ \{\sigma_h, B\} \\ &\qquad \Longrightarrow \vdash \{\sigma_0, A\}\ \exists x : p_c(x) \wedge \bigcirc p_f(x)\ \{\sigma_h, B\}. \end{aligned}$$

Hence we can conclude

$$\models \{\sigma_0, A\}\ \exists x : p(x)\ \{\sigma_h, B\}\ \Longrightarrow\ \vdash \{\sigma_0, A\}\ \exists x : p(x)\ \{\sigma_h, B\}.$$ □

5. Verification of a mutual exclusion example

As an example, we will consider the classical mutual exclusion problem of two processes competing for a shared resource. There are both finite and infinite versions of the problem,

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

```
Initially count=0
Process A                          Process B
    while (count ≠ n)                  while (count ≠ n)
A.pc=0 wait for B.pc=0             B.pc=0 wait for A.pc=0
A.pc=1 enter critical region{      B.pc=1 enter critical region{
        access shared resource             access shared resource
        count=count+1}                     count=count+1}
    end while                          end while
end process                        end process
```

Fig. 9. Pseudocode for the terminating mutual exclusion problem

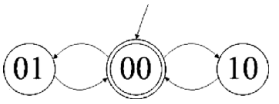


Fig. 10. A graphical representation of the mutual exclusion problem

but in this paper, we will only consider the terminating version, for which pseudocode is shown in Figure 9; see Biere *et al.* (2003) for pseudocode for the non-terminating version.

We assume that the processes are executed on a single computing unit in an interleaving manner. The *wait* statement puts a process to sleep. When all processes are asleep the scheduler tries to find a waiting condition that holds, and reactivates the corresponding process. If all waiting conditions are false, the system stalls. This terminating mutual exclusion example can be encoded in MSVL as follows (the program is deduced over an interval of length 100):

$$\begin{aligned} p \stackrel{\text{def}}{=} & \mathit{frame}(Ars, Brs, Apc, Bpc, count) \wedge (Ars = 0 \wedge Apc = 0 \wedge Brs = 0 \wedge \\ & Bpc = 0 \wedge count = 1 \wedge skip \ ; \mathit{while}\,(\neg(count = 100))\, do\, \{(await(Bpc = 0); \\ & Apc = 1 \wedge Ars = 1 \wedge count = count + 1 \wedge skip \ ; Apc = 0 \wedge Ars = 0 \wedge \\ & \mathit{if}\ (\neg(count = 100))\, then\, (count = count + 1 \wedge skip\ )\, else\, \varepsilon) \vee \\ & (await(Apc = 0); Bpc = 1 \wedge Brs = 1 \wedge count = count + 1 \wedge skip\ ; \\ & Bpc = 0 \wedge Brs = 0 \wedge \mathit{if}\ (\neg(count = 100))\, then\, (count = count + 1 \wedge \\ & skip)\, else\, \varepsilon)\}) \end{aligned}$$

where $Ars = 1\,(Brs = 1)$ means process $A\,(B)$ is in the shared resource, while $Ars = 0\,(Brs = 0)$ means process $A\,(B)$ has released the shared resource. Intuitively, *count* represents the length of the interval, which is set to 1 at the beginning. The program will terminate for *count=100*.

A graphical representation of this scenario is shown in Figure 10. The initial state has an incoming edge without a source, while the final state is circled by two rings. Obviously, we know the state 00 is both the beginning and terminating state. Each of the other edges corresponds to one of the four transitions. Thus, any sequence, for instance, 00, 01, 00, 10, 00, 01, . . . , 00, is a valid path if the path is finite.

From the graph, it is easy to see that the property of neither process being in the critical region, denoted by state 00, always holds at even states over a path. This property cannot be expressed using PLTL formulas, but can be specified using a PPTL formula. Let ϕ be

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

a PPTL formula denoting $Ars = 0 \wedge Brs = 0$. The property that formula ϕ always holds at even states regardless of odd states over a path is specified by $A \equiv \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*$. Because the length of the path is even, *viz.* 100, the property at the final state can be represented as $B \equiv \phi$. We will now prove that program p satisfies properties A and B using the axiomatic system with the triple $\{A\}\, p\, \{B\}$.

Proof. Using the state axioms and state inference rules in Tables 2 and 3 and Theorem 5, we know that program p can be reduced to normal form as $p \cong p_c^0 \wedge \bigcirc p_f^0$. Note that, for brevity, we omit the detailed step-by-step transformation from program p to its normal form and only present the resulting normal form. Furthermore, by the Interval Substitution Rule (ISR) in Table 4, we have

$$\begin{aligned}\{A\}\, p\, \{B\} &= \{\sigma_0, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p\, \{\sigma_{100}, \phi\} \\ &\Longleftrightarrow \{\sigma_0, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p_c^0 \wedge \bigcirc p_f^0\, \{\sigma_{100}, \phi\} \qquad (1)\end{aligned}$$

where

$$\begin{aligned}p_c^0 &\cong Ars = 0 \wedge Apc = 0 \wedge Brs = 0 \wedge Bpc = 0 \wedge count = 1 \\ p_f^0 &\cong lbf(Ars, Brs, Apc, Bpc, count) \wedge frame(Ars, Brs, Apc, Bpc, count) \wedge \\ &\quad (while\,(\neg(count = 100))\, do\, \{(await(Bpc = 0)\ ; Apc = 1 \wedge Ars = 1 \wedge \\ &\quad count = count + 1 \wedge skip\ ; Apc = 0 \wedge Ars = 0 \wedge if\,(\neg(count = 100)) \\ &\quad then\,(count = count + 1 \wedge skip\ )\, else\, \varepsilon) \vee (await(Apc = 0)\ ; Bpc = 1 \wedge \\ &\quad Brs = 1 \wedge count = count + 1 \wedge skip\ ; Bpc = 0 \wedge Brs = 0 \wedge \\ &\quad if\,(\neg(count = 100))\, then\,(count = count + 1 \wedge skip\ )\, else\, \varepsilon)\}).\end{aligned}$$

The normal form of property $\phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*$ is

$$\begin{aligned}&\phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^* \\ &\quad \equiv \phi \wedge \varepsilon \vee \phi \wedge \bigcirc((\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more) \qquad (NF1)\end{aligned}$$

Thus, we have

$$A_e^0 \equiv \phi, \quad A_c^0 \equiv \phi, \quad A_f^0 \equiv (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more$$

It is obvious that

$$(Ars = 0 \wedge Apc = 0 \wedge Brs = 0 \wedge Bpc = 0 \wedge count = 1) \rightarrow (Ars = 0 \wedge Brs = 0).$$

Hence $p_c^0 \rightarrow A_c^0$, that is, property A is satisfied at state s_0. By rule (ANext), we also have

$$\begin{aligned}(1) \Longleftrightarrow &\{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_f^0\, \{\sigma_{100}, \phi\} \\ &\text{and } \{\sigma_0, \phi\}\, p_c^0\, \{\sigma_0, \phi\}\end{aligned}$$

By axiom (APC) in Table 4, it is easy to see that $\{\sigma_0, \phi\}\, p_c^0\, \{\sigma_0, \phi\}$. Thus, in the following, we need to prove

$$\{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_f^0\, \{\sigma_{100}, \phi\}.$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

By the axioms and rules in Tables 2 and 3, we can obtain $p_f^0 \cong p_{fA}^0 \vee p_{fB}^0$. According to rule (ISR), we have

$$\{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_f^0\, \{\sigma_{100}, \phi\}$$
$$\Longleftrightarrow \{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_{fA}^0 \vee p_{fB}^0\, \{\sigma_{100}, \phi\}$$

where

$$\begin{aligned}
p_{fA}^0 \cong\ & lbf(Ars, Brs, Apc, Bpc, count) \wedge frame(Ars, Brs, Apc, Bpc, count) \wedge \\
& ((await(Bpc = 0)\ ; Apc = 1 \wedge Ars = 1 \wedge count = count + 1 \wedge skip\ ; \\
& Apc = 0 \wedge Ars = 0 \wedge if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ ) \\
& else\ \varepsilon) \wedge more\ ; (while\ (\neg(count = 100))\ do\ \{(await(Bpc = 0)\ ; Apc = 1 \\
& \wedge Ars = 1 \wedge count = count + 1 \wedge skip\ ; Apc = 0 \wedge Ars = 0 \wedge if \\
& (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\ else\ \varepsilon) \vee (await(Apc = 0)\ ; \\
& Bpc = 1 \wedge Brs = 1 \wedge count = count + 1 \wedge skip\ ; Bpc = 0 \wedge Brs = 0 \\
& \wedge if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\ else\ \varepsilon)\}))
\end{aligned}$$

$$\begin{aligned}
p_{fB}^0 \cong\ & lbf(Ars, Brs, Apc, Bpc, count) \wedge frame(Ars, Brs, Apc, Bpc, count) \wedge \\
& ((await(Apc = 0)\ ; Bpc = 1 \wedge Brs = 1 \wedge count = count + 1 \wedge skip\ ; \\
& Bpc = 0 \wedge Brs = 0 \wedge if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ ) \\
& else\ \varepsilon) \wedge more\ ; (while\ (\neg(count = 100))\ do\ \{(await(Bpc = 0)\ ; Apc = 1 \\
& \wedge Ars = 1 \wedge count = count + 1 \wedge skip\ ; Apc = 0 \wedge Ars = 0 \wedge if \\
& (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\ else\ \varepsilon) \vee (await(Apc = 0)\ ; \\
& Bpc = 1 \wedge Brs = 1 \wedge count = count + 1 \wedge skip\ ; Bpc = 0 \wedge Brs = 0 \\
& \wedge if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\ else\ \varepsilon)\})).
\end{aligned}$$

By rule (SSR), to prove the triple

$$\{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_f^0\, \{\sigma_{100}, \phi\},$$

we only need to prove the triple

$$\{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_{fA}^0\, \{\sigma_{100}, \phi\}$$

or the triple

$$\{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_{fB}^0\, \{\sigma_{100}, \phi\}.$$

So in a similar way, we can obtain the following:

$$\begin{aligned}
& \{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_{fA}^0\, \{\sigma_{100}, \phi\} \\
\Longleftrightarrow\ & \{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_{cA}^1 \wedge \bigcirc p_{fA}^1\, \{\sigma_{100}, \phi\} \\
& \{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_{fB}^0\, \{\sigma_{100}, \phi\} \\
\Longleftrightarrow\ & \{\sigma_1, (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more\}\, p_{cB}^1 \wedge \bigcirc p_{fB}^1\, \{\sigma_{100}, \phi\}
\end{aligned}$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

where

$$
\begin{aligned}
p^1_{cA} &\cong Brs = 0 \wedge Bpc = 0 \wedge Apc = 1 \wedge p_{Apc} \wedge Ars = 1 \wedge p_{Ars} \wedge count = 2 \wedge p_{count}\\
p^1_{fA} &\cong lbf(Ars, Brs, Apc, Bpc, count) \wedge frame(Ars, Brs, Apc, Bpc, count) \wedge\\
&\quad (Apc = 0 \wedge Ars = 0 \wedge if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\\
&\quad else\ \varepsilon\ ; (while\ (\neg(count = 100))\ do\ \{(await(Bpc = 0); Apc = 1 \wedge Ars = 1 \wedge\\
&\quad count = count + 1 \wedge skip\ ; Apc = 0 \wedge Ars = 0 \wedge if\ (\neg(count = 100))\ then\ (\\
&\quad count = count + 1 \wedge skip\ )\ else\ \varepsilon) \vee (await(Apc = 0); Bpc = 1 \wedge Brs = 1 \wedge\\
&\quad count = count + 1 \wedge skip\ ; Bpc = 0 \wedge Brs = 0 \wedge\\
&\quad if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\ else\ \varepsilon)\}))\\
p^1_{cB} &\cong Ars = 0 \wedge Apc = 0 \wedge Bpc = 1 \wedge p_{Bpc} \wedge Brs = 1 \wedge p_{Brs} \wedge count = 2 \wedge p_{count}\\
p^1_{fB} &\cong lbf(Ars, Brs, Apc, Bpc, count) \wedge frame(Ars, Brs, Apc, Bpc, count) \wedge\\
&\quad (Bpc = 0 \wedge Brs = 0 \wedge if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\\
&\quad else\ \varepsilon\ ; (while\ (\neg(count = 100))\ do\ \{(await(Bpc = 0); Apc = 1 \wedge Ars = 1 \wedge\\
&\quad count = count + 1 \wedge skip\ ; Apc = 0 \wedge Ars = 0 \wedge if\ (\neg(count = 100))\ then\ (\\
&\quad count = count + 1 \wedge skip\ )\ else\ \varepsilon) \vee (await(Apc = 0); Bpc = 1 \wedge Brs = 1 \wedge\\
&\quad count = count + 1 \wedge skip\ ; Bpc = 0 \wedge Brs = 0 \wedge\\
&\quad if\ (\neg(count = 100))\ then\ (count = count + 1 \wedge skip\ )\ else\ \varepsilon)\})).
\end{aligned}
$$

The normal form of $(\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more$ is

$$(\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more \equiv \bigcirc(\phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*)\ (NF2)$$

Thus $A^1_c \equiv true$ and $A^1_f \equiv \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*$. It is obvious that

$$(Brs = 0 \wedge Bpc = 0 \wedge Apc = 1 \wedge p_{Apc} \wedge Ars = 1 \wedge p_{Ars} \wedge count = 2 \wedge p_{count}) \rightarrow true$$

or

$$(Ars = 0 \wedge Apc = 0 \wedge Bpc = 1 \wedge p_{Bpc} \wedge Brs = 1 \wedge p_{Brs} \wedge count = 2 \wedge p_{count}) \rightarrow true$$

can be satisfied. Hence $p^1_{cA} \rightarrow A^1_c$ or $p^1_{cB} \rightarrow A^1_c$ hold at state s_1. By rule (ANext), we continue to deduce p^1_{fA} and p^1_{fB} separately at state s_2 as follows:

$$
\begin{aligned}
&\{\sigma_2, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\ p^1_{fA}\ \{\sigma_{100}, \phi\}\\
&\qquad\qquad \Longleftrightarrow \{\sigma_2, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\ p^2_{cA} \wedge \bigcirc p^0_f\ \{\sigma_{100}, \phi\}\\
&\{\sigma_2, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\ p^1_{fB}\ \{\sigma_{100}, \phi\}\\
&\qquad\qquad \Longleftrightarrow \{\sigma_2, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\ p^2_{cB} \wedge \bigcirc p^0_f\ \{\sigma_{100}, \phi\}
\end{aligned}
$$

where

$$
\begin{aligned}
p^2_{cA} &\cong Brs = 0 \wedge Bpc = 0 \wedge Apc = 0 \wedge p_{Apc} \wedge Ars = 0 \wedge p_{Ars} \wedge count = 3 \wedge p_{count}\\
p^2_{cB} &\cong Ars = 0 \wedge Apc = 0 \wedge Bpc = 0 \wedge p_{Bpc} \wedge Brs = 0 \wedge p_{Brs} \wedge count = 3 \wedge p_{count}.
\end{aligned}
$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

According to (NF1), $\phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*$ can be deduced. It is obvious that

$$(Brs = 0 \wedge Bpc = 0 \wedge Apc = 0 \wedge p_{Apc} \wedge Ars = 0 \wedge p_{Ars} \wedge count = 3 \wedge p_{count}) \rightarrow (Ars = 0 \wedge Brs = 0)$$

and

$$(Ars = 0 \wedge Apc = 0 \wedge Bpc = 0 \wedge p_{Bpc} \wedge Brs = 0 \wedge p_{Brs} \wedge count = 3 \wedge p_{count}) \rightarrow (Ars = 0 \wedge Brs = 0).$$

Hence $p^2_{cA} \rightarrow A^0_c$ and $p^2_{cB} \rightarrow A^0_c$, that is, property A is satisfied at state s_2.

We will now prove that property A is satisfied over the interval $\sigma_{(0\ldots100)}$ by induction on even states $2k$ $(0 \leqslant k \leqslant 50)$:

Base case: For $k = 0$ and $k = 1$, by the above proof, we know that property A is satisfied at states s_0 and s_2.

Induction step: We assume that A is satisfied at state s_{2k} for $1 < k \leqslant 49$. We know that two programs p^1_{fA} and p^1_{fB} repeatedly start from state s_{2k}, so we discuss the two programs separately:

— For program p^1_{fA},

$$\{\sigma_{2k}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p^1_{fA}\, \{\sigma_{100}, \phi\} \iff \{\sigma_{2k}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p^2_{cA} \wedge \bigcirc p^0_f\, \{\sigma_{100}, \phi\}$$

— For program p^1_{fB}

$$\{\sigma_{2k}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p^1_{fB}\, \{\sigma_{100}, \phi\} \iff \{\sigma_{2k}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p^2_{cB} \wedge \bigcirc p^0_f\, \{\sigma_{100}, \phi\}.$$

So the normal form of property $\phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*$ is reduced to:

$$\begin{aligned} &\phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^* \\ &\quad \equiv \phi \wedge \varepsilon \vee \phi \wedge \bigcirc((\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more) \end{aligned} \qquad \text{(NF1)}$$

Therefore,

$$\begin{aligned} A^0_e &\equiv \phi \\ A^0_c &\equiv \phi \\ A^0_f &\equiv (\bigcirc(\phi \wedge \varepsilon); (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*) \vee \bigcirc(\phi \wedge \varepsilon) \wedge \Box more. \end{aligned}$$

It is obvious that $p^2_{cA} \rightarrow A^0_c$ and $p^2_{cB} \rightarrow A^0_c$ is always true at state $s_{2k} (1 < k \leqslant 49)$. Hence, by induction, we know that property A holds at each state $s_{2k} (0 \leqslant k \leqslant 49)$. So program p satisfies property A over the interval of $\sigma_{(0\ldots98)}$.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

For $k = 50$, that is, the final state s_{100}, we now prove that property A is satisfied at state s_{100}. The triple is deduced as follows:

$$\{\sigma_{100}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p^1_{fA}\, \{\sigma_{100}, \phi\}$$
$$\Longleftrightarrow \quad \{\sigma_{100}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p_{eA} \wedge \varepsilon\, \{\sigma_{100}, \phi\} \tag{2}$$
$$\{\sigma_{100}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p^1_{fB}\, \{\sigma_{100}, \phi\}$$
$$\Longleftrightarrow \quad \{\sigma_{100}, \phi \wedge (\bigcirc \bigcirc (\phi \wedge \varepsilon))^*\}\, p_{eB} \wedge \varepsilon\, \{\sigma_{100}, \phi\} \tag{3}$$

where

$$p_{eA} \cong Brs = 0 \wedge Bpc = 0 \wedge Apc = 0 \wedge p_{Apc} \wedge Ars = 0 \wedge p_{Ars} \wedge count = 100$$
$$p_{eB} \cong Ars = 0 \wedge Apc = 0 \wedge Bpc = 0 \wedge p_{Bpc} \wedge Brs = 0 \wedge p_{Brs} \wedge count = 100.$$

From NF1, $A^0_e \equiv \phi$. It is obvious that $A^0_e \wedge \phi \equiv \phi$. It is easy to see that

$$(Brs = 0 \wedge Bpc = 0 \wedge Apc = 0 \wedge p_{Apc} \wedge Ars = 0 \wedge p_{Ars} \wedge count = 100) \rightarrow (Ars = 0 \wedge Brs = 0)$$

and

$$(Ars = 0 \wedge Apc = 0 \wedge Bpc = 0 \wedge p_{Bpc} \wedge Brs = 0 \wedge p_{Brs} \wedge count = 100) \rightarrow (Ars = 0 \wedge Brs = 0)$$

can be satisfied. So $p_{eA} \rightarrow A^0_e \wedge \phi$ and $p_{eB} \rightarrow A^0_e \wedge \phi$ hold in the final state s_{100}. By rule (AEmpty), we have

$$(2) \Longleftrightarrow \{\sigma_{100}, A^0_e \wedge \phi\}\, p_{eA}\, \{\sigma_{100}, A^0_e \wedge \phi\} \text{ and } \{\sigma_{100}, \varepsilon\}\, \varepsilon\, \{\sigma_{100}, \varepsilon\}$$
$$(3) \Longleftrightarrow \{\sigma_{100}, A^0_e \wedge \phi\}\, p_{eB}\, \{\sigma_{100}, A^0_e \wedge \phi\} \text{ and } \{\sigma_{100}, \varepsilon\}\, \varepsilon\, \{\sigma_{100}, \varepsilon\}.$$

Axioms (AEM) and (APC) mean (2) and (3) are true. Thus, program p satisfies properties A and B over the whole interval $\sigma_{(0\ldots100)}$. □

A proof assistant for the axiom system has been developed in Xidian University as a semi-automatic tool for checking whether a particular application of a proof rule in the axiom system is correct. Rather than constructing it from scratch, we embedded the axiom system within an existing proof tool, which is called the Prototype Verification System (PVS) (Owre *et al.* 1999; Shankar *et al.* 1999). Proofs within the assistant follow almost the same pattern as proofs carried out 'by hand'.

The specification of mutual exclusion can be formalised as follows (due to space limitations, the construction of the proof assistant and the complete proof are omitted here, but a rough description of the verification of the mutual exclusion example using the proof assistant is shown in Appendix B – further details can be found in Ma *et al.* (2010)):

$MutualExclusion : theorem\ H(sigma)(0, ((Ars = const(0) \wedge Brs = const(0))$
$\wedge chopstar(O(O((Ars = const(0) \wedge Brs = const(0)) \wedge empty))))) ((frame(Ars)$
$\wedge frame(Brs) \wedge frame(Apc) \wedge frame(Bpc) \wedge frame(count))(\hat{}((Ars = const(0)$
$\wedge Apc = const(0) \wedge Brs = const(0) \wedge Bpc = const(0) \wedge count = const(1)) \wedge$
$skip, while(count/ = const(100), \vee(\hat{}(\hat{}(await(Bpc = const(0)), (Apc = const(1) \wedge Ars = const(1)$
$\wedge count = count + const(1)) \wedge skip), (Apc = const(0) \wedge Ars = const(0)) \wedge ifelse(count/ =$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

$const(100), count = count + const(1) \wedge skip, empty)),$
$\hat{}(\hat{}(await(Apc = const(0)), (Bpc = const(1) \wedge Brs = const(1) \wedge count = count + const(1)) \wedge$
$skip), (Bpc = const(0) \wedge Brs = const(0)) \wedge ifelse(count/ = const(100),$
$count = count + const(1) \wedge skip, empty)))))))(100, Ars = const(0) \wedge Brs = const(0))$

where *sigma* is a finite interval of length 100, *const(n)* means the constant n, the symbol "^" represents ';' , *H(sigma)(k,A)(p)(h,B)* denotes the triple $\{\sigma_k, A\}\, p\, \{\sigma_h, B\}$, and the other symbols are as defined in the previous sections.

6. Conclusions

We have presented an axiomatic semantics for the interval temporal logic programming language MSVL. We have also proved the soundness and relative completeness of the system with respect to an operational model of MSVL. The proof system can be used to prove not only the partial correctness of framed programs but also run-time properties such as safety properties. Moreover, a verifier based on PVS for the axiom system has been developed, and it has been used to verify run-time properties within a mutual exclusive system semi-automatically. However, the proof of the completeness is restricted to programs that terminate, so in the next stage of our research, we will investigate the completeness of the proof system for non-terminating programs. We will also investigate ways to improve the efficiency of the tool and verify more and larger programs.

Appendix A. Proofs of Theorems 5, 6 and 7

Proof of Theorem 5. The proof proceeds by induction on the structure of statements in MSVL – the state inference rule (R1) is used by default:

— p is ε:
The conclusion is straightforward.

— p is the *assignment* statement $x = e$:
Then

$$\begin{aligned} x = e &\cong x = e \wedge \mathsf{true} && \text{(A20, R3)} \\ &\cong x = e \wedge (\varepsilon \vee \mathsf{more}) && \text{(A20, R3)} \\ &\cong (x = e \wedge \varepsilon) \vee (x = e \wedge \mathsf{more}) && \text{(A20, R3)} \\ &\cong (x = e \wedge \varepsilon) \vee (x = e \wedge \bigcirc \mathsf{true}) \end{aligned}$$

— p is the *positive immediate statement* $x \Leftarrow e$:
Then

$$\begin{aligned} x \Leftarrow e &\cong x \Leftarrow e \wedge \mathsf{true} && \text{(A20, R3)} \\ &\cong x \Leftarrow e \wedge (\varepsilon \vee \mathsf{more}) && \text{(A20, R3)} \\ &\cong (x \Leftarrow e \wedge \varepsilon) \vee (x \Leftarrow e \wedge \mathsf{more}) && \text{(A20, R3)} \\ &\cong (x = e \wedge p_x \wedge \varepsilon) \vee (x = e \wedge p_x \wedge \bigcirc \mathsf{true}) \end{aligned}$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

— p is the *state frame* $\mathsf{lbf}(x)$:
Then

$$\begin{aligned}
\mathsf{lbf}(x) &\cong \mathsf{lbf}(x) \wedge \mathsf{true} && \text{(A20, R3)}\\
&\cong \mathsf{lbf}(x) \wedge (\varepsilon \vee \mathsf{more}) && \text{(A20, R3)}\\
&\cong (\mathsf{lbf}(x) \wedge \varepsilon) \vee (\mathsf{lbf}(x) \wedge \mathsf{more}) && \text{(A20, R3)}\\
&\cong ((p_x \vee (\neg p_x \wedge x = \ominus x)) \wedge \varepsilon) \vee && (\mathsf{lbf}(x) \stackrel{\text{def}}{=} p_x \vee (\neg p_x \wedge (x = \ominus x)))\\
&\quad ((p_x \vee (\neg p_x \wedge x = \ominus x)) \wedge \bigcirc\mathsf{true})\\
&\cong (p_x \wedge \varepsilon) \vee (\neg p_x \wedge x = \ominus x \wedge \varepsilon) \vee\\
&\quad (p_x \wedge \bigcirc\mathsf{true}) \vee (\neg p_x \wedge x = \ominus x \wedge \bigcirc\mathsf{true})
\end{aligned}$$

Since $\ominus x$ has been evaluated to a constant n ($n \in N_0$) at the previous state, we can regard $x = \ominus x$ as $x = n$.

— p is the *interval frame* $\mathsf{frame}(x)$:
Then

$$\begin{aligned}
\mathsf{frame}(x) &\cong \mathsf{frame}(x) \wedge \mathsf{true} && \text{(A20, R3)}\\
&\cong \mathsf{frame}(x) \wedge (\varepsilon \vee \mathsf{more}) && \text{(A20, R3)}\\
&\cong (\mathsf{frame}(x) \wedge \varepsilon) \vee (\mathsf{frame}(x) \wedge \mathsf{more}) && \text{(A20, R3)}\\
&\cong \varepsilon \vee \bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x))) && \text{(A5, A6)}
\end{aligned}$$

— p is the *conjunction* statement $p \wedge q$:
Suppose

$$\begin{aligned}
p &\cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)\\
q &\cong (q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f)
\end{aligned}$$

where p_e, p_c, q_e, q_c are present components. Then

$$\begin{aligned}
p \wedge q &\cong ((p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)) \wedge ((q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f)) && \text{(hypothesis)}\\
&\cong (p_e \wedge q_e \wedge \varepsilon) \vee (p_c \wedge q_c \wedge \bigcirc(p_f \wedge q_f))) && \text{(A20, R3)}
\end{aligned}$$

— p is the *selection* statement $p \vee q$:
Suppose

$$\begin{aligned}
p &\cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)\\
q &\cong (q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f)
\end{aligned}$$

Then

$$p \vee q* \cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f) \vee (q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f)) \qquad \text{(hypothesis)}$$

— p is a statement in the form of $\bigcirc p$:
This is already in its normal form.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

— p is the *always* statement $\Box p$:
Suppose $p \cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)$. Then

$$\begin{aligned}
\Box p &\cong \Box p \wedge \mathsf{true} && \text{(A20, R3)}\\
&\cong \Box p \wedge (\varepsilon \vee \mathsf{more}) && \text{(A20, R3)}\\
&\cong (\Box p \wedge \varepsilon) \vee (\Box p \wedge \mathsf{more}) && \text{(A20, R3)}\\
&\cong (p \wedge \varepsilon) \vee (p \wedge \bigcirc \Box p) && \text{(A3, A4)}\\
&\cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc (p_f \wedge \Box p)) && \text{(hypothesis, A20, R3)}
\end{aligned}$$

— p is the *sequential* statement $p \ ; q$:
Suppose $p \cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)$. Then

$$p \ ; q \cong ((p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)) \ ; q$$

In fact, $(p_e \wedge \varepsilon)$ and $(p_c \wedge \bigcirc p_f)$ are executed exclusively. So we discuss them separately:

Case 1: $p \cong (p_e \wedge \varepsilon)$:

$$\begin{aligned}
(p_e \wedge \varepsilon) \ ; q &\cong p_e \wedge (\varepsilon \ ; q) && \text{(A9)}\\
&\cong p_e \wedge q && \text{(A8)}
\end{aligned}$$

Case 2: $p \cong (p_c \wedge \bigcirc p_f)$:

$$\begin{aligned}
(p_c \wedge \bigcirc p_f) \ ; q &\cong p_c \wedge (\bigcirc p_f \ ; q) && \text{(A9)}\\
&\cong p_c \wedge \bigcirc (p_f \ ; q) && \text{(A7)}
\end{aligned}$$

— p is the *existential quantification* statement $\exists x : p(x)$:
Let $D = \{d_1, \ldots, d_n\}$ be a finite set of data. Suppose

$$p(x) \cong (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x)),$$

where $p_e(x), p_c(x)$ are present components. Then

$$\exists x : p(x) \cong (\exists x : p_e(x) \wedge \varepsilon) \vee (\exists x : p_c(x) \wedge \bigcirc \exists x : \mathsf{p_f}(x)) \qquad \text{(R2)}$$

— p is the *conditional* statement `if` b `then` p `else` q:
Suppose

$$\begin{aligned}
p &\cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)\\
q &\cong (q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f).
\end{aligned}$$

Then

$$\texttt{if } b \texttt{ then } p \texttt{ else } q \cong (b \wedge p) \vee (\neg b \wedge q) \qquad \text{(A18)}$$

Therefore, if b is true, the construct is deduced to $p \cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)$. Otherwise, it is deduced to $q \cong (q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f)$.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

— p is the *iterative* statement `while` b `do` p:
Suppose $p \cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f)$. Then

$$\begin{aligned} \texttt{while}\ b\ \texttt{do}\ p &\cong \texttt{if}\ b\ \texttt{then}\ (p \wedge \mathsf{more})\ \texttt{;}\ \texttt{while}\ b\ \texttt{do}\ p\ \texttt{else}\ \varepsilon && \text{(A19)} \\ &\cong (b \wedge ((p \wedge \mathsf{more})\ \texttt{;}\ \texttt{while}\ b\ \texttt{do}\ p)) \vee (\neg b \wedge \varepsilon) && \text{(A18)} \end{aligned}$$

For $b = \mathsf{true}$, the construct can be deduced to

$$\begin{aligned} (p \wedge \mathsf{more})\ \texttt{;}\ \texttt{while}\ b\ \texttt{do}\ p &\cong (\mathsf{p_c} \wedge \bigcirc \mathsf{p_f} \wedge \bigcirc \mathsf{true})\ \texttt{;}\ \texttt{while}\ b\ \texttt{do}\ p && \text{(hypothesis)} \\ &\cong \mathsf{p_c} \wedge \bigcirc(\mathsf{p_f}\ \texttt{;}\ \texttt{while}\ b\ \texttt{do}\ p) && \text{(A7, A9)} \end{aligned}$$

Otherwise, it is deduced to ε.

— p is the *parallel* statement $p \parallel q$:
Suppose

$$\begin{aligned} p &\cong (p_e \wedge \varepsilon) \vee (p_c \wedge \bigcirc p_f) \\ q &\cong (q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f) \\ \mathsf{true} &\cong \varepsilon \vee \bigcirc \mathsf{true}. \end{aligned}$$

Then

$$p \parallel q \cong (p \wedge (q\ \texttt{;}\ \mathsf{true})) \vee (q \wedge (p\ \texttt{;}\ \mathsf{true})) \qquad \text{(A10)}$$

In this proof, by hypothesis, p, q and true can be reduced to normal form, and we have already proved that the conjunction, selection and sequential statements can be reduced to normal form, so it is easy to see that $p \parallel q$ can also be reduced to normal form.

— p is the *projection* statement $(p_1, \ldots, p_m)\ \mathsf{prj}\ q$:
Suppose

$$\begin{aligned} p_i &\cong (p_{e_i} \wedge \varepsilon) \vee (p_{c_i} \wedge \bigcirc p_{f_i})\ (1 \leqslant i \leqslant m) \\ q &\cong (q_e \wedge \varepsilon) \vee (q_c \wedge \bigcirc q_f) \end{aligned}$$

where $p_{e_i}, p_{c_i}, q_e, q_c$ are present components. We discuss the construct in the following four situations.

Case 1: $p_1 \cong p_{e_1} \wedge \varepsilon$ and $q \cong q_e \wedge \varepsilon$.
Then

$$\begin{aligned} (p_1, \ldots, p_m)\ \mathsf{prj}\ q &\cong (p_{e_1} \wedge \varepsilon, p_2, \ldots, p_m)\ \mathsf{prj}\ (q_e \wedge \varepsilon) && \text{(hypothesis)} \\ &\cong p_{e_1} \wedge q_e \wedge (\varepsilon, p_2, \ldots, p_m)\ \mathsf{prj}\ \varepsilon && \text{(A15, A16)} \\ &\cong p_{e_1} \wedge q_e \wedge (\varepsilon\ \texttt{;}\ p_2\ \texttt{;}\ \ldots\ \texttt{;}\ p_m) && \text{(A13)} \\ &\cong p_{e_1} \wedge q_e \wedge (p_2\ \texttt{;}\ \ldots\ \texttt{;}\ p_m) && \text{(A8)} \end{aligned}$$

Since the conjunction and selection constructs have already been proved to have normal form, Case 1 can be reduced to normal form.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Case 2: $p_1 \cong p_{e_1} \wedge \varepsilon$ and $q \cong q_c \wedge \bigcirc q_f$.
Then

$$\begin{aligned}
(p_1, \ldots, p_m)\ \mathsf{prj}\ q \cong\ & (p_{e_1} \wedge \varepsilon, p_2, \ldots, p_m)\ \mathsf{prj}\ (q_c \wedge \bigcirc q_f) && \text{(hypothesis)}\\
\cong\ & p_{e_1} \wedge q_c \wedge (\varepsilon, p_2, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f && \text{(A15, A16)}\\
\cong\ & p_{e_1} \wedge q_c \wedge (p_2, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f && \text{(A14)}
\end{aligned}$$

(i) If $p_2 \cong \mathsf{p_{c2}} \wedge \bigcirc \mathsf{p_{f2}}$,

$$\begin{aligned}
& p_{e_1} \wedge q_c \wedge (p_2, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f\\
& \quad \cong\ p_{e_1} \wedge q_c \wedge (p_{c_2} \wedge \bigcirc p_{f_2}, p_3, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f && \text{(hypothesis)}\\
& \quad \cong\ p_{e_1} \wedge q_c \wedge p_{c_2} \wedge (\bigcirc p_{f_2}, p_3, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f && \text{(A15)}\\
& \quad \cong\ p_{e_1} \wedge q_c \wedge p_{c_2} \wedge \bigcirc (p_{f_2}\ ;\ (p_3, \ldots, p_m)\ \mathsf{prj}\ q_f && \text{(A17)}
\end{aligned}$$

(ii) Otherwise, if $p_2 \cong p_{e_2} \wedge \varepsilon$, the construct can be deduced to

$$\begin{aligned}
& p_{e_1} \wedge q_c \wedge (p_2, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f\\
& \quad \cong\ p_{e_1} \wedge q_c \wedge (p_{e_2} \wedge \varepsilon, p_3, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f && \text{(hypothesis)}\\
& \quad \cong\ p_{e_1} \wedge q_c \wedge p_{e_2} \wedge (p_3, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f && \text{(A14, A15)}
\end{aligned}$$

By hypothesis, the normal forms of $p_3, \ldots, p_m$ are $(p_{e_i} \wedge \varepsilon) \vee (p_{c_i} \wedge \bigcirc p_{f_i})$ $(3 \leqslant i \leqslant m)$. Since the form $(p_{c_t} \wedge \bigcirc p_{f_t}, p_{t+1}, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f$ $(1 \leqslant t \leqslant m)$ has already been proved to have the normal form (which is shown in (i)) we now only need to consider the construct $p_i \cong p_{e_i} \wedge \varepsilon$.

$$\begin{aligned}
& p_{e_1} \wedge q_c \wedge p_{e_2} \wedge (p_3, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q_f\\
& \quad \cong p_{e_1} \wedge q_c \wedge p_{e_2} \wedge ((p_{e_3} \wedge \varepsilon, \ldots, p_{e_m} \wedge \varepsilon)\ \mathsf{prj}\ \bigcirc q_f) && \text{(hypothesis)}\\
& \quad \cong p_{e_1} \wedge q_c \wedge p_{e_2} \wedge p_{e_3} \wedge \ldots \wedge p_{e_m} \wedge (\varepsilon\ \mathsf{prj}\ \bigcirc q_f) && \text{(A14, A15)}\\
& \quad \cong p_{e_1} \wedge q_c \wedge p_{e_2} \wedge p_{e_3} \wedge \ldots \wedge p_{e_m} \wedge \bigcirc q_f && \text{(A12)}
\end{aligned}$$

Case 3: $p_1 \cong p_{c_1} \wedge \bigcirc p_{f_1}$ and $q \cong q_e \wedge \varepsilon$.
Then

$$\begin{aligned}
(p_1, \ldots, p_m)\ \mathsf{prj}\ q \cong\ & (p_{c_1} \wedge \bigcirc p_{f_1}, p_2, \ldots, p_m)\ \mathsf{prj}\ (q_e \wedge \varepsilon) && \text{(hypothesis)}\\
\cong\ & \mathsf{p_{c1}} \wedge q_e \wedge (\bigcirc p_{f_1}, p_2, \ldots, p_m)\ \mathsf{prj}\ (\varepsilon) && \text{(A15, A16)}\\
\cong\ & p_{c_1} \wedge q_e \wedge (\bigcirc p_{f_1}\ ;\ p_2\ ;\ \ldots\ ;\ p_m) && \text{(A13)}\\
\cong\ & p_{c_1} \wedge q_e \wedge \bigcirc (p_{f_1}\ ;\ p_2\ ;\ \ldots\ ;\ p_m) && \text{(A7)}
\end{aligned}$$

Case 4: $p_1 \cong p_{c_1} \wedge \bigcirc p_{f_1}$ and $q \cong q_c \wedge \bigcirc q_f$
Then

$$\begin{aligned}
(p_1, \ldots, p_m)\ \mathsf{prj}\ q \cong\ & (p_{c_1} \wedge \bigcirc p_{f_1}, p_2, \ldots, p_m)\ \mathsf{prj}\ (q_c \wedge \bigcirc q_f) && \text{(hypothesis)}\\
\cong\ & p_{c_1} \wedge q_c \wedge (\bigcirc p_{f_1}, p_2, \ldots, p_m)\ \mathsf{prj}\ (\bigcirc q_f) && \text{(A15, A16)}\\
\cong\ & p_{c_1} \wedge q_e \wedge \bigcirc (p_{f_1}\ ;\ (p_2, \ldots, p_m)\ \mathsf{prj}\ q_f) && \text{(A17)}
\end{aligned}$$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Therefore, we can conclude that any framed program can be deduced to its normal form by the state axioms and state inference rules. □

Proof of Theorem 6.

(1) Each axiom in Table 2 corresponds to a logic law in the model theory of MSVL, so all of them are valid.

(2) We now prove that the inference rules in Table 3 are valid in the model theory.

(R1): $p \equiv q \Longrightarrow prog[p] \equiv prog[q/p]$

This result is straightforward since it is the congruence property of the equivalence relation.

(R2): $p(x) \equiv (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x))$
$\Longrightarrow \exists x : p(x) \equiv (\exists x : p_e(x) \wedge \varepsilon) \vee (\exists x : p_c(x) \wedge \bigcirc \exists x : p_f(x))$.

The proof can be found in Duan (1996) and is omitted here.

(R3): $\models P \Longrightarrow \models \Box P$

Let σ be an arbitrary interval. Suppose $\sigma \models P$, that is, $(\sigma, 0, 0, |\sigma|) \models P$. Since P is valid, P can be satisfied by every suffix interval of σ, namely, for all $0 \leqslant k \preceq \sigma$, we have $(\sigma, 0, k, |\sigma|) \models P$. So we have $(\sigma, 0, 0, |\sigma|) \models \Box P$, that is, $\sigma \models \Box P$. Thus, under the condition of $\models P$, we get $\models P \rightarrow \Box P$, and, by *modus ponens*, we have $\models \Box P$. □

Proof of Theorem 7.

(1) In presenting the operational semantics of MSVL in Yang and Duan (2008), we have given a set of semantic equivalence rules for reducing a framed program to its normal form – see Figure 11. Yang and Duan (2008) also proved, by means of the semantic equivalence rules in Figure 11, that any framed program can be reduced to normal form.

In our axiom system, the state axioms and state inference rules are the same as those in Figure 11, except that they are formalised as axioms. Hence, the following results clearly hold:

(1) $\mathsf{lbf}(x) \wedge x = e \equiv x \Leftarrow e(\text{Ass}=) \Longrightarrow \mathsf{lbf}(x) \wedge x = e \cong x \Leftarrow e(\text{A1})$
(2) $\mathsf{lbf}(x) \wedge x \Leftarrow e \equiv x \Leftarrow e(\text{Ass} \Leftarrow) \Longrightarrow \mathsf{lbf}(x) \wedge x \Leftarrow e \cong x \Leftarrow e(\text{A2})$
(3) $\Box p \wedge \mathsf{more} \equiv p \wedge \bigcirc \Box p(\text{Alw}(2)) \Longrightarrow \Box p \wedge \mathsf{more} \cong p \wedge \bigcirc \Box p(\text{A3})$
(4) $\Box p \wedge \varepsilon \equiv p \wedge \varepsilon(\text{Alw}(1)) \Longrightarrow \Box p \wedge \varepsilon \cong p \wedge \varepsilon(\text{A4})$
(5) $\mathsf{frame}(x) \wedge \mathsf{more} \equiv \bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x))(\text{Fr}(2))$
$\Longrightarrow \mathsf{frame}(x) \wedge \mathsf{more} \cong \bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x))(\text{A5})$
(6) $\mathsf{frame}(x) \wedge \varepsilon \equiv \varepsilon(\text{Fr}(2)) \Longrightarrow \mathsf{frame}(x) \wedge \varepsilon \cong \varepsilon(\text{A6})$
(7) $\bigcirc p \,;\, q \equiv \bigcirc(p \;;\; q)(\text{Chop}(2)) \Longrightarrow \bigcirc p \,;\, q \cong \bigcirc(p \;;\; q)(\text{A7})$
(8) $\varepsilon \,;\, q \equiv q(\text{Chop}(3)) \Longrightarrow \varepsilon \,;\, q \cong q(\text{A8})$
(9) $(w \wedge p) \,;\, q \equiv w \wedge (p \;;\; q)(\text{Chop}(1)) \Longrightarrow (w \wedge p) \,;\, q \cong w \wedge (p \;;\; q)(\text{A9})$

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

ASS	$(Ass=)\ \mathsf{lbf}(x) \wedge x = e \equiv x \Leftarrow e\ (Ass \Leftarrow)\ \mathsf{lbf}(x) \wedge x \Leftarrow e \equiv x \Leftarrow e$
NEXT	$\bigcirc p \equiv \bigcirc p \wedge \mathsf{more}$
ALW	(1) $\Box p \wedge \varepsilon \equiv p \wedge \varepsilon$. (2) $\Box p \wedge \mathsf{more} \equiv p \wedge \bigcirc \Box p$
CHOP	(1) $(w \wedge p)\ ;\ q \equiv w \wedge (p\ ;\ q)$. (2) $\bigcirc p\ ;\ q \equiv \bigcirc(p\ ;\ q)$. (3) $\varepsilon\ ;\ q \equiv q$
IF	$\texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ q \equiv (b \wedge p) \vee (\neg b \wedge q)$
WHL	$\texttt{while}\ b\ \texttt{do}\ p \equiv \texttt{if}\ b\ \texttt{then}\ ((p \wedge \mathsf{more})\ ;\ \texttt{while}\ b\ \texttt{do}\ p)\ \texttt{else}\ \varepsilon$
PAR	$p \mid\mid q \equiv ((q\ ;\ \mathsf{true}) \wedge p) \vee (q \wedge (p\ ;\ \mathsf{true}))$
FR	(1) $\mathsf{frame}(x) \wedge \varepsilon \equiv \varepsilon$ (2) $\mathsf{frame}(x) \wedge \mathsf{more} \equiv \bigcirc(\mathsf{lbf}(x) \wedge \mathsf{frame}(x))$
FR	(1) $(p_1, \ldots, p_m)\ \mathsf{prj}\ \varepsilon \equiv p_1\ ;\ \ldots\ ;\ p_m$ (2) $\varepsilon\ \mathsf{prj}\ q \equiv q$ (3) $(p_1, \ldots, p_{i-1}, w \wedge \varepsilon, p_{i+1}, \ldots, p_m)\ \mathsf{prj}\ q \equiv$ $\quad (p_1, \ldots, p_{i-1}, w \wedge p_{i+1}, \ldots, p_m)\ \mathsf{prj}\ q$ (4) $(w \wedge p_1, p_2, \ldots, p_m)\ \mathsf{prj}\ q \equiv w \wedge (p_1, \ldots, p_m)\ \mathsf{prj}\ q$ (5) $(p_1, \ldots, p_m)\ \mathsf{prj}\ (w \wedge q) \equiv w \wedge (p_1, \ldots, p_m)\ \mathsf{prj}\ q$ (6) $(\bigcirc p_1, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q \equiv \bigcirc(p_1; (p_2, \ldots, p_m)\ \mathsf{prj}\ q)$
LOC	$\exists x : p(x) \stackrel{y}{=} p(y)$, if $p(y)$ is a renamed program of $\exists x : p(x)$
LOC-NEXT	$\bigcirc(\exists x : p(x)) \equiv \exists x : \bigcirc p(x)$
CONG I	$\dfrac{q \equiv q'}{prog \equiv prog[q'/q]}$

Fig. 11. Semantic equivalence rules for framed programs

(10) $p \mid\mid q \equiv (p \wedge (q\ ;\ \mathsf{true})) \vee (q \wedge (p\ ;\ \mathsf{true}))$ (PAR)
$\Longrightarrow p \mid\mid q \cong (p \wedge (q\ ;\ \mathsf{true})) \vee (q \wedge (p\ ;\ \mathsf{true}))$(A10)
(11) $\bigcirc p \wedge q \equiv \bigcirc p \wedge q \wedge \mathsf{more}$(NEXT) $\Longrightarrow \bigcirc p \wedge q \cong \bigcirc p \wedge q \wedge \mathsf{more}$(A11)
(12) $\varepsilon\ \mathsf{prj}\ q \equiv q$(PRJ(2)) $\Longrightarrow \varepsilon\ \mathsf{prj}\ q \cong q$(A12)
(13) $(p_1, \ldots, p_m)\ \mathsf{prj}\ \varepsilon \equiv p_1\ ;\ \ldots\ ;\ p_m$(PRJ(1))
$\Longrightarrow (p_1, \ldots, p_m)\ \mathsf{prj}\ \varepsilon \cong p_1\ ;\ \ldots\ ;\ p_m$(A13)
(14) $(p_1, \ldots, p_{i-1}, w \wedge \varepsilon, p_{i+1}, \ldots, p_m)\ \mathsf{prj}\ q$
$\quad \equiv (p_1, \ldots, p_{i-1}, w \wedge p_{i+1}, \ldots, p_m)\ \mathsf{prj}\ q$(PRJ(3))
$\Longrightarrow (p_1, \ldots, p_{i-1}, w \wedge \varepsilon, p_{i+1}, \ldots, p_m)\ \mathsf{prj}\ q$
$\quad \cong (p_1, \ldots, p_{i-1}, w \wedge p_{i+1}, \ldots, p_m)\ \mathsf{prj}\ q$(A14)
(15) $(w \wedge p_1, p_2, \ldots, p_m)\ \mathsf{prj}\ q \equiv w \wedge (p_1, \ldots, p_m)\ \mathsf{prj}\ q$(PRJ(4))
$\Longrightarrow (w \wedge p_1, p_2, \ldots, p_m)\ \mathsf{prj}\ q \cong w \wedge (p_1, \ldots, p_m)\ \mathsf{prj}\ q$(A15)
(16) $(p_1, \ldots, p_m)\ \mathsf{prj}\ (w \wedge q) \equiv w \wedge (p_1, \ldots, p_m)\ \mathsf{prj}\ q$(PRJ(5))
$\Longrightarrow (p_1, \ldots, p_m)\ \mathsf{prj}\ (w \wedge q) \cong w \wedge (p_1, \ldots, p_m)\ \mathsf{prj}\ q$(A16)
(17) $(\bigcirc p_1, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q \equiv \bigcirc(p_1\ ;\ (p_2, \ldots, p_m)\ \mathsf{prj}\ q)$(PRJ(6))
$\Longrightarrow (\bigcirc p_1, \ldots, p_m)\ \mathsf{prj}\ \bigcirc q \cong \bigcirc(p_1\ ;\ (p_2, \ldots, p_m)\ \mathsf{prj}\ q)$(A17)
(18) $\texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ q \equiv (b \wedge p) \vee (\neg b \wedge q)$(IF)
$\Longrightarrow \texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ q \cong (b \wedge p) \vee (\neg b \wedge q)$(A18)
(19) $\texttt{while}\ b\ \texttt{do}\ p \equiv \texttt{if}\ b\ \texttt{then}\ (p \wedge \mathsf{more}\ ;\ \texttt{while}\ b\ \texttt{do}\ p)\ \texttt{else}\ \varepsilon$(WHL)
$\Longrightarrow \texttt{while}\ b\ \texttt{do}\ p \cong \texttt{if}\ b\ \texttt{then}\ (p \wedge \mathsf{more}\ ;\ \texttt{while}\ b\ \texttt{do}\ p)\ \texttt{else}\ \varepsilon$(A19)

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

(20) $p \equiv q \Longrightarrow prog[p] \equiv prog[q/p]$(Cong I)
$\Longrightarrow p \cong q \Longrightarrow prog[p] \cong prog[q/p]$(R1)

Therefore, it is obvious that for any reduction sequence, such as $p \equiv p_1 \equiv p_2 \equiv \ldots \equiv p_n$, carried out by means of the semantic equivalence rules, there is a corresponding deduction sequence, such as $p \cong p_1 \cong p_2 \cong \ldots \cong p_n$, using the corresponding state axioms and state inference rules. Therefore, $p \equiv q \Longrightarrow p \cong q$ holds for a quantifier-free program p.

(2) We now consider the existence quantification statement $\exists x : p(x)$, and prove that $\exists x : p(x) \equiv q \Longrightarrow \exists x : p(x) \cong q$ for some program q. For any quantifier free program $p(x)$, we assume

$$p(x) \equiv (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x)) \Longrightarrow p(x) \cong (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x)).$$

So we have

$$\exists x : p(x) \equiv (\exists x : p_e(x) \wedge \varepsilon) \vee (\exists x : p_c(x) \wedge \bigcirc \exists x : p_f(x)) \qquad \text{(Loc-Next)}$$

Furthermore, by hypothesis, we have

$$p(x) \cong (p_e(x) \wedge \varepsilon) \vee (p_c(x) \wedge \bigcirc p_f(x))$$

So

$$\exists x : p(x) \cong \qquad (\exists x : p_e(x) \wedge \varepsilon) \vee (\exists x : p_c(x) \wedge \bigcirc \exists x : p_f(x)) \qquad \text{(R2)}$$

Hence, we have $\exists x : p(x) \equiv q \Longrightarrow \exists x : p(x) \cong q$. Therefore, for any MSVL program p, we have $p \equiv q \Longrightarrow p \cong q$ holds. □

Appendix B. Using the proof assistant for the mutual exclusion example

In this appendix we outline how the proof assistant can be used for the mutual exclusion example in Section 5. Figure 12 shows the proof assistant's interface, which has two buffers: the specification buffer in the top window and the *pvs* buffer in the bottom. The specification of the mutual exclusion problem is shown in the top buffer and its proof is shown in the bottom. Once the proof is initiated, the main goal is displayed in the *pvs* buffer followed by a *Rule?* prompt. The user commands are then typed in at this prompt. The basic objective of the proof is to generate a proof tree of *sequents* (that is, $\vdash$) in which all of the leaves are trivially true.

When proving a theorem, if Tcl/Tk is available, a graphical proof tree is displayed, as in Figure 13. The proving goals are shown as red nodes; the proved goals are shown as blue nodes; and the current goal is shown as a purple node surrounded by a circle. When a theorem is proved, the whole proof tree will turn blue.

Figure 14 shows the interface after the mutual exclusion example has been proved in the proof assistant. To complete the goal, the proof assistant creates three subgoals, named *Mutual Exclusion.1*, *Mutual Exclusion.2* and *Mutual Exclusion.3*. The text *This completes the proof of Mutual Exclusion.3* shows that it has successfully completed the third subgoal

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

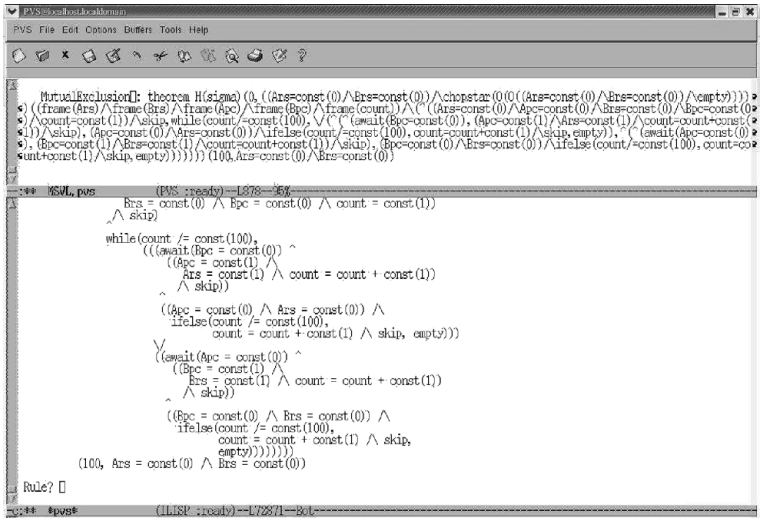


Fig. 12. The specification of the mutual exclusion

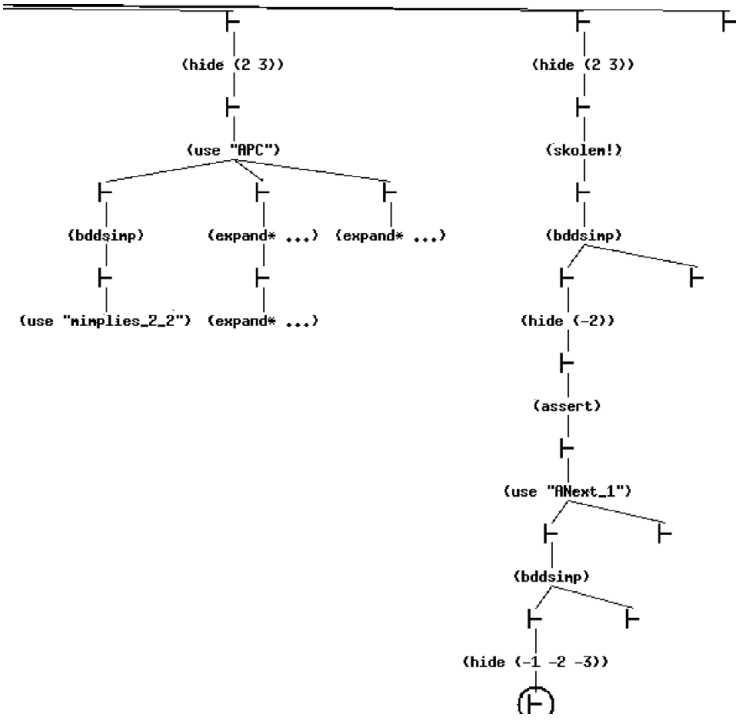


Fig. 13. Part of the proof tree for the mutual exclusion example during the proof

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

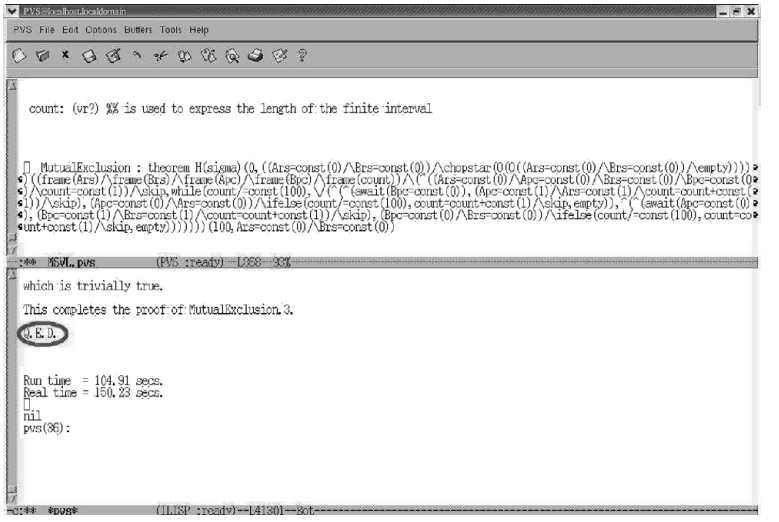


Fig. 14. The proof assistant interface at the end of the verification of the mutual exclusion example

of the example. When the whole theorem has been proved, the *pvs* buffer will show *Q.E.D*. The *Run time* and *Real time* times shown in the display are the CPU time and actual elapsed time, respectively, for the complete proof.

References

Abadi, M. and Manna, Z. (1989) Temporal Logic Programming. *Journal of Symbolic Computation* **8** 277–295.

Biere, A., Cimati, A., Clark, E. M., Strichman, O. and Zhu, Y. S. (2003) Bounded Model Checking. *Advances in Computers* **58**.

Brzoska, C. (1993) Temporal Logic Programming with Bounded Universal Modality Goal. *Proceedings of the Tenth International Conference on Logic Programming*, MIT Press 239–256.

Brzoska, C. (1998) Programming in Metric Temporal Logic. *Theoretical Computer Science* **202** (1–2) 55–125.

Clarke, E. M., Grumberg, O. and Peled, D. (1999) *Model Checking*, MIT Press.

Cook, S. A. (1978) Soundness and Completeness of An Axiom System for Program Verification. *SIAM Journal on Computing* **7** 70–90.

Duan, Z. (1996) *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming*, Ph. D. Thesis, University of Newcastle upon Tyne. (Technical Report No. 556.)

Duan, Z. and Koutny, M. (2004) A Framed Temporal Logic Programming Language. *Journal of Computer Science and Technology* **19** 341–351.

Duan, Z., Koutny, M. and Holt, C. (1994) Projection in temporal logic programming. Proceedings of Logic Programming and Automatic Reasoning. *Springer-Verlag Lecture Notes in Artificial Intelligence* **822** 333–344.

Duan, Z., Tian, C. and Zhang, L. (2008a) A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. *Acta Informatica* **45** (1) 43–78.

Duan, Z., Yang, X. and Koutny, M. (2008b) Framed Temporal Logic Programming. *Science of Computer Programming* **70** (1) 31–61.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Fujita, M., Kono, S., Tanaka, H. and Moto-oka, T. (1986) Tokio: Logic programming language based on temporal logic and its compilation to PROLOG. In: Third International Conference on Logic Programming. *Springer-Verlag Lecture Notes in Computer Science* **225** 695–709.

Gabbay, D. M. (1987) Modal and Temporal Logic Programming. *Temporal Logics and Their Applications*, Academic Press 197–237.

Hale, R. (1987) Temporal Logic Programming. *Temporal Logics and Their Applications*, Academic Press 91–119.

Hansen, M. R. and Zhao, C. (1997) Duration Calculus: Logical Foundations. *Formal Aspects of Computing* **9** 283–330.

Hoare, C. A. R. (1969) An axiomatic basis for computer programming. *Communications of ACM* **12** (10) 576–583.

Hrycej, T. (1988) Temporal Prolog. In: Kodratoff, Y. (ed.) *Proceedings of the European Conference on Artificial Intelligence (ECAI-88)*, Pitman Publishing 296–301.

Hrycej, T. (1993) A Temporal Extension of Prolog. *The Journal of Logic Programming* **15** 113–145.

Jones, C. B. (1981) *Development Mehods for Computer Programs including a Notion of Interference*, Ph. D. thesis, Oxford University.

Lamport, L. (1994) The temporal logic of actions. *ACM Transactions on Programming Languages and Systems* **16** (3) 872–923.

Ma, Q., Duan, Z. and Yang, X. (2010) Case Study of MSVL Axiom System. Technical Report available at `http://lcs.ios.ac.cn/~xzx/case.pdf`.

Manna, Z. and Pnueli, A. (1992) *Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag.

Misra, J. and Chandy, K. M. (1981) Proofs of Networks of Processes. *IEEE Transactions on Software Engineering* **7** 417–426.

Moszkowski, B. C. (1986) *Executing temporal logic programs*, Cambridge University Press.

Orgun, M. A. and Wadge, W. W. (1988) Chronolog: A Temporal Logic Programming Language and Its Formal Semantics, Department of Computer Science, University of Victoria, B.C., Canada.

Orgun, M. A. and Wadge, W. W. (1992) A Theory and Practice of Temporal Logic Programming. *Intensional Logics for Programming*, Oxford University Press 23–50.

Owicki, S. (1975) *Axiomatic Proof Techniques for Parallel Programs*, Ph. D. thesis, Department of Computer Science, Cornell University.

Owicki, S. and Gries, D. (1976) An Axiomatic Proof Technique for Parallel Programs I. *Acta Informatica* **6** 319–340.

Owre, S., Shankar, N., Rushby, J. M. and Stringer-Calvert, D. W. J. (1999) PVS Language Reference, Computer Science Laboratory, SRI International, Menlo Park, CA.

Pnueli, A. (1977) The temporal semantics of concurrent programs, Tel-Aviv University.

de Roever, W. P. (1985) The Quest for Compositionality. *Proceedings of IFIP Working Conference on the Role of Abstract Models in Computer Science*, North-Holland.

Rondogiannis, R., Gergatsoulis, M. and Panayiotopoulos, T. (1997) Cactus: A Branching Time Logic Programming Language. Proceedings of the 1st International Joint Conference Qualitative and Quantitative Practical Reasoning. *Springer-Verlag Lecture Notes in Artificial Intelligence* **1244** 511–524.

Rondogiannis, R., Gergatsoulis, M. and Panayiotopoulos, T. (1998) Branching Time Logic Programming: The Language Cactus and Its Application. *Computer Languages* **24** (3) 155–178.

Shankar, N., Owre, S., Rushby, J. M. and Stringer-Calvert, D. W. J. (1999) PVS Prover Guide, Computer Science Laboratory, SRI International, Menlo Park, CA.

Tang, C. S. (1983) Toward a unified logic basis for programming languages. *Proceedings of IFIP Congress 83*, Elsevier Science Publishers 425–429.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press

Tian, C. and Duan, Z. (2008) Propositional Projection Temporal Logic, Bchi Automata and omega-Regular Expressions. In: Proceedings TAMC2008. *Springer-Verlag Lecture Notes in Computer Science* **4978** 47–58.

Wadge, W. W. (1988) Tense Logic Programming. A Respectable Alternative. In: Proceedings of the 1988 International Symposium on Lucid and Intensional Programming, Sidney, B.C., Canada 26–32.

Wolper, P. L. (1983) Temporal logic can be more expressive. *Information and Control* **56** 72–99.

Xu, Q., de Roever, W. P. and He, J. (1997) The Rely-Gurarantee Method for Verifying Shared Variable Concurrent Programs. *Formal Aspects of Computing* **9** (2) 149–174.

Yang, X. and Duan, Z. (2008) Operational Semantics of Framed Tempura. *Journal of Logic and Algebraic Programming* **78** (1) 22–51.

https://doi.org/10.1017/S0960129510000241 Published online by Cambridge University Press