Presented at *COMPOS'97 - Compositionality: The Significant Difference*
Bad Malente, Germany, Sept. 8-12, 1997
W.-P. de Roever, H. Langmaack and A. Pnueli, editors
Lecture Notes in Computer Science, Vol. 1536, Springer-Verlag, pages 439–464

Compositional Reasoning using Interval Temporal Logic and Tempura

B. C. Moszkowski*

Department of Electrical and Electronic Engineering,
University of Newcastle upon Tyne, Newcastle NE1 7RU, Great Britain
email: Ben.Moszkowski@ncl.ac.uk

Abstract. We present a compositional methodology for specification and proof using Interval Temporal Logic (ITL). After given an introduction to ITL, we show how fixpoints of various ITL operators provide a flexible way to modularly reason about safety and liveness. In addition, some new techniques are described for compositionally transforming and refining ITL specifications. We also consider the use of ITL's programming language subset Tempura as a tool for testing the kinds of specifications dealt with here.

1 Introduction

Modularity is of great importance in computer science. Its desirability in formal methods is evidenced by the growing interest in *compositional* specification and proof techniques. Work by us over the last few years has shown that a powerful generalization of the increasing popular assumption/commitment approach to compositionality can be naturally embedded in *Interval Temporal Logic* (ITL) [12] through the use of temporal fixpoints [14]. Reasoning about safety, liveness and multiple time granularities are all feasible [15, 17].

In the present paper, we extend our methods to compositional transformation of specifications into other specifications. Basically, we show how to sequentially combine commitments containing specification fragments. The process continues until we have obtained the desired result. This is useful when verifying, say, the equivalence of two specifications. One sequentially transforms each specification into the other. The transformation techniques can also be applied to the refinement of relatively abstract specifications into more concrete programs.

We also show that various compositional ITL specification and proof techniques have executable variants. An interpreter for ITL's programming-language subset Tempura [13] serves as a prototype tool. Generally speaking, our approach represents theorems as Tempura programs annotated with temporal assertions over periods of time. This can be viewed as a generalization of the use of pre- and post-conditions as annotations for documenting and run-time checking of conventional sequential programs. Because our assertion language is an executable

* The research described here has been kindly supported by EPSRC research grant GR/K25922.

subset of ITL, we can specify and check for behavior over periods of time whereas conventional assertions are limited to single states.

The remaining sections of the paper are organized as follows. Section 2 gives a summary of ITL's syntax and semantics. In Sect. 3 we overview compositionality in ITL. Section 4 looks at compositional reasoning about liveness. Section 5 presents a compositional approach to transformation of specifications. Section 6 considers execution of compositional specifications. The appendix discusses a practical ITL axiom system for compositional proofs.

2 Review of Interval Temporal Logic

We now describe Interval Temporal Logic for finite time. The presentation is rather brief and the reader should refer to references such as [11, 3, 12, 14] for more details. Infinite intervals can also be handled by us but for simplicity we do not consider them until Subsect. 2.1. An ITL proof system is contained in the appendix.

ITL is a linear-time temporal logic with a discrete model of time. An interval σ in general has a length $|\sigma| \geq 0$ and a finite, nonempty sequence of $|\sigma| + 1$ states $\sigma_0, \ldots, \sigma_{|\sigma|}$. Thus the smallest intervals have length 0 and one state. Each state σ_i for $i \leq |\sigma|$ maps variables a, b, c, $\ldots$, A, B, C, $\ldots$ to data values. Lower-case variables a, b, c, $\ldots$ are called *static* and do not vary over time. Basic ITL contains conventional propositional operators such as $\wedge$ and first-order ones such as $\forall$ and $=$. Normally expressions and formulas are evaluated relative to the beginning of the interval. For example, the formula $J = I + 1$ is true on an interval σ iff the J's value in σ's initial state is one more that I's value in that state.

There are three primitive temporal operators *skip*, ";" (*chop*) and "*" (*chop-star*). Here is their syntax, assuming that S and T are themselves formulas:

$$\mathit{skip} \qquad S;T \qquad S^* \quad .$$

The formula *skip* has no operands and is true on an interval iff the interval has length 1 (i. e., exactly two states). Both *chop* and *chop-star* permit evaluation within various subintervals. A formula $S;T$ is true on an interval σ with states $\sigma_0, \ldots, \sigma_{|\sigma|}$ iff the interval can be chopped into two sequential parts sharing a single state σ_k for some $k \leq |\sigma|$ and in which the subformula S is true on the left part $\sigma_0, \ldots, \sigma_k$ and the subformula T is true on the right part $\sigma_k, \ldots, \sigma_{|\sigma|}$. For instance, the formula $\mathit{skip}; (J = I+1)$ is true on an interval σ iff σ has at least two states $\sigma_0, \sigma_1, \ldots$ and $J = I+1$ is true in the second one σ_1. A formula S^* is true on an interval iff the interval can be chopped into zero or more sequential parts and the subformula S is true on each. An empty interval (one having exactly one state) trivially satisfies any formula of the form S^* (including false^*). The following sometimes serves as an alternative syntax for S^*:

$$\mathit{chopstar}\, S \quad .$$

Figure 1 pictorially illustrates the semantics of *skip*, *chop*, and *chopstar*. Some simple ITL formulas together with intervals which satisfy them are shown in Fig. 2. Some further propositional operators definable in ITL are shown in Table 1.

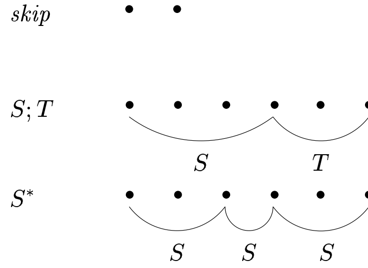


Fig. 1. Informal illustration of ITL semantics

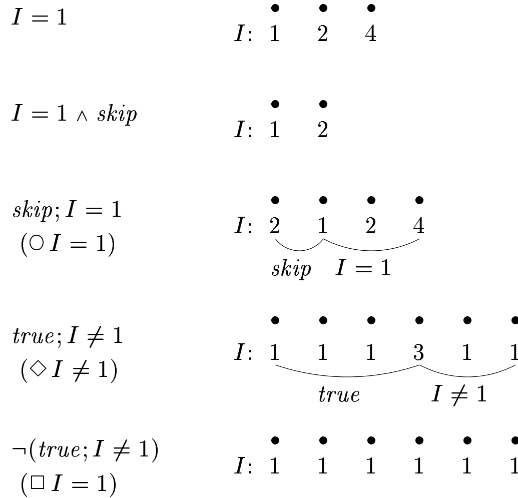


Fig. 2. Some sample ITL formulas and satisfying intervals

We generally use w, w', x, x' and so forth to denote *state formulas* with no temporal operators in them. Expressions are denoted by e, e' and so on.

In [14] we make use of the conventional logical notion of *definite descriptions* of the form $\imath v\colon S$ where v is a variable and S is a formula (see for example Kleene [7, pp. 167–171]). These allow a uniform semantic and axiomatic treatment in ITL of expressions such as $\bigcirc e$ (e's next value), *fin* e (e's final value)

Table 1. Some other definable propositional ITL operators

$\textcircled{w}\, S$	$\stackrel{\text{def}}{\equiv}$	$\neg \bigcirc \neg S$	Weak next
more	$\stackrel{\text{def}}{\equiv}$	$\bigcirc\, true$	Nonempty interval
empty	$\stackrel{\text{def}}{\equiv}$	$\neg more$	Empty interval
$\diamondsuit_i\, S$	$\stackrel{\text{def}}{\equiv}$	$S; true$	Some initial subinterval
$\boxdot_i\, S$	$\stackrel{\text{def}}{\equiv}$	$\neg \diamondsuit_i \neg S$	All initial subintervals
$\diamondsuit_a\, S$	$\stackrel{\text{def}}{\equiv}$	$true; S; true$	Some subinterval
$\boxdot_a\, S$	$\stackrel{\text{def}}{\equiv}$	$\neg \diamondsuit_a \neg S$	All subintervals
keep S	$\stackrel{\text{def}}{\equiv}$	$\boxdot_a(skip \supset S)$	All *unit* subintervals
fin S	$\stackrel{\text{def}}{\equiv}$	$\Box(empty \supset S)$	Final state
halt S	$\stackrel{\text{def}}{\equiv}$	$\Box(S \equiv empty)$	Exactly final state

and *len* (the interval's length). For example, $\bigcirc e$ can be defined as follows:

$$\bigcirc e \quad \stackrel{\text{def}}{=} \quad \imath a\colon \bigcirc (e = a) \ ,$$

where a does not occur freely in e. Here is a way to define temporal assignment using a *fin* term:

$$e \leftarrow e' \quad \stackrel{\text{def}}{\equiv} \quad (\mathit{fin}\ e) = e' \ .$$

The following operator *stable* tests whether an expression's value changes:

$$\mathit{stable}\ e \quad \stackrel{\text{def}}{\equiv} \quad \exists a\colon \Box(e = a) \ ,$$

where the static variable a is chosen so as not to occur freely in the expression e. The formula e *gets* e' is true iff in every unit subinterval, the initial value of the expression e' equals the final value of the expression e:

$$e\ \mathit{gets}\ e' \quad \stackrel{\text{def}}{\equiv} \quad \mathit{keep}\,(e \leftarrow e') \ .$$

An expression is said to be *padded* iff it is stable except for possibly the last state in the interval:

$$\mathit{padded}\ e \quad \stackrel{\text{def}}{\equiv} \quad \exists a\colon \mathit{keep}\,(e = a) \ ,$$

where the static variable a does not to occur freely in e. A useful version of assignment called *padded temporal assignment* can then be defined:

$$e \lessdot\!\sim e' \quad \stackrel{\text{def}}{\equiv} \quad (\mathit{fin}\ e) = e' \wedge \mathit{padded}\ e \ .$$

This ensures that e does not change until possibly the very end of the interval when the assignment takes effect. Figure 3 shows examples of these operators.

Formula						
stable K	K:	4	4	4	4	4
$K \leftarrow K+1$	K:	2	6	1	8	3
K *gets* $K+1$	K:	4	5	6	7	8
padded K	K:	3	3	3	3	1
$K \lessdot\!\sim K+1$	K:	2	2	2	2	3

Fig. 3. Sample formulas illustrating *stable*, etc.

2.1 ITL with Infinite Time

The semantics so far presented is suitable for reasoning about finite intervals. We now discuss some modifications needed to permit infinite intervals as well. First, we apply our semantics of $S;T$ and S^* to infinite intervals. As before, $S;T$ is true on an interval if the interval can be divided into one part for S and another adjacent part for T and that S^* is true if the interval can be divided into a finite number of parts, each satisfying S. In addition, we now also let $S;T$ be true on an infinite interval which satisfies S. For such an interval, we can ignore T. Furthermore, we let S^* be true on an infinite interval that is divisible into a finite number of subintervals where the last one has infinite length and each satisfies S or alternatively into an infinite number of finite intervals each satisfying S. We define new constructs for testing whether an interval is infinite or finite, and alter the definition of $\Diamond$:

$$\begin{array}{lcllcl} \mathit{inf} & \stackrel{\mathrm{def}}{\equiv} & \mathit{true};\mathit{false} & \quad \mathit{finite} & \stackrel{\mathrm{def}}{\equiv} & \neg\mathit{inf} \\ \Diamond S & \stackrel{\mathrm{def}}{\equiv} & \mathit{finite};S & \quad \mathit{sfin}\ S & \stackrel{\mathrm{def}}{\equiv} & \Diamond(\mathit{empty} \wedge S) \ . \end{array}$$

Here *sfin* S is a strong version of *fin* S and is true only on finite intervals. In contrast, *fin* S is vacuously true on all infinite intervals. The first-order operators for *temporal assignment* and *padded temporal assignment* are redefined to deal with both finite and infinite intervals:

$$\begin{array}{lcl} e \leftarrow e' & \stackrel{\mathrm{def}}{\equiv} & \mathit{finite} \supset (\mathit{fin}\ e) = e' \ , \\ e \lessdot\!\sim e' & \stackrel{\mathrm{def}}{\equiv} & \mathit{finite} \wedge (\mathit{fin}\ e) = e' \wedge \mathit{padded}\ e \ . \end{array}$$

Our experience seems to suggest that it is preferable to define $e \leftarrow e'$ to be vacuously true on infinite intervals and to define $e \lessdot\!\sim e'$ to be false on them.

3 Introduction to Compositionality in ITL

Modularity is a desirable attribute of any formal method. One of the best known modular logical notations is Hoare logic [4]. It uses the important insight that proofs about the pre/post-condition behavior of a sequential program can be decomposed into subproofs of the program's parts. In ITL we can express a Hoare clause as a theorem about discrete intervals of time consisting of one or more states:

$$\vdash \quad w \wedge \mathit{Sys} \supset \mathit{fin}\ w' \ .$$

Here w and w' are state formulas containing no temporal operators and Sys is some arbitrary temporal formula we wish to reason about. The temporal formula $\mathit{fin}\ w'$ is true on an interval iff w' is true in the interval's final state.

The pre/post-condition approach is not particularly well suited for specifying and verifying systems in which ongoing and parallel behavior are important. However, this can be remedied through the addition of what are commonly known as *assumptions* and *commitments*. Francez and Pnueli [2] are the first to consider them and refer to them as *interface predicates*. The following implication shows the basic form of an ITL theorem incorporating an assumption As and a commitment Co:

$$w \wedge As \wedge \mathit{Sys} \supset Co \wedge \mathit{fin}\ w' \ .$$

Table 2 briefly describes the role of each logical variable in such an implication. This can be seen as an embedding of Jones' *rely* and *guarantee conditions* [5] in ITL. In Fig. 4, we show a graphical representation of the implication called a *proof outline*.

Table 2. Compositional specification of system Sys

$$w \wedge As \wedge \mathit{Sys} \supset Co \wedge \mathit{fin}\ w' \quad ,$$

where:

w:	state formula about initial state,
As:	*assumption* about *overall* interval,
Sys:	the system under consideration,
Co:	*commitment* about *overall* interval,
w':	state formula about final state.

In general As and Co can be arbitrary temporal formulas. However, when compositional reasoning about sequential parts of a system is needed, it is useful to select assumptions and commitments for which the following derived ITL proof rule is sound:

$$\frac{\begin{array}{l}\vdash\ w \wedge As \wedge \mathit{Sys} \supset Co \wedge \mathit{fin}\ w' \ , \\ \vdash\ w' \wedge As \wedge \mathit{Sys}' \supset Co \wedge \mathit{fin}\ w''\end{array}}{\vdash\ w \wedge As \wedge (\mathit{Sys}; \mathit{Sys}') \supset Co \wedge \mathit{fin}\ w'' \ .} \tag{1}$$

$$As \left[\begin{array}{c} \{w\} \\ Sys \\ \{w'\} \end{array} \right] Co$$

Fig. 4. Proof outline for specification *Sys*

The rule uses the ITL operator *chop* to combine the formulas *Sys* and *Sys′* sequentially. An associated proof outline is shown in Fig. 5. Here is an analogous rule for decomposing a proof for zero or more iterations of a formula *Sys*:

$$\frac{\vdash\ w \wedge As \wedge Sys\ \supset\ Co \wedge \mathit{fin}\ w}{\vdash\ w \wedge As \wedge Sys^*\ \supset\ Co \wedge \mathit{fin}\ w\ .} \tag{2}$$

Figure 6 shows a corresponding proof outline. Similar rules are possible for *if*, *while* and other constructs.

$$As \left[\begin{array}{c} As \left[\begin{array}{c} \{w\} \\ Sys \\ \{w'\} \end{array} \right] Co \\ As \left[\begin{array}{c} Sys' \\ \{w''\} \end{array} \right] Co \end{array} \right] Co$$

Fig. 5. Proof outline for specification $Sys; Sys'$

$$As \left[\begin{array}{l} \{w\} \\ \mathit{chopstar}\ (\\ As \left[\begin{array}{c} \{w\} \\ Sys \\ \{w\} \end{array} \right] Co \\ ) \\ \{w\} \end{array} \right] Co$$

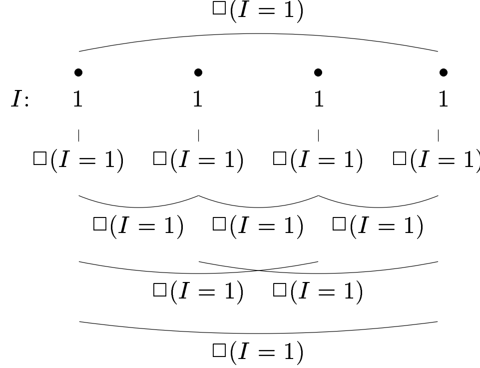
Fig. 6. Proof outline for specification Sys^*

To ensure soundness of proof rules 1 and 2, we require that *As* and *Co* be respective fixpoints of the ITL operators $\boxed{a}$ and *chop-star* as is now shown:

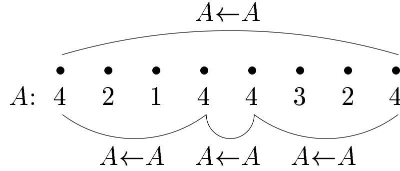
$$As \equiv \boxed{a}\, As\ , \qquad Co \equiv Co^*\ .$$

The first equivalence ensures that if the assumption *As* is true on an interval, it is also true in all subintervals. We say that such an assumption is *importable*.

The second equivalence ensures that if zero or more sequential instances of the commitment Co span an interval, Co is also true on the interval itself. A commitment with this property is said to be *exportable*. Importable assumptions and exportable commitments are collectively referred to as *sequentially compositional*. The temporal formula $\Box(I = 1)$ (read "*I always equals 1*") is a typical importable assumption. An example of its behavior can be pictorially represented as follows:



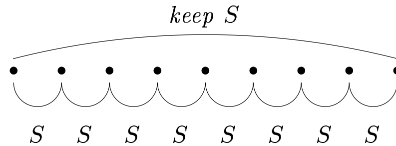
The set of importable assumptions turns out to consist exactly of those formulas expressible as $\boxed{a}\, S$ for some arbitrary subformula S. The temporal formula $A \leftarrow A$ ("*A's initial and final values on the interval are equal*") is an exportable commitment. Here is an interval illustrating this:



One can show that a formula is an exportable commitment if and only if it can be expressed in the form S^* for some arbitrary S. Some formulas such as *stable* K ("*K's value remains the same throughout the interval*") can be used both as assumptions and commitments. These are precisely the fixpoints of the ITL operator *keep* defined earlier in Table 1. We recall that formula *keep* S, for some subformula S, is defined to be true on an interval iff S is true on every unit subinterval (i.e., consisting of exactly two adjacent states):

$$\mathit{keep}\ S \stackrel{\mathrm{def}}{\equiv} \boxed{a}(\mathit{skip} \supset S) \ .$$

Here is a graphical representation of the semantics of a formula *keep* S on a typical interval:



The formula *keep* $(K \leftarrow K+1)$ is an example of such a fixpoint. It states that K increases by 1 between every pair of adjacent states. In Fig. 7 we show a proof outline for the following lemma:

$$\begin{array}{lll} \vdash & J=1 \wedge \mathit{stable}\ J \wedge (\mathit{stable}\ K; K \mathrel{<\!\!\sim} K+J) & \\ & \supset \quad \mathit{keep}\,(K \leq \bigcirc K \leq K+1) \wedge \mathit{fin}\,(J=1) \ . & \end{array} \tag{3}$$

$$\mathit{stable}\ J \left[\begin{array}{l} \mathit{stable}\ J \left[\begin{array}{l} \{J=1\} \\ \mathit{stable}\ K \\ \{J=1\} \end{array} \right] Co \\ \mathit{stable}\ J \left[\begin{array}{l} \{J=1\} \\ K \mathrel{<\!\!\sim} K+J \\ \{J=1\} \end{array} \right] Co \end{array} \right] Co$$

where Co is *keep* $(K \leq \bigcirc K \leq K+1)$.

Fig. 7. Proof outline for lemma (3).

Note that our approach only requires that assumptions and commitments which are used directly in rules such (1) and (2) are sequentially compositional. Compositional proofs about a system in ITL typically also involve reasoning about assumptions and commitments which are not sequentially compositional. For instance, there is an important class of formulas using the standard temporal operator $\Box$. In general they can neither be used directly as sequentially compositional assumptions or commitments. Nevertheless, those of the form $\Box\, w$, for some state formula w, can be used as importable assumptions since they are fixpoints of the operator $\boxed{\scriptstyle a}$:

$$\vdash \quad \Box\, w \equiv \boxed{\scriptstyle a}\,\Box\, w \ .$$

Unfortunately, even these cannot be used as exportable commitments since, for example, the formula $(\Box\, w)^*$ (and indeed any formula S^*) is vacuously true on intervals having exactly one state whereas $\Box\, w$ is not necessarily true on them. In other words $(\Box\, w)^* \wedge \neg \Box\, w$ is satisfiable for some w and therefore $\Box\, w \equiv (\Box\, w)^*$ is normally not a theorem. However, there are simple ways around this. For instance, we can express $\Box\, w$ as the conjunction of *keep* w and *fin* w:

$$\vdash \quad \Box\, w \equiv \mathit{keep}\ w \wedge \mathit{fin}\ w \ .$$

Since w is a state formula, *keep* w turns out to be true on an interval iff w is true on all of the interval's states except possibly the last one. Since we already mentioned that *keep* S for *any* formula S is a perfectly good exportable commitment, we can use *keep* w in compositional proofs and at the very end combine it with *fin* w to obtain the desired (generally nonexportable) commitment $\Box\, w$.

4 Compositional Analysis of Liveness

The techniques so far presented do not address reasoning about formulas involving liveness such as $\Box \Diamond x$ and $\Box(x \supset \Diamond x')$, where x and x' are state formulas. We now briefly discuss how to handle such temporal formulas in compositional proofs. More details and examples of proofs can be found in [17]. Let us now use the temporal operator $\boxed{\mathrm{m}}$ ("*box-m*" or "*mostly*") defined as follows:

$$\boxed{\mathrm{m}}\, S \stackrel{\text{def}}{\equiv} \Box(\mathit{more} \supset S) \ .$$

A formula $\boxed{\mathrm{m}}\, S$ is true on an interval iff the subformula S is true on all terminal (suffix) subintervals with *more* than one state, that is all the interval's *nonempty* terminal subintervals. Therefore $\boxed{\mathrm{m}}$ ignores the last (empty) terminal subinterval consisting of one state and is slightly weaker than $\Box$. In Fig. 8 we illustrate the difference between the two operators. On infinite intervals, their behavior is identical.

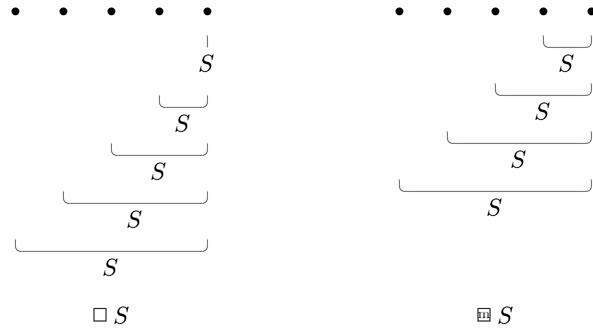


Fig. 8. Comparison of $\Box\, S$ with $\boxed{\mathrm{m}}\, S$

It turns out that for any state formulas w and w' and an arbitrary formula S, the formula $\boxed{\mathrm{m}}(w \supset S; w')$ is a fixpoint of *chop-star*:

$$\vdash \quad \boxed{\mathrm{m}}(w \supset S; w') \equiv \big(\boxed{\mathrm{m}}(w \supset S; w')\big)^* \ .$$

This is because $\boxed{\mathrm{m}}(w \supset S; w')$ can be expressed as $\boxed{\mathrm{m}}\, \Diamond_{\mathrm{i}}\big(w \supset (S \wedge \mathit{fin}\ w')\big)$ and any formula of the form $\boxed{\mathrm{m}}\, \Diamond_{\mathrm{i}}\, T$ for some arbitrary formula T is a fixpoint of *chop-star*.

For state formulas x and x', the implication $x \supset \Diamond x'$ can be expressed as $x \supset \mathit{finite}; x'$. Consequently, the formula $\boxed{\mathrm{m}}(x \supset \Diamond x')$ is a fixpoint of *chop-star*. Table 3 gives examples of exportable commitments expressible in the form $\boxed{\mathrm{m}}(x \supset S; x')$ for suitable x, x' and S.

One way to prove a formula $\Box(x \supset \Diamond x')$, is by establishing the related formula $\boxed{\mathrm{m}}(x \supset \Diamond x')$ through sequential composition and also showing the formula

Table 3. Examples of formulas expressible as $\Box_m(x \supset S; x')$

$\Box_m\, x$	
$\Box_m \Diamond x$	
$\Box_m(x \;\supset\; \Diamond x')$	
$\Box_m \Diamond_i(\mathit{skip} \wedge S)$	(same as *keep* S)

$\mathit{fin}\,(x \supset \Diamond x')$. We then use the following lemma relating $\Box$ with $\Box_m$ and fin:

$$\vdash \quad \Box S \equiv \Box_m S \wedge \mathit{fin}\ S \ .$$

The fixpoints of the ITL operator $\Diamond_a$ (read "*diamond-a*") are important when we reason about liveness. In general, $\Diamond_a S$ is true on an interval iff S is true on some subinterval (possibly the interval itself). Formulas such as $\Diamond x$, where x is a state formula, and $\neg\, \mathit{stable}\ A$ (meaning "*The variable A has more than one value over the interval*") are fixpoints of $\Diamond_a$. If DA is a fixpoint of $\Diamond_a$, $\Box_m DA$ is a fixpoint of *chop-star* and hence an exportable commitment. More generally, for any state formula x and $\Diamond_a$-fixpoint DA, a formula of the form $\Box_m(x \supset DA)$ is always a fixpoint of *chop-star*. This is because $\Box_m(x \supset DA)$ can be expressed as $\Box_m(x \supset DA; \mathit{true})$. The fixpoints of $\Diamond_a$ are closed under disjunction.

Let us consider another benefit of fixpoints of $\Diamond_a$. Suppose one wishes to prove that a formula $\mathit{Sys}; \mathit{Sys}'$ with a suitable precondition and an importable assumption implies a commitment $\Box_m(x \supset DA)$ for some state formula x and some fixpoint DA of $\Diamond_a$. The most straightforward thing to do is to first show the commitment both for Sys and Sys' and then combine the results using proof rule (1). However, this is not always possible since DA might never be true in Sys and only occur in Sys' even though x is perhaps somewhere true in Sys. In such cases, we can use the following derivable proof rule for all intervals, both finite and infinite:

$$\frac{\begin{array}{l}\vdash\ w \wedge As \wedge \mathit{Sys} \ \supset\ \mathit{finite} \wedge \mathit{fin}\ w' \ ,\\ \vdash\ w' \wedge As \wedge \mathit{Sys}' \ \supset\ \Box_m(x \supset DA) \wedge DA \wedge \mathit{fin}\ w''\end{array}}{\vdash\ w \wedge As \wedge (\mathit{Sys}; \mathit{Sys}') \ \supset\ \Box_m(x \supset DA) \wedge DA \wedge \mathit{fin}\ w''} \ . \tag{4}$$

This shows that the only thing we need to verify about Sys is that it terminates with the formula w' true in its final state. Both the desired commitment $\Box_m(x \supset DA)$ and DA itself can be obtained for $\mathit{Sys}; \mathit{Sys}'$ from Sys' alone because DA is a fixpoint of $\Diamond_a$. A proof outline for this is given in Fig. 9.

Figure 10 shows a proof outline for the following lemma in which the variable K is never stable, except trivially in the last state (if the overall interval is finite):

$$\begin{array}{ll}\vdash & J \geq 1 \wedge \mathit{keep}\,(J \leq \bigcirc J) \wedge \big((\mathit{stable}\ K \wedge \mathit{finite}); K \lessdot\!\!\sim K + J\big)\\ & \quad \supset \quad \Box_m \neg\, \mathit{stable}\ K \wedge \neg\, \mathit{stable}\ K \wedge \mathit{fin}\,(J \geq 1) \ .\end{array} \tag{5}$$

$\{w\}$
As As Sys *finite*
$\{w'\}$
As Sys' $\boxdot(x \supset DA) \wedge DA$ $\boxdot(x \supset DA) \wedge DA$
$\{w''\}$

Fig. 9. A proof outline for rule (4)

$\{J \geq 1\}$
$keep\ (J \leq \bigcirc J)$ $stable\ K \wedge finite$ *finite*
$keep\ (J \leq \bigcirc J)$ $\{J \geq 1\}$ $\boxdot \neg\, stable\ K$
$keep\ (J \leq \bigcirc J)$ $K \lessdot\!\!\sim K + J$ $\boxdot \neg\, stable\ K$ $\wedge \neg\, stable\ K$
$\{J \geq 1\}$ $\wedge \neg\, stable\ K$

Fig. 10. Proof outline for lemma (5).

Sometimes a more powerful technique for analyzing reachability is needed. We originally introduced the notion of *markers* in [11, p. 127]. A marker is a boolean state variable, called here Mk, which is true exactly at the start and end of loop iterations. For example, a variant of *chop-star* having a marker can be defined as follows:

$$chopstar_{Mk}\, S \quad \stackrel{\text{def}}{\equiv} \quad (S \wedge \bigcirc halt\ Mk)^* \ .$$

Without loss of generality, we can always existentially introduce a marker as an auxiliary variable. The following provable lemma states this:

$$\vdash \quad S^* \equiv \exists Mk\colon (Mk \wedge chopstar_{Mk}\, S) \ ,$$

where Mk does not occur freely in the formula S. The use of markers in liveness proofs is discussed in more detail in [17].

5 Compositional Transformation of Specifications

Assumptions and commitments are usually thought of as being simpler than the systems they describe. However in ITL it is possible to embed arbitrary formulas in them. This provides a framework for compositional transformation and refinement of specifications. For example, we can specify that one system Sys implies that whenever some state formula x is true, the behavior of another system Sys' is observed followed by another state formula x' being true:

$$w \wedge As \wedge Sys \quad \supset \quad \Box(x \ \supset\ Sys'; x') \wedge fin\ w' \ .$$

The use of formulas of the form $\Box(x \ \supset\ S; x')$ provides a powerful means for decomposition. For example, suppose we wish to establish the following commitment which embeds $S; S'$:

$$\Box(x \ \supset\ S; S'; x') \ .$$

This can be split into two smaller commitments for S and S' using the general ITL theorem shown below:

$$\vdash \quad \Box(x \supset S; y) \wedge \Box(y \supset S'; x') \quad \supset \quad \Box(x \supset S; S'; x') \ . \tag{6}$$

Here we introduce a new state formula y to connect the two individual commitments. A similar decomposition theorem can be used for while-loops which are themselves expressible in ITL as follows:

$$\mathit{while}\, w\ \mathit{do}\ S \ \stackrel{\text{def}}{\equiv}\ (w \wedge S)^* \wedge \mathit{fin}\ \neg w \ .$$

A commitment with an embedded while-loop has the following form:

$$\Box\big(x \supset (\mathit{while}\, w\ \mathit{do}\ S); x'\big) \ .$$

It can be broken down using the theorem now given:

$$\begin{array}{ll} \vdash & \Box\big(x \wedge w \supset (S \wedge \mathit{more}); x\big) \wedge \Box\big(x \wedge \neg w \supset x'\big) \\ & \quad \supset \quad \Box\big(x \supset (\mathit{while}\, w\ \mathit{do}\ \mathit{Sys}); x'\big) \ . \end{array}$$

The formula x serves as the while-loop's invariant. Here is a corollary of this for introducing a while-loop itself:

$$\begin{array}{ll} \vdash & \Box\big(x \wedge w \supset (S \wedge \mathit{more}); x\big) \wedge \Box\big(x \wedge \neg w \supset \mathit{empty}\big) \\ & \quad \supset \quad x \supset (\mathit{while}\, w\ \mathit{do}\ \mathit{Sys}) \wedge \mathit{fin}\,(x \wedge \neg w) \ . \end{array} \tag{7}$$

Sometimes, we wish to prove that one system implies another:

$$w \wedge \mathit{As} \wedge \mathit{Sys} \quad \supset \quad \mathit{Sys}' \wedge \mathit{fin}\ w' \ .$$

This can be thought of as stating the existence of a transformation from Sys to Sys'. If we have already compositionally demonstrated a commitment $\Box(x \supset S; x')$, we can obtain S from it through the next theorem:

$$\vdash \quad x \wedge \Box(x \supset S; x') \wedge \Box(x' \supset \mathit{empty}) \quad \supset \quad S \ .$$

A commitment expressed as $\Box(x \supset S; x')$ is in general not exportable. However, we noted in Sect. 4 that a formula such as $\boxed{\mathrm{m}}(x \supset S; x')$ when used as a commitment is exportable since it is always a fixpoint of *chop-star*. This greatly facilitates modular proofs since we obtain the benefits of sequential compositionality. The following lemmas assist in moving between the two types of commitments:

$$\begin{array}{ll} \vdash & \Box(x \supset S; x') \quad \supset \quad \boxed{\mathrm{m}}(x \supset S; x') \\ \vdash & \boxed{\mathrm{m}}(x \supset S; x') \wedge \mathit{fin}\ \neg x \quad \supset \quad \Box(x \supset S; x') \ . \end{array}$$

The subformula $\mathit{fin}\ \neg x$ in the second lemma ensures that the implication $x \supset S; x'$ is trivially true in the interval's final state if the interval is finite.

5.1 An Example

Figure 11 shows two logically equivalent specifications $p1(K, n)$ and $p2(K, n)$ which monotonically increase a variable K until it equals $2n$. Here is the behavior of K and n in a sample interval having 12 states:

	•	•	•	•	•	•	•	•	•	•	•	•
n:	3	3	3	3	3	3	3	3	3	3	3	3
K:	0	0	1	2	2	2	3	4	5	5	5	6

$$p1(K, n): \quad \textit{while } K \neq 2n \textit{ do } (K \lessapprox K + 1)$$

$$\begin{aligned} p2(K, n): \quad & \textit{halt } (K = 2n) \\ & \wedge \big(K \lessapprox K + 1; \textit{halt } \textit{even}(K)\big)^* \\ & \wedge \big(\textit{halt } \textit{odd}(K); K \lessapprox K + 1\big)^* \end{aligned}$$

Fig. 11. Two equivalent specifications

We will consider how to establish equivalence when K is initially even. This can be reduced to proving the following two implications:

$$\vdash \quad \textit{even}(K) \wedge p1(K, n) \quad \supset \quad p2(K, n) \tag{8}$$

$$\vdash \quad \textit{even}(K) \wedge p2(K, n) \quad \supset \quad p1(K, n) \ . \tag{9}$$

Each of these is analyzed individually.

Proof of $\boldsymbol{even(K) \wedge p1(K, n) \ \supset \ p2(K, n)}$. In order to prove lemma (8), we give names to $p2$'s conjuncts as shown in Table 4 and prove the following lemmas which demonstrate that $p1$ implies each of them:

$$\vdash \quad \textit{even}(K) \wedge p1(K, n) \quad \supset \quad p2a(K, n) \tag{10}$$

$$\vdash \quad \textit{even}(K) \wedge p1(K, n) \quad \supset \quad p2b(K) \tag{11}$$

$$\vdash \quad \textit{even}(K) \wedge p1(K, n) \quad \supset \quad p2c(K) \ . \tag{12}$$

The simplest of the three lemmas is the first one (10). A proof outline is shown in Fig. 12. It uses the following equivalence for the *halt* construct:

$$\vdash \quad \textit{halt } w \quad \equiv \quad \boxed{\scriptstyle m} \neg w \wedge \textit{fin } w \ . \tag{13}$$

The proofs of lemma (11) for $p2b$ and lemma (12) for $p2c$ are similar to each other so we only look at the one for $p2b$. The lemma's proof uses an auxiliary boolean state variable X which is acts as follows:

$$X \wedge X \textit{ gets } (K \neq \bigcirc K) \ .$$

Table 4. Decomposition of specification $p2(K,n)$

$$
\begin{array}{ll}
p2(K,n)\colon & p2a(K,n) \wedge p2b(K) \wedge p2c(K) \\[6pt]
p2a(K,n)\colon & halt\ (K = 2n) \\
p2b(K)\colon & \big(K \lessdot\!\sim K+1;\, halt\ \ even(K)\big)^{*} \\
p2c(K)\colon & \big(halt\ \ odd(K);\, K \lessdot\!\sim K+1\big)^{*}
\end{array}
$$

$$
\left.\left.\left.\begin{array}{l}
\{true\} \\
while\ K \neq 2n\ do\ (\\
\quad \left.\begin{array}{l}\{K \neq 2n\} \\ K \lessdot\!\sim K+1 \\ \{true\}\end{array}\right]\ \boxdot\, K \neq 2n \\
) \\
\{K = 2n\}
\end{array}\right]\ \boxdot\, K \neq 2n\right]\ \begin{array}{l}\boxdot\, K \neq 2n \\ \wedge\ fin\ (K = 2n)\end{array}\right]\ halt\ (K = 2n)
$$

Fig. 12. Proof outline for lemma (10).

It is initially true and subsequently is true exactly whenever K's value changes. We can introduce X without loss of generality using existential quantification. Here is the behavior of K and X in the interval described earlier:

	•	•	•	•	•	•	•	•	•	•	•	•
K:	0	0	1	2	2	2	3	4	5	5	5	6
X:	*true*	*false*	*true*	*true*	*false*	*false*	*true*	*true*	*true*	*false*	*false*	*true*

The subformula $X\ gets\ (K \neq \bigcirc K)$ is used as an importable assumption in the proofs for *p2b*.

The outermost operator used in *p2b* is *chop-star*. The following general theorem provides a way to introduce a formula S^* from a commitment which embeds S in it:

$$x \wedge \boxdot\big(x \supset (S \wedge more); x\big) \quad \supset \quad S^* \ .$$

In the case of *p2b*, we use $X \wedge even(K)$ as an instance of x and take the following as an instance of $\boxdot\big(x \supset (S \wedge more); x\big)$:

$$
\begin{array}{l}
\boxdot\Big(\, X \wedge even(K) \ \supset \\
\qquad \big((K \lessdot\!\sim K+1;\, halt\ \ even(K)) \wedge more\big); \\
\qquad \big(X \wedge even(K)\big)\Big) \ .
\end{array}
$$

This can be further split into two commitments using a variant of lemma (6) given below for sequentially decomposing loop bodies:

$$
\begin{array}{ll}
\vdash & \boxdot\big(x \supset (S \wedge more); x'\big) \wedge \Box(x' \supset S'; x'') \\
& \quad \supset \quad \boxdot\big(x \supset ((S; S') \wedge more); x''\big) \ .
\end{array}
$$

One commitment is for $K <\!\!\sim K+1 \wedge \mathit{more}$ and the other is for $\mathit{halt}\ \mathit{even}(K)$:

$$\boxed{m}\Big(X \wedge \mathit{even}(K) \supset (K <\!\!\sim K+1 \wedge \mathit{more}); \big(X \wedge \mathit{odd}(K)\big)\Big) \qquad \Box\Big(X \wedge \mathit{odd}(K) \supset \mathit{halt}\ \mathit{even}(K); \big(X \wedge \mathit{even}(K)\big)\Big) \ .$$

The associated lemmas are now given:

$$\vdash \quad \mathit{even}(K) \wedge X \wedge X \ \mathit{gets}\ (K \neq \bigcirc K) \wedge \mathit{P1}(K,n) \supset \boxed{m}\Big(X \wedge \mathit{even}(K) \supset (K <\!\!\sim K+1 \wedge \mathit{more}); \big(X \wedge \mathit{odd}(K)\big)\Big) \tag{14}$$

$$\vdash \quad \mathit{even}(K) \wedge X \wedge X \ \mathit{gets}\ (K \neq \bigcirc K) \wedge \mathit{P1}(K,n) \supset \Box\Big(X \wedge \mathit{odd}(K) \supset \mathit{halt}\ \mathit{even}(K); \big(X \wedge \mathit{even}(K)\big)\Big) \ . \tag{15}$$

Proof outlines for these are shown in Figs. 13 and 14, respectively. Figure 15 summarizes the overall proof of lemma (11).

$$\mathit{As}\left[\begin{array}{l} \{X\} \\ \mathit{while}\ K \neq 2n\ \mathit{do}\ (\\ \quad \mathit{As}\left[\begin{array}{l} \{X \wedge K \neq 2n\} \\ K <\!\!\sim K+1 \\ \{X\} \end{array}\right] \begin{array}{l} \mathit{even}(K) \supset \\ \quad (K <\!\!\sim K+1 \wedge \mathit{more}); \\ \quad \big(X \wedge \mathit{odd}(K)\big) \\ \wedge\ \textcircled{w}\,\boxed{m}\,\neg X \end{array} \ \mathit{Co} \\ ) \\ \{X\} \end{array}\right] \mathit{Co}$$

where As is $X\ \mathit{gets}\ (K \neq \bigcirc K)$

and Co is $\boxed{m}\Big(X \wedge \mathit{even}(K) \supset (K <\!\!\sim K+1 \wedge \mathit{more}); \big(X \wedge \mathit{odd}(K)\big)\Big)$.

Fig. 13. Proof outline for lemma (14).

Proof of $\boldsymbol{\mathit{even}(K) \wedge \mathit{p2}(K,n) \supset \mathit{p1}(K,n)}$. In order to obtain $\mathit{p1}(K,n)$ from $\mathit{p2}(K,n)$, we first use the fact that $\mathit{p1}(K,n)$ is expressed as a while-loop and can therefore be decomposed using the lemma now given which is provable from corollary (7):

$$\vdash \quad \mathit{halt}\ \neg w \wedge S^{*} \quad \supset \quad \mathit{while}\ w\ \mathit{do}\ S \ . \tag{16}$$

$$
As\left[\begin{array}{l}
\{X\} \\
\mathit{while}\ K \neq 2n\ \mathit{do}\ (\\
\quad As\left[\begin{array}{l}
\{X \wedge K \neq 2n\} \\
K \lesssim K+1 \\
\{X\}
\end{array}\right]
\begin{array}{l}
odd(K) \supset \\
\quad \left.\begin{array}{l} halt\ \ even(K); \\ \big(X \wedge even(K)\big) \end{array}\right] Co \\
\wedge\, \textcircled{w}\, \boxed{m}\, \neg X
\end{array} \\
) \\
\{X \wedge K = 2n\}
\end{array}\right] Co
\ \ \begin{array}{l} Co \\ \wedge\ \mathit{fin}\ \neg odd(K) \end{array}\right] Co'
$$

$$
\begin{array}{l}
\text{where } As \text{ is } X\ gets\ (K \neq \bigcirc K)\quad , \\[1ex]
Co \text{ is } \boxed{m}\Big(X \wedge odd(K) \supset \\
\qquad\qquad halt\ \ even(K); \\
\qquad\qquad \big(X \wedge even(K)\big)\Big) \\[1ex]
\text{and } Co' \text{ is } \Box\Big(X \wedge odd(K) \supset \\
\qquad\qquad\qquad halt\ \ even(K); \\
\qquad\qquad\qquad \big(X \wedge even(K)\big)\Big)\quad .
\end{array}
$$

Fig. 14. Proof outline for lemma (15).

$$
even(K) \wedge X \wedge X\ gets\ (K \neq \bigcirc K) \wedge p1(K, n)
$$

a: $\supset$

$$
\begin{array}{l}
\boxed{m}\big(X \wedge even(K) \supset \\
\quad (K \lesssim K+1 \wedge more); \\
\quad (X \wedge odd(K))\big)
\end{array}
\quad \bigwedge \quad
\begin{array}{l}
\Box\big(X \wedge odd(K) \supset \\
\quad halt\ \ even(K); \\
\quad (X \wedge even(K))\big)
\end{array}
$$

b: $\supset$

$$
\begin{array}{l}
\boxed{m}\Big(X \wedge even(K) \supset \\
\quad \big((K \lesssim K+1; halt\ \ even(K)) \wedge more\big); \\
\quad \big(X \wedge even(K)\big)\Big)
\end{array}
$$

c: $\supset$

$$
\begin{array}{l}
X \wedge even(K) \quad \supset \\
\quad \big(K \lesssim K+1; halt\ \ even(K)\big)^*
\end{array}
$$

Fig. 15. Overview of proof of lemma (8)

Here is the particular instance of this that we need to show:

$$\vdash \quad \mathit{halt}\ \neg(K \neq 2n) \wedge (K \leftsquigarrow K+1)^* \quad \supset \quad \mathit{while}\ K \neq 2n\ \mathit{do}\ (K \leftsquigarrow K+1)\ .$$

The antecedent of this can be obtained from $p2(K, n)$ in the following manner:

$$\vdash \quad \mathit{even}(K) \wedge p2(K,n) \ \supset\ \mathit{halt}\ \neg(K \neq 2n) \wedge (K \leftsquigarrow K+1)^*\ . \tag{17}$$

The main work in proving lemma (17) involves obtaining $(K \leftsquigarrow K+1)^*$:

$$\vdash \quad \mathit{even}(K) \wedge p2(K,n) \ \supset\ (K \leftsquigarrow K+1)^*\ . \tag{18}$$

As done previously, we use an auxiliary variable X which is initially true and subsequently is true exactly whenever K's value changes. Figure 16 summarizes the overall proof of lemma (18).

$$\begin{array}{ccc}
\begin{array}{c}\mathit{even}(K) \wedge X \wedge X\ \mathit{gets}\ (K \neq \bigcirc K) \\ \wedge\ p2b(K)\end{array} & & \begin{array}{c}\mathit{even}(K) \wedge X \wedge X\ \mathit{gets}\ (K \neq \bigcirc K) \\ \wedge\ p2c(K)\end{array} \\
\supset & & \supset \\
\boxed{m}\big(X \wedge \mathit{even}(K) \ \supset\ (K \leftsquigarrow K+1 \wedge \mathit{more}); X\big) & \wedge & \boxed{m}\big(X \wedge \mathit{odd}(K) \ \supset\ (K \leftsquigarrow K+1 \wedge \mathit{more}); X\big) \\
 & \supset & \\
 & \boxed{m}\big(X \ \supset\ (K \leftsquigarrow K+1 \wedge \mathit{more}); X\big) & \\
 & \supset & \\
 & X \ \supset\ (K \leftsquigarrow K+1)^* &
\end{array}$$

Fig. 16. Overview of proof of lemma (18)

A proof outline for the following lemma about $p2b(K)$ is shown in Fig. 17.

$$\begin{array}{ll}\vdash \quad & \mathit{even}(K) \wedge X \wedge X\ \mathit{gets}\ (K \neq \bigcirc K) \wedge p2b(K) \\ & \quad \supset \quad \boxed{m}\big(X \wedge \mathit{even}(K) \ \supset\ (K \leftsquigarrow K+1 \wedge \mathit{more}); X\big)\ .\end{array} \tag{19}$$

6 Executable Compositional Specifications

The Tempura programming language [13] is based on an *executable* subset of ITL. With some care, many interesting ITL specifications can be directly run by a Tempura interpreter. This consequently provides a valuable tool for “hands-on” access to ITL. It appears to be worthwhile to explore ways of exploiting Tempura

$$
As\left[\begin{array}{l}
\{X \wedge even(K)\} \\
chopstar\ (\\
\quad As\left[\begin{array}{l}
As\left[\begin{array}{l}\{X \wedge even(K)\} \\ K \lessapprox K+1 \\ \{odd(K)\}\end{array}\right]
\begin{array}{l} even(K)\ \supset \\ \quad (K \lessapprox K+1 \wedge more); X \\ \wedge\ ⓦ \boxed{\scriptstyle m}\, \neg X \end{array} \Bigg]\ Co \\
As\left[\begin{array}{l} halt\ even(K) \\ \{X \wedge even(K)\}\end{array}\right] \boxed{\scriptstyle m}\, odd(K) \Big]\ Co
\end{array}\right] Co \\
) \\
\{X \wedge even(K)\}
\end{array}\right] Co
$$

where As is $X\ gets\ (K \neq \bigcirc K)$

and Co is $\boxed{\scriptstyle m}\big(X \wedge even(K)\ \supset\ (K \lessapprox K+1 \wedge more); X\big)$.

Fig. 17. Proof outline for lemma (19).

for testing executable specifications which have assumptions and commitments in them. At present we are experimenting with various Tempura programming styles and interpreter implementation techniques to improve facilities for carrying this out. For instance, consider the following simple Tempura conjunction:

$$K = 0 \wedge p1(K,3) \wedge p2(K,3) \wedge \Box\, output(K)\ . \tag{20}$$

This initializes the variable K to 0 and runs $p1(K,3)$ and $p2(K,3)$ in parallel on a state-by-state basis. Furthermore, the value of K in each state is displayed. Figure 18 shows a typical run. The construct $K \lessapprox K+1$ is nondeterministic since it does not specifying any particular interval length. The interpreter therefore generates a pseudo-random value in some user-adjustable range. We note that the Tempura source code for *p1* and *p2* is annotated with many of assumptions and commitments described earlier in various proofs in Subsect. 5.1. Therefore the run shown in Fig. 18 also extensively checks them.

Now consider the following general ITL formula containing an assumption and a commitment:

$$w \wedge As \wedge Sys\ \supset\ Co \wedge fin\ w'\ .$$

This can sometimes be *tested* in Tempura for inconsistencies using the following conjunction:

$$w \wedge As \wedge Sys \wedge Co \wedge fin\ w'\ .$$

Of course, it is not feasible to attempt to execute arbitrary assumptions and commitments. Here are two reasons why:

- They can contain arbitrary undecidable first-order ITL subformulas.
- Satisfiability can be nonelementary even for decidable propositional ITL formulas (Kozen in [11, p. 24]).

However, there are interesting and useful classes. For example, Table 5 shows various importable assumptions which can be tested. Similarly, Table 6 contains

```
Tempura 4> run (K=0 and p1(K,3) and p2(K,3) and always output(K)).
State   0: K=0
State   1: K=0
State   2: K=0
State   3: K=1
State   4: K=1
State   5: K=2
State   6: K=2
State   7: K=2
State   8: K=3
State   9: K=3
State  10: K=4
State  11: K=5
State  12: K=5
State  13: K=5
State  14: K=6

Done!  Computation length:  14.  Total Passes:  40.
```

Fig. 18. Sample Tempura output for formula (20)

a number of checkable exportable commitments. Indeed, we discovered that formulas having the form $\boxed{\scriptstyle m}(w \supset S; w')$ were suitable as exportable commitments only after we tried to prove compositionally the equivalence of some experimental Tempura specifications. Many assumptions and commitments which are not sequentially compositional can also be handled by Tempura. Examples include commitments of the form $\Box(w \supset S; w')$ as long as w, S and w' are themselves executable. We are even investigating ways of implementing negation of suitable Tempura programs. This would permit empirical testing of the validity of an implication of the form $\mathit{Sys} \supset \mathit{Sys}'$ by examining satisfiability of a program such as $\mathit{Sys} \wedge \neg \mathit{Sys}'$.

Table 5. Some executable importable assumptions

$\mathit{stable}\ A$	A's value remains stable
$\mathit{keep}\,(K \le \bigcirc K \le K+1)$	K's value weakly increases monotonically
$\Box(K = 0)$	K always equals 0
$\Box\Diamond(K = 1 \vee \mathit{empty})$	Always eventually either K equals 1 or the interval terminates

Table 6. Some executable exportable commitments

$\mathit{stable}\ A$	A's value remains stable
$\mathit{keep}\,(K \le \bigcirc K \le K+1)$	K's value weakly increases monotonically
$\boxed{\mathrm{m}}(K=0)$	K's value is *mostly* zero
$\boxed{\mathrm{m}}\Diamond K=1$	K's value is *mostly* sometimes 1
$\boxed{\mathrm{m}}(K=j \supset \Diamond K=j+1)$	*Mostly* when $K=j$, eventually $K=j+1$
$\boxed{\mathrm{m}}\exists i\colon \big(i=A \wedge \Diamond(A \neq i)\big)$	A is *mostly* not stable
$\boxed{\mathrm{m}}(w \supset S; w')$	*Mostly* w implies S then w'

Let us now enumerate some benefits of using Tempura for testing compositional specifications:

- Tempura offers a "learning-by-doing" approach to ITL.
- Larger ITL specifications can be developed and tested than with pencil and paper alone.
- Modular, reusable Tempura test suites can be developed.
- Several specifications can be compared over a range of test data.
- The use of specialized theorem provers and model checkers can be postponed until after a preliminary run-time consistency check of candidate specifications and proofs.
- In contrast to model checking, execution can be used to test theorems which are not decidable.
- ITL and Tempura both improve through the increased feedback between theory and practice. Particular benefits are:
  - The discovery of further executable assumptions and commitments.
  - The development of more and better compositional proof techniques.
- Interval Temporal Logic serves as the single unifying formalization at all stages of analysis.

Of course, we do not realistically expect the use of an interpreter to replace theorem provers and model checkers. However, this does seem to be an intriguing alternative suitable in various circumstances. For example, we already mentioned that the compositional equivalence proofs discussed in this paper have been partially checked using a Tempura interpreter. We have also been able to do a run-time parallel check of seven different ITL specifications for doing a breadth-first walk down a tree. The specifications range from a register-transfer level description to a somewhat object-oriented approach based on parallel recursive decent by several processes. Furthermore, we checked some safety and liveness proofs for mutual exclusion presented in [17]. As time goes on, we hope to obtain more experience with the advantages and limitations of using Tempura for run-time checking of ITL assertions.

7 Discussions

We have presented the basis of a compositional methodology of specification and proof using fixpoints of various ITL operators. Issues considered include reasoning about safety, liveness and even equivalence of specifications. Our current work has identified an interesting class of commitments which can be used for compositional transformation and refinement of specifications. The exploitation of executable specifications based on ITL's programming language subset Tempura helps to accelerate development of both the underlying theory as well as practical tool support.

Much work remains to be done. We need to conduct larger case studies using by ITL and Tempura to ensure scalability of the techniques. Also, at present there is little experience with using compositionality in ITL together with a frame semantics for imperative destructive assignments developed by us in [16]. Furthermore, programming with Tempura has some difficulties. In particular, support for debugging of parallel programs needs improvement.

Acknowledgements

We wish to thank Antonio Cau, Nick Coleman, Li Xiaoshan and Hussein Zedan for discussions. The Engineering and Physical Sciences Research Council kindly funded our research.

References

1. Dutertre, B.: On first order interval temporal logic. In: 10th Annual IEEE Symposium on Logic in Computer Science. IEEE Computer Society Press, Los Alamitos, California (1995) 36–43
2. Francez, N., Pnueli, A.: A proof method for cyclic programs. Acta Inf. **9** (1978) 133–157
3. Halpern J., Manna Z., Moszkowski B.: A hardware semantics based on temporal intervals. In: Diaz, J. (Ed.) Proceedings of the 10th International Colloquium on Automata, Languages and Programming (ICALP'83). Lecture Notes in Computer Science Vol. 154. Springer-Verlag, Berlin Heidelberg New York (1983) 278–291
4. Hoare, C.A.R.: An axiomatic basis for computer programming. Comm. ACM **12** (1969) 576–580, 583
5. Jones, C.B.: Specification and design of (parallel) programs. In: Mason, R.E.A. (Ed.) Proceedings of Information Processing '83. North Holland Publishing Co., Amsterdam (1983) 321–332
6. Kesten, Y., Pnueli, A.: A complete proof system for QPTL. In: Proc. 10th IEEE Symp. on Logic in Computer Science. IEEE Computer Society Press, Los Alamitos, California (1995) 2–12
7. Kleene, S.C.: Mathematical Logic. John Wiley & Sons, Inc., New York (1967)
8. Kono, S.: A combination of clausal and non clausal temporal logic programs. In: Fisher, M., Owens, R. (Eds.) Executable Modal and Temporal Logics. Lecture Notes in Computer Science, Vol. 897. Springer-Verlag, Berlin Heidelberg New York (1995) 40–57

9. Kröger, F.: Temporal Logic of Programs. Springer-Verlag, Berlin Heidelberg New York (1987)
10. Manna, Z.: Verification of sequential programs: temporal axiomatization. In: Broy, M., Schmidt, G. (Eds.), Theoretical Foundations of Programming Methodology. D. Reidel Publishing Co. (1982) 53–102
11. Moszkowski B.: Reasoning about Digital Circuits. PhD thesis, Stanford University, Stanford, California (1983)
12. Moszkowski, B.: A temporal logic for multilevel reasoning about hardware. IEEE Computer **18** (1985) 10–19
13. Moszkowski, B.: Executing Temporal Logic Programs. Cambridge University Press, Cambridge, England (1986)
14. Moszkowski, B.: Some very compositional temporal properties. In: Olderog, E.-R. (Ed.) Programming Concepts, Methods and Calculi. IFIP Transactions, Vol. A-56, North-Holland (1994) 307–326
15. Moszkowski, B.: Compositional reasoning about projected and infinite time. In: Proceedings of the First IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95). IEEE Computer Society Press, Los Alamitos, California (1995) 238–245
16. Moszkowski, B.: Embedding imperative constructs in interval temporal logic. Internal memorandum EE/0895/M1. Dept. of Elec. and Elec. Eng., Univ. of Newcastle, Newcastle upon Type, UK (1995)
17. Moszkowski, B.: Using temporal fixpoints to compositionally reason about liveness. In: BCS-FACS 7th Refinement Workshop, "Electronic Workshops in Computing" series. Springer-Verlag, London (1996)
18. Paech, B.: Gentzen-systems for propositional temporal logics. In: Börger, E. et al. (Eds.) Proceedings of the 2nd Workshop on Computer Science Logic. Lecture Notes in Computer Science Vol. 385. Springer-Verlag, Berlin Heidelberg New York (1988) 240–253
19. Rosner, R., Pnueli, A.: A choppy logic. In: Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science. IEEE Computer Society Press, Los Alamitos, California (1986) 306–314

Appendix A Practical Proof System for ITL

In this appendix, we present a very powerful and practical compositional proof system for ITL. Our experience in rigorously developing hundreds of propositional and first-order proofs has helped us refine the axioms and convinced us they are sufficient for a very wide range of purposes. See Moszkowski [14] for more about this. The proof system is divided into a propositional part and a first-order part. Our discussion looks at each in turn.

Propositional Axioms and Inference Rules. The propositional axioms and inference rules mainly deal with *chop*, and *skip* and operators derived from them. Only one axiom is needed for *chop-star*. The proof system gives nearly equal treatment to initial and terminal subintervals. This is exceedingly important for the kinds of proofs we do. In addition, this makes the proof system easier to understand since much of it consists simply of duals in this sense. In contrast,

most temporal logics cannot handle initial subintervals and even other proof systems for ITL largely neglect them.

Rosner and Pnueli [19] and Paech [18] give propositional proof systems for ITL with infinite intervals and prove completeness. However, neither system has ever been used much. More recently, Kesten and Pnueli [6] were able to prove the completeness of a very nice proof system for *Quantified Propositional Temporal Logic* (QPTL) using Büchi Automata. Perhaps a similar technique can be applied to propositional ITL with infinite time since it has the same expressiveness as QPTL and can even be translated into it as shown by Halpern and Moszkowski in [11, pp. 23–24]. Our proof system presented here contains some of the propositional axioms suggested by Rosner and Pnueli but also includes our own axioms and inference rule for the operators $\boxdot$, *halt*, and *chop-star*. These assist in deducing propositional and first-order theorems and in deriving rules for importing, exporting and other important aspects of composition.

$$
\begin{array}{ll}
\textbf{Prop} & \vdash \text{Substitutions of tautologies} \\
\textbf{P2} & \vdash (S;T);U \equiv S;(T;U) \\
\textbf{P3} & \vdash (S \vee S');T \supset (S;T) \vee (S';T) \\
\textbf{P4} & \vdash S;(T \vee T') \supset (S;T) \vee (S;T') \\
\textbf{P5} & \vdash \mathit{empty}; S \equiv S \\
\textbf{P6} & \vdash S; \mathit{empty} \equiv S \\
\textbf{P7} & \vdash w \supset \boxdot w \\
\textbf{P8} & \vdash \boxdot(S \supset S') \wedge \Box(T \supset T') \supset (S;T) \supset (S';T') \\
\textbf{P9} & \vdash \bigcirc S \supset \neg \bigcirc \neg S \\
\textbf{P10} & \vdash \Diamonddot\big((\bigcirc\, \mathit{halt}\ w) \wedge S\big) \supset \boxdot\big((\bigcirc\, \mathit{halt}\ w) \supset S\big) \\
\textbf{P11} & \vdash S \wedge \Box(S \supset \textcircled{w} S) \supset \Box S \\
\textbf{P12} & \vdash S^* \equiv \mathit{empty} \vee (S \wedge \mathit{more}); S^* \\
\\
\textbf{MP} & \vdash S \supset T, \quad \vdash S \quad \Rightarrow \quad \vdash T \\
\Box\textbf{Gen} & \vdash S \quad \Rightarrow \quad \vdash \Box S \\
\boxdot\textbf{Gen} & \vdash S \quad \Rightarrow \quad \vdash \boxdot S
\end{array}
$$

We now give a sample theorem and its proof:

$$\vdash \quad \boxdot(S \supset T) \quad \supset \quad \Diamonddot S \supset \Diamonddot T \ .$$

Proof:

$$
\begin{array}{llll}
1 & \vdash & \mathit{true} \supset \mathit{true} & \textbf{Prop} \\
2 & \vdash & \Box(\mathit{true} \supset \mathit{true}) & 1,\Box\textbf{Gen} \\
3 & \vdash & \boxdot(S \supset T) \wedge \Box(\mathit{true} \supset \mathit{true}) & \textbf{P8} \\
  & \vdash & \qquad \supset \ (S;\mathit{true}) \supset (T;\mathit{true}) & \\
4 & \vdash & \boxdot(S \supset T) \ \supset \ (S;\mathit{true}) \supset (T;\mathit{true}) & 2,3,\textbf{Prop} \\
5 & \vdash & \boxdot(S \supset T) \ \supset \ \Diamonddot S \supset \Diamonddot T & 4,\text{def. of } \Diamonddot
\end{array}
$$

Theorem A. *The propositional proof system is complete for quantifier-free formulas containing only boolean-valued static and state variables.*

Outline of proof: For a given formula, we construct a finite tableau consisting of a number of states. Each state is represented as a disjunction whose disjuncts are themselves conjunctions of primitive propositions, *next* formulas and their negations. Now suppose S is a valid formula. Construct a tableau for its negation $\neg S$. Call a state in a tableau *final* if it is satisfiable by some empty interval. No state reachable from the initial state in our tableau for $\neg S$ is final, since otherwise we can use the path to construct a model for $\neg S$. Therefore the tableau reflects that $\neg S$ is not true in any finite intervals. We convert this to a proof-by-contradiction for S. This technique also applies to a version of Rosner and Pnueli's proof system restricted to finite intervals.

First-Order Axioms and Inference Rules. Below are axioms and inference rules for reasoning about first-order concepts. They are to be used together with the propositional ones already introduced. See Manna [10] and Kröger [9] for proof systems for *chop*-free first-order temporal logic. We let v and v' refer to both static and state variables.

F1 $\vdash$ All substitution instances of valid nonmodal formulas of conventional first-order logic with arithmetic.

F2 $\vdash \forall v\colon S \quad\supset\quad S_v^e$,
where the expression e is sort-compatible with v and v is free for e in S. If e contains any temporal operators, then v must be a state variable not occurring freely in S within the left side of a *chop* formula or within a *chop-star* formula.

F3 $\vdash \forall v\colon (S \supset T) \;\supset\; (S \supset \forall v\colon T)$,
where v doesn't occur freely in S.

F4 $\vdash (\imath v\colon S) = (\imath v'\colon S_v^{v'})$,
where v and v' are static variables of one sort and v is free for v' in S.

F5 $\vdash \forall v\colon (S \equiv T) \quad\supset\quad (\imath v\colon S) = (\imath v\colon T)$,
where v is static.

F6 $\vdash (\exists v\colon S) \wedge (\imath v\colon S) = v \quad\supset\quad S$,
where v is a static variable.

F7 $\vdash w \quad\supset\quad \Box w$,
where w only contains static variables.

F8 $\vdash \exists v\colon (S;T) \quad\supset\quad (\exists v\colon S); T$,
where v doesn't occur freely in T.

F9 $\vdash \exists v\colon (S;T) \quad\supset\quad S; (\exists v\colon T)$,
where v doesn't occur freely in S.

F10 $\vdash (\exists v\colon S); \bigcirc(\exists v\colon T) \;\supset\; \exists v\colon (S; \bigcirc T)$,
where v is a state variable.

∀Gen $\vdash S \quad\Rightarrow\quad \vdash \forall v\colon S$,
for any variable v.

Induct $\vdash S_n^0, \quad \vdash S \supset S_n^{n+1} \quad\Rightarrow\quad \vdash S$,
for any static variable n whose sort is the natural numbers.

The axiom **F1** permits using properties of conventional first-order logic with arithmetic without proof. Most of the other axioms and the two inference rules at the end are adaptations of conventional nonmodal equivalents for quantifiers and definite descriptions. Only four axioms actually contain temporal operators. Axiom **F7** deals with state formulas containing only static variables. The two axioms **F8** and **F9** show how to move an existential quantifier out of the scope of *chop*. The remaining temporal axiom **F10** shows how to combine two state variables in nearly adjacent subintervals into one state variable for the entire interval. We extensively use it and lemmas derived from it for constructing auxiliary variables. Dutertre [1] gives a complete first-order ITL proof system but unfortunately with a nonstandard semantics of intervals. In addition, it has not be developed with compositional proofs in mind.

A.1 Axioms for Infinite Time

The proof system for ITL with infinite time contains all the axioms and basic inference rules of the basic proof system. We also include the following two propositional axioms:

P13 $\vdash \quad (S \wedge \mathit{inf}\,);T \quad \equiv \quad S \wedge \mathit{inf}$,
P14 $\vdash \quad S \wedge \Box\big(S \supset (T \wedge \mathit{more});S\big) \;\supset\; T^{*}$.

The first-order axiom now given is sometimes needed for constructing auxiliary variables with *chop-star*:

F11 $\vdash \big(\forall v\colon \exists v'\colon (v = v' \wedge S)\big)^{*} \;\supset\; \forall v\colon \exists v'\colon (v = v' \wedge S^{*})$,
where v and v' are state variables and v does not occur freely S.

It may be that a complete axiom system for even propositional ITL with infinite intervals can only be achieved by means of a nonconventional inference rule. This is not central to our approach.