Appears as an invited paper in the Proceedings of the
BCS-FACS 7th Refinement Workshop (Bath, UK, 3–5 July, 1996),
He Jifeng, J. Cooke and P. Wallis (editors), in the series *electronic Workshops in Computing*,
Springer-Verlag, London, 1996.

Using Temporal Fixpoints to Compositionally Reason about Liveness

Ben Moszkowski*
Department of Electrical and Electronic Engineering
University of Newcastle upon Tyne, Newcastle NE1 7RU, Great Britain

Abstract

The compositional specification and verification of the behavior of concurrent processes is a challenging research area. The assumption/commitment approach has emerged as one way to systematically achieve the desired modularity. However, it is generally limited to reasoning about safety properties which apply throughout the execution of a system. Liveness properties involving intermittent behavior are harder to address. We investigate the use of assumptions and commitments in Interval Temporal Logic and show how to augment them with some more information for handling liveness. The proposed techniques are a continuation of our previous research on formalizing assumptions and commitments through the use of fixpoints of certain simple temporal operators. Associated with this is a generalized notion of Owicki and Gries' proof outlines. We illustrate the approach with examples including a mutual exclusion system with time stamps.

1 Introduction

Assumptions and commitments are recognized as one way to reasoning about concurrent systems. Jones [6] shows how to augment the pre- and post-conditions found in Hoare logic with compositional assumptions and commitments (called by Jones *rely*- and *guarantee-conditions*). Assumptions are especially selected so that if they are true for a system component, then they are automatically true for all sequential subcomponents. Thus one can say that such assumptions are easy to *import*. Commitments are chosen such that when true for a series of sequential subcomponents are also automatically true for the overall component. Thus, such commitments are easy to *export*. This style of analysis works best when one reasons about safety properties [11] which hold everywhere but it is more limited when dealing with liveness. One reason is that liveness properties are by definition not necessarily true all the time but only some of the time. Consequently, they are too weak to be exported with conventional compositional commitments.

Other researchers have investigated how to handle liveness. Stølen [23] deals with it by adding a *wait-condition* to Jones' approach. See also Xu and He [27], and Xu, Cau and Collette [24] and a survey by Xu, de Roever and He [26]. Pandya and Joseph [21] and Jonsson and Tsay [7] use linear-time temporal logic.

In this work we show how to augment compositional assumptions and commitments with some extra notation to bridge the gaps where exportable commitments by themselves are not sufficient. Interval Temporal Logic (ITL) [13, 4, 14], serves as our framework. In previous work [16] we characterized generalized versions of compositional assumptions and commitments in ITL as fixpoints of certain simple temporal operators. An application of this approach to intervals with infinite length was illustrated in [17]. We now show that fixpoints of some other ITL operators facilitate reasoning about liveness. Our approach also supports an extended form of the proof outlines of Owicki and Gries [18, 19] as a means to visually elucidate compositional proofs about both safety and liveness.

The remaining sections of the paper are organized as follows. In Section 2 we overview how to embed assumptions and commitments in Interval Temporal Logic and briefly discuss how to extend this to handle liveness. Section 3 gives a summary of ITL's syntax and semantics as well as a compositional proof system. Section 4 gives more details about dealing with liveness and includes various examples. For simplicity, Sections 3 and 4 only consider temporal intervals having finite length. Section 5 extends ITL and our compositional methods to reasoning about infinite intervals. We

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

include a number of small examples throughout Sections 4 and 5. One involves proving the absence of deadlock in a simple two-process system with shared write access to a counter variable. Section 6 looks at two nontrivial mutual exclusion systems operating over infinite time. The first is easier to describe but can deadlock. The second one remedies this deficiency through the use of time stamps. Both of these systems served as our primary case studies during the development of the methods presented here.

2 Overview of Assumptions and Commitments in ITL

Modularity is a desirable attribute of any formal method. We wish to address how to modularly specify and prove liveness properties. However, it is necessary to first present some background material. One of the best known modular logical notations is Hoare logic [5]. 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 Interval Temporal Logic we can express a Hoare clause as a theorem about discrete intervals of time consisting of one or more states:

$$w \wedge Sys \supset fin\, w'.$$

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 $fin\, w'$ is true on an interval iff w' is true in the interval's final state. A more precise definition of *fin* and other ITL operators is given in Section 3.

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 [3] 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*:

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

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 require that *As* and *Co* be respective fixpoints of the ITL operators $\boxdot$ (read "*box-a*") and * (read "*chop-star*") as is now shown:

$$As \equiv \boxdot As, \qquad Co \equiv Co^*.$$

The first equivalence ensures that if the assumption *As* is true on an interval, it is also true in all subintervals. The second ensures that if zero or more sequential instances of the commitment *Co* span an interval, *Co* is also true on the interval itself. The ITL operator * (*chop-star*) used here is a repetitive version of the chop operator mentioned above and is similar to the Kleene star found in regular expressions. The temporal formula $\Box(K = 1)$ (read "*K always equals 1*") is an example of an importable assumption. The temporal formula $K \leftarrow K$ ("*K's initial and final values on the interval are equal*") is an exportable commitment. 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*, where the formula *keep S*, for some subformula *S*, is true on an interval iff *S* is true on every unit subinterval (i.e., consisting of exactly two adjacent states). For assumptions and commitments obeying the above, the next derivable proof rule is sound:

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

The rule uses the ITL operator ";" (*chop*) which combines the formulas *Sys* and Sys' in series. That is, $Sys; Sys'$ is true on an interval iff *Sys* is true on a left subinterval and *Sys'* is true on the corresponding right subinterval which shares one state.

Here is an analogous rule for decomposing a proof for zero or more iterations of a formula *Sys*:

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

Similar rules are possible for *if*, *while* and other constructs.

Note that our approach only requires assumptions and commitments which are used directly in rules such (1) and (2) to be compositional. Compositional proofs about a system in ITL typically also involve reasoning about noncompositional assumptions and commitments as well. For instance, there is an important class of formulas using the standard temporal operator $\Box$ (read "*box*" or "*always*") and of the form $\Box S$ which often occur in temporal logic specifications. In general they can neither be used directly as compositional assumptions or commitments. However, those of the form $\Box w$, for some state formula w, can be used as assumptions since they are fixpoints of the operator $\boxed{a}$:

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

However, 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 not in general a theorem. However, there is a simple way around this. We 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 $\Box w$.

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. In what follows we investigate how to handle such temporal formulas in compositional proofs. This is facilitated through the exploitation of further fixpoints of ITL operators. In general, such formulas are not suitable as compositional assumptions or commitments. Let us now use the temporal operators $\boxed{m}$ ("*box-m*") and $\Diamond\!\!\!\!\cdot$ ("*diamond-i*"). A formula $\boxed{m}S$ 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 $\boxed{m}$ ignores the last (empty) terminal subinterval consisting of one state and is slightly weaker than $\Box$. A formula $\Diamond\!\!\!\!\cdot\, S$ is true on an interval iff S is true on some *initial* (prefix) subinterval (which might be the interval itself). It turns out that for any S, the formula $\boxed{m}\,\Diamond\!\!\!\!\cdot\, S$ is a fixpoint of *chop-star*:

$$\vdash \quad \boxed{m}\,\Diamond\!\!\!\!\cdot\, S \equiv (\boxed{m}\,\Diamond\!\!\!\!\cdot\, S)^*.$$

It therefore follows that for any formula DI which is a fixpoint of $\Diamond\!\!\!\!\cdot$, the formula $\boxed{m}DI$ is always a fixpoint of *chop-star* and is a suitable compositional commitment. Now, for any state formulas x and x', the formula $x \supset \Diamond x'$ is a fixpoint of the temporal operator $\Diamond\!\!\!\!\cdot$. Consequently, the formula $\boxed{m}(x \supset \Diamond x')$ is a fixpoint of *chop-star*. In order to prove a formula $\Box(x \supset \Diamond x')$, we compositionally establish the related formula $\boxed{m}(x \supset \Diamond x')$ and also show $\mathit{fin}(x \supset \Diamond x')$, thus obtaining $\Box(x \supset \Diamond x')$. Here we are using the following lemma relating $\Box$ with $\boxed{m}$ and *fin*:

$$\vdash \quad \Box S \equiv \boxed{m}S \wedge \mathit{fin}\, S.$$

There are also other useful fixpoints of $\Diamond\!\!\!\!\cdot$ which we consider later. In addition, the conjunction and disjunction of two such fixpoints are themselves fixpoints of $\Diamond\!\!\!\!\cdot$.

The fixpoints of the ITL operator $\Diamond\!\!\!\!\cdot_a$ (read "*diamond-a*") are also important. In general, $\Diamond\!\!\!\!\cdot_a S$ 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 $\Diamond\!\!\!\!\cdot_a$. If DA is a fixpoint of $\Diamond\!\!\!\!\cdot_a$, then it is also a fixpoint of $\Diamond\!\!\!\!\cdot$ so $\boxed{m}DA$ is a fixpoint of *chop-star* and hence a compositional commitment. More generally, the formula $x \supset DA$ is always a fixpoint of $\Diamond\!\!\!\!\cdot$ and therefore $\boxed{m}(x \supset DA)$ is a fixpoint of *chop-star*. The fixpoints of $\Diamond\!\!\!\!\cdot_a$ are closed under conjunction and disjunction.

Let us consider another benefit of fixpoints of $\Diamond\!\!\!\!\cdot_a$. Suppose one wishes to prove that a formula *Sys*; *Sys'* with a suitable precondition and an importable assumption implies a commitment $\boxed{m}(x \supset DA)$ for some state formula x and some fixpoint DA of $\Diamond\!\!\!\!\cdot_a$. 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 as long as all intervals are assumed for simplicity to have finite length:

$$\frac{\begin{array}{l}\vdash\ w \wedge As \wedge Sys \ \supset\ \mathit{fin}\, w',\\ \vdash\ w' \wedge As \wedge Sys' \ \supset\ \boxed{m}(x \supset DA) \wedge DA \wedge \mathit{fin}\, w''\end{array}}{\vdash\ w \wedge As \wedge (Sys; Sys') \ \supset\ \boxed{m}(x \supset DA) \wedge DA \wedge \mathit{fin}\, w''.} \qquad (3)$$

This shows that the only thing we need to verify about *Sys* is that it ensures that the statement w' is true in its final state. Both the desired commitment $\boxed{m}(x \supset DA)$ and *DA* itself can be obtained for *Sys*; *Sys*′ from *Sys*′ alone because *DA* is a fixpoint of $\diamondsuit$. Later on in Section 4 we illustrate how to use this proof technique. A variant rule for infinite time is discussed in Section 5.

In what follows, we will further examine the significance of fixpoints of such temporal operators as *chop-star*, $\boxed{a}$, $\diamondsuit$ and $\diamondsuit$ in reasoning about liveness.

3 Review of Interval Temporal Logic

We now describe Interval Temporal Logic. The presentation is rather brief and the reader should refer to references such as [13, 4, 14, 16] for more details. 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*, ..., *A*, *B*, *C*, ... to data values. Lower case variables *a*, *b*, *c*, ... are called *static* and do not vary over time. Infinite intervals can also be handled by us but for simplicity we do not consider them until Section 5. 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:

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

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 *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 serves as an alternative programming-like syntax for S^*:

for some times do S.

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. Table 1 shows a variety of useful temporal operators definable in ITL. A summary of the kinds of fixpoints used in our approach is shown in Table 2. It also includes examples of them. Note that in general the variables *As* and *Co* can refer respectively to arbitrary assumptions and commitments and they are fixpoints only when it is specifically stated, such as when they are supposed to be compositionally importable or exportable. This is the case most of the time in our presentation here. In the long run, it might be better to refer to an assumption which is a fixpoint of $\boxed{a}$ by the name *BA* and a commitment which is a fixpoint of *chop-star* by *CS*.

In [16] we made use of the conventional logical notion of *definite descriptions* of the form $\imath v: S$ where *v* is a variable and *S* is a formula (see for example Kleene [8, pages 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) and *len* (the interval's length). For example, $\bigcirc e$ can be defined as follows:

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

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

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

The following operator *stable* tests whether an expression's value changes and is also later needed by us:

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

Conventional linear-time temporal logic operators		
$\bigcirc S$	$\stackrel{\text{def}}{\equiv}$ *skip*; S	*Next*
$\textcircled{w} S$	$\stackrel{\text{def}}{\equiv} \neg \bigcirc \neg S$	*Weak next*
$\Diamond S$	$\stackrel{\text{def}}{\equiv}$ *true*; S	*Sometimes (some terminal subinterval)*
$\Box S$	$\stackrel{\text{def}}{\equiv} \neg \Diamond \neg S$	*Always (all terminal subintervals)*
Some other important operators		
more	$\stackrel{\text{def}}{\equiv} \bigcirc$ *true*	*Nonempty interval*
empty	$\stackrel{\text{def}}{\equiv} \neg$*more*	*Empty interval*
fin S	$\stackrel{\text{def}}{\equiv} \Box(\mathit{empty} \supset S)$	*Final state*
halt S	$\stackrel{\text{def}}{\equiv} \Box(S \equiv \mathit{empty})$	*Exactly final state*
$\boxed{\text{m}} S$	$\stackrel{\text{def}}{\equiv} \Box(\mathit{more} \supset S)$	*All nonempty terminal subintervals*
$\textcircled{\text{m}} S$	$\stackrel{\text{def}}{\equiv} \Diamond(\mathit{more} \wedge S)$	*Some nonempty terminal subinterval*
While and repeat loops		
while w *do* S	$\stackrel{\text{def}}{\equiv} (w \wedge S)^* \wedge \mathit{fin} \neg w$	
repeat S *until* w	$\stackrel{\text{def}}{\equiv} S$; *while* $\neg w$ *do* S	
More interval-oriented operators		
$\textcircled{\text{a}} S$	$\stackrel{\text{def}}{\equiv}$ *true*; S; *true*	*Some subinterval*
$\boxed{\text{a}} S$	$\stackrel{\text{def}}{\equiv} \neg \textcircled{\text{a}} \neg S$	*All subintervals*
$\textcircled{\text{i}} S$	$\stackrel{\text{def}}{\equiv} S$; *true*	*Some initial subinterval*
$\boxed{\text{i}} S$	$\stackrel{\text{def}}{\equiv} \neg \textcircled{\text{i}} \neg S$	*All initial subintervals*
keep S	$\stackrel{\text{def}}{\equiv} \boxed{\text{a}}(\mathit{skip} \supset S)$	*All* unit *subintervals*
keepnow S	$\stackrel{\text{def}}{\equiv} \textcircled{\text{i}}(\mathit{skip} \wedge S)$	*First unit subinterval*

Table 1: Some definable ITL operators

Operator	*Formula name*	*Sample formula*	*Meaning*
$\boxed{\text{a}}$	*As*	*stable* K	*"The variable K is stable"*
$\Box$	*BT*	*fin* Mk	*"The final value of Mk is true"*
chop-star	*Co*	$\boxed{\text{m}} \neg$ *stable* K	*"In all nonempty terminal subintervals, K is not stable"*
$\textcircled{\text{i}}$	*DI*	*even*$(K) \supset \neg$ *stable* K	*"If K is initially even, it is not stable"*
$\textcircled{\text{a}}$	*DA*	$\neg$ *stable* K	*"K is not stable"*
keep	(none)	*keep*$(K \leq \bigcirc K)$	*"K never decreases"*

Table 2: Various useful temporal fixpoints

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:

$$e \textit{ gets } e' \quad \stackrel{\text{def}}{=} \quad \textit{keep}(e \leftarrow e').$$

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

$$\textit{padded } e \quad \stackrel{\text{def}}{=} \quad \exists a{:}\, \textit{keep}(e = a),$$

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:

$$e \mathrel{\ll\!\sim} e' \quad \stackrel{\text{def}}{=} \quad (\textit{fin } e) = e' \wedge \textit{padded } e.$$

This ensures that e does not change until possibly the very end of the interval when the assignment takes effect.

3.1 A Practical Proof System

We now present a very powerful and practical compositional proof system for ITL. The reader may prefer to initially omit this subsection. 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 [16] 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.

3.1.1 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 [22] and Paech [20] give propositional proof systems for ITL with infinite intervals and prove completeness. Our proof system contains some of the propositional axioms suggested by Rosner and Pnueli but also includes our own axioms and inference rule for the operators $\boxdot$, *keepnow*, and *chop-star*. These assist in deducing propositional and first-order theorems and in deriving rules for importing, exporting and other aspects of composition.

Prop $\vdash$ *Substitutions of tautologies*
P2 $\vdash (S; T); U \equiv S; (T; U)$
P3 $\vdash (S \vee S'); T \supset (S; T) \vee (S'; T)$
P4 $\vdash S; (T \vee T') \supset (S; T) \vee (S; T')$
P5 $\vdash \textit{empty}; S \equiv S$
P6 $\vdash S; \textit{empty} \equiv S$

MP $\vdash S \supset T, \ \vdash S \ \Rightarrow \ \vdash T$
$\Box$**Gen** $\vdash S \ \Rightarrow \ \vdash \Box S$

P7 $\vdash w \supset \boxdot w$
P8 $\vdash \boxdot(S \supset S') \wedge \Box(T \supset T') \supset (S; T) \supset (S'; T')$
P9 $\vdash \bigcirc S \supset \neg \bigcirc \neg S$
P10 $\vdash \diamondsuit\big((\bigcirc \textit{halt } w) \wedge S\big) \supset \boxdot\big((\bigcirc \textit{halt } w) \supset S\big)$
P11 $\vdash S \wedge \Box(S \supset \circledcirc S) \supset \Box S$
P12 $\vdash S^* \equiv \textit{empty} \vee (S \wedge \textit{more}); S^*$

$\boxdot$**Gen** $\vdash S \ \Rightarrow \ \vdash \boxdot S$

We have strengthened axiom **P10** in order to facilitate reasoning about initial subintervals and what are called *markers* (see Subsect. 4.1). The following earlier version used in proofs about the *keep* operator can be readily deduced from **P10** (with w replaced by *true*), the other axioms and inference rules:

$$\vdash \quad \textit{keepnow } S \ \supset \ \neg \textit{ keepnow} \neg S.$$

We now give a sample theorem and its proof:

$$\vdash \quad \boxdot(S \supset T) \quad \supset \quad \diamondsuit S \supset \diamondsuit T$$

Proof:

1 $\vdash$ *true* $\supset$ *true* **Prop**

2	$\vdash$	$\Box(\mathit{true} \supset \mathit{true})$	*1,*$\Box$**Gen**
3	$\vdash$	$\boxdot(S \supset T) \wedge \Box(\mathit{true} \supset \mathit{true})$	**P8**
	$\vdash$	$\supset \quad (S; \mathit{true}) \supset (T; \mathit{true})$	
4	$\vdash$	$\boxdot(S \supset T) \quad \supset \quad (S; \mathit{true}) \supset (T; \mathit{true})$	*2,3,***Prop**
5	$\vdash$	$\boxdot(S \supset T) \quad \supset \quad \diamondsuit S \supset \diamondsuit T$	*4,def. of* $\diamondsuit$

Theorem 3.1 *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.

3.1.2 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 [12] and Kröger [10] 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 $\vdash \forall v\colon S \quad \supset \quad S^e_v,$
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.

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

Induct $\vdash S^0_n, \quad \vdash S \supset S^{n+1}_n \quad \Rightarrow \quad \vdash S,$
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 [2] gives a complete first-order ITL proof system but with a nonstandard semantics of intervals.

4 More about Analyzing Liveness

We now review and expand upon some of the ideas presented in Section 2 about liveness. As mentioned there, a formula of the form $\Box S$ can not in general be used as a compositional assumption or commitment. A formula of the more restricted form $\Box w$, for some state formula w, can be used directly as an importable assumption and a minor variant of it of the form *keep* w can be used as an exportable commitment which is ultimately combined with *fin* w to establish $\Box w$. Now there are many other important formulas of the form $\Box S$ which are not addressed by this. We now consider one way to deal with at least some such formulas. Recall that $\boxdot S$ to be true on an interval iff S is true on all terminal (suffix) subintervals except possibly the last one:

$$\boxdot S \quad \stackrel{\text{def}}{\equiv} \quad \Box(\mathit{more} \supset S).$$

Note that for a state formula w the formulas *keep* S and $\boxdot S$ are in fact provably equivalent:

$$\vdash \quad \mathit{keep}\, w \equiv \boxdot w.$$

An important observation already noted in Section 2 is that for any S, the formula $\boxdot \diamondsuit S$ is a fixpoint of *chop-star*:

$$\vdash \quad \boxdot \diamondsuit S \equiv (\boxdot \diamondsuit S)^*.$$

It follows that for any $\diamondsuit$-fixpoint DI, the formula $\boxdot DI$ is a fixpoint of *chop-star* and is suitable for use as a compositional commitment. Important liveness-related formulas such as $\diamondsuit x$ and $x \supset \diamondsuit x'$ are in fact fixpoints of $\diamondsuit$, provided that x and x' are state formulas. The important construct *keep* S which is frequently used as a compositional assumption and commitment in proofs can be expressed with the $\boxdot$ operator and a $\diamondsuit$-fixpoint:

$$\vdash \quad \mathit{keep}\, S \equiv \boxdot \diamondsuit(\mathit{skip} \wedge S).$$

Furthermore, all conjunctions and disjunctions of fixpoints of $\diamondsuit$ are themselves fixpoints of $\diamondsuit$.

Consider for example the formula $(K \lessdot\sim K+1)^*$. We can compositionally prove that this implies the formula $\boxdot\neg\, \mathit{stable}\, K$. The formula $\boxdot\neg\, \mathit{stable}\, K$ can be directly used as the commitment in the proof because the subformula $\neg\, \mathit{stable}\, K$ is provably a fixpoint of $\diamondsuit$:

$$\vdash \quad \neg\, \mathit{stable}\, K \equiv \diamondsuit \neg\, \mathit{stable}\, K.$$

In fact it is even a fixpoint of $\diamondsuit$. We first deduce that $K \lessdot\sim K+1$ implies the commitment $\boxdot\neg\, \mathit{stable}\, K$:

$$\vdash \quad K \lessdot\sim K+1 \quad \supset \quad \boxdot\neg\, \mathit{stable}\, K.$$

Next, we use this in a simplified version of proof rule (1) in Section 2 which omits pre- and post-conditions as well as assumptions to obtain the desired goal:

$$\vdash \quad (K \lessdot\sim K+1)^* \quad \supset \quad \boxdot\neg\, \mathit{stable}\, K.$$

Let us now look at how to prove the following in a compositional way:

$$\vdash \quad K \geq 1 \wedge (\mathit{stable}\, K; K \lessdot\sim 2K)^* \quad \supset \quad \boxdot\neg\, \mathit{stable}\, K \wedge \mathit{fin}(K \geq 1). \tag{4}$$

Note that we are unable to obtain the commitment $\boxdot\neg\, \mathit{stable}\, K$ from the first part of each loop step, that is *stable* K, by purely local analysis. In order to get around this, more proof techniques are needed as was noted in Section 2. First of all, in addition to compositional assumptions and commitments, we sometimes include another formula DA which is some fixpoint of the temporal operator $\diamondsuit$. Here is the general form of a suitable implication:

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

Now the formula $x \supset DA$ is a fixpoint of $\diamondsuit$. Among other things, this ensures that the formula $\boxdot(x \supset DA)$ is a fixpoint of *chop-star* and hence a compositional commitment. However, DA itself is not in general a fixpoint of *chop-star* and therefore cannot be exported in the usual way. However, if we have a specification $\mathit{Sys}; \mathit{Sys}'$ and can show that under appropriate circumstances the second subformula Sys' implies both $\boxdot(x \supset DA)$ and DA, then both $\boxdot(x \supset DA)$ and

$$\left.\begin{array}{l}\{K \geq 1\}\\ K \lessapprox 2K\\ \{K \geq 1\}\end{array}\right] \boxdot\neg\, \mathit{stable}\, K \wedge \neg\, \mathit{stable}\, K$$

Figure 1: Proof outline for lemma (7).

$$\begin{array}{l}\{K \geq 1\}\\ \mathit{for\ some\ times\ do}\ (\\ \quad \left.\begin{array}{l}\{K \geq 1\}\\ \mathit{stable}\, K\\ \{K \geq 1\}\end{array}\right] \mathit{true}\\ \quad \left.\begin{array}{l}\{K \geq 1\}\\ K \lessapprox 2K\\ \{K \geq 1\}\end{array}\right] \boxdot\neg\, \mathit{stable}\, K \wedge \neg\, \mathit{stable}\, K\\ )\\ \{K \geq 1\}\end{array}\left.\begin{array}{l}\\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right] \boxdot\neg\, \mathit{stable}\, K \left.\begin{array}{l}\\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right] \boxdot\neg\, \mathit{stable}\, K$$

Figure 2: Proof outline for lemma (4).

DA can be automatically exported from $Sys; Sys'$. This avoids first proving that *Sys* implies $\boxdot(x \supset DA)$. Here is a derivable proof rule for fixpoints of *DA*:

$$\frac{\vdash\ w \wedge As \wedge Sys \ \supset\ \mathit{fin}\, w',\quad \vdash\ w' \wedge As \wedge Sys' \ \supset\ \boxdot(x \supset DA) \wedge DA \wedge \mathit{fin}\, w''}{\vdash\ w \wedge As \wedge (Sys; Sys') \ \supset\ \boxdot(x \supset DA) \wedge DA \wedge \mathit{fin}\, w''.} \tag{5}$$

The following corollary rule uses $\boxdot DA$ as the commitment by replacing *x* by *true* and simplifying:

$$\frac{\vdash\ w \wedge As \wedge Sys \ \supset\ \mathit{fin}\, w',\quad \vdash\ w' \wedge As \wedge Sys' \ \supset\ \boxdot DA \wedge DA \wedge \mathit{fin}\, w''}{\vdash\ w \wedge As \wedge (Sys; Sys') \ \supset\ \boxdot DA \wedge DA \wedge \mathit{fin}\, w''.} \tag{6}$$

Such a rule provides a way to export the commitment $\boxdot\neg\, \mathit{stable}\, K$ in the example above since $\neg\, \mathit{stable}\, K$ is in fact a fixpoint of $\diamondsuit$. Below is an application of proof rule (6) to the example:

$$\frac{\vdash\quad K \geq 1 \wedge \mathit{true} \wedge \mathit{stable}\, K \ \supset\ \mathit{fin}\, K \geq 1,\quad \vdash\quad K \geq 1 \wedge \mathit{true} \wedge K \lessapprox 2K \ \supset\ \boxdot\neg\, \mathit{stable}\, K \wedge \neg\, \mathit{stable}\, K \wedge \mathit{fin}\, K \geq 1}{\vdash\quad K \geq 1 \wedge \mathit{true} \wedge (\mathit{stable}\, K; K \lessapprox 2K) \ \supset\ \boxdot\neg\, \mathit{stable}\, K \wedge \neg\, \mathit{stable}\, K \wedge \mathit{fin}\, K \geq 1.}$$

Once this is done, we simply export the commitment $\boxdot\neg\, \mathit{stable}\, K$ from the iterative formula $(\mathit{stable}\, K; K \lessapprox 2K)^*$ using proof rule (2) found in Section 2. Owicki and Gries [18, 19] developed *proof outlines* as a visual tool for concisely reasoning about Hoare clauses. Figure 1 contains a small proof outline generalized to include a commitment and which corresponds to the following lemma:

$$\vdash\quad K \geq 1 \wedge \mathit{true} \wedge K \lessapprox 2K \quad \supset \quad \boxdot\neg\, \mathit{stable}\, K \wedge \neg\, \mathit{stable}\, K \wedge \mathit{fin}\, K \geq 1. \tag{7}$$

The proof outline shows the pre- and post-conditions, the sequential component $K \lessapprox 2K$ and on the right of the large bracket is the resulting commitment. We keep the assumption implicit here and in other proof outlines since it is usually importable and remains the same in all sequential subcomponents. A generalized proof outline for lemma (4) is shown in Figure 2. In particular, it illustrates the composition of commitments of subcomponents into those for larger parts of a system.

A weakened variant of rule (6) is sometimes used to export *DA* from the left part of $Sys; Sys'$ without exporting $\boxdot DA$ as well:

$$\frac{\vdash\ w \wedge As \wedge Sys \ \supset\ DA \wedge \mathit{fin}\, w',\quad \vdash\ w' \wedge As \wedge Sys' \ \supset\ \mathit{fin}\, w''}{\vdash\ w \wedge As \wedge (Sys; Sys') \ \supset\ DA \wedge \mathit{fin}\, w''.}$$

$$\left.\begin{array}{l}\{K=j\}\\ \mathit{stable}\,K\\ \{K=j\}\end{array}\right\rceil \boxdot(K=j) \qquad \left.\begin{array}{l}\{K=j\}\\ K \lessapprox 2K\\ \{K=2j\}\end{array}\right\rceil \begin{array}{l}\boxdot(K=j)\\ \wedge\ \boxdot\Diamond K=2j\\ \wedge\ \Diamond K=2j\end{array} \qquad \left.\begin{array}{l}\boxdot(K=j)\\ \wedge\ \boxdot\Diamond K=2j\\ \wedge\ \Diamond K=2j\end{array}\right\rceil \boxdot(K=i \supset \Diamond K=2i)$$

Figure 3: Proof outline for lemma (9).

Here is another theorem we wish to verify about our example by using a $\diamondsuit$-fixpoint:

$$\vdash \quad (\mathit{stable}\,K; K \lessapprox 2K)^* \wedge \mathit{fin}(K=n) \ \supset\ \Box(K=i \wedge i \neq n \ \supset\ \Diamond K=2i). \tag{8}$$

The proof of this theorem introduces an auxiliary static variable j and the formula $\Diamond K=2j$ is used as a $\diamondsuit$-fixpoint. A generalization of rule (6) permitting an additional exportable commitment *Co* is utilized:

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

An overview of the proof of theorem (8) is given below. Most pre- and post-conditions as well as all assumptions are simply *true* and we omit them:

1.		$\vdash$	$K=j \wedge \mathit{stable}\,K \ \supset\ \boxdot(K=j) \wedge \mathit{fin}(K=j)$
2.		$\vdash$	$K=j \wedge K \lessapprox 2K \ \supset\ \boxdot(K=j) \wedge \boxdot\Diamond K=2j \wedge \Diamond K=2j$
3.	$1,2 \Rightarrow$	$\vdash$	$K=j \wedge (\mathit{stable}\,K; K \lessapprox 2K) \ \supset\ \boxdot(K=j) \wedge \boxdot\Diamond K=2j \wedge \Diamond K=2j$
4.		$\vdash$	$\boxdot(K=j) \wedge \boxdot\Diamond K=2j \ \supset\ \boxdot(K=i \ \supset\ \Diamond K=2i)$
5.	$3,4 \Rightarrow$	$\vdash$	$K=j \wedge (\mathit{stable}\,K; K \lessapprox 2K) \ \supset\ \boxdot(K=i \ \supset\ \Diamond K=2i)$
6.	$5 \Rightarrow$	$\vdash$	$(\mathit{stable}\,K; K \lessapprox 2K) \ \supset\ \boxdot(K=i \ \supset\ \Diamond K=2i)$
7.	$6 \Rightarrow$	$\vdash$	$(\mathit{stable}\,K; K \lessapprox 2K)^* \ \supset\ \boxdot(K=i \ \supset\ \Diamond K=2i)$
8.		$\vdash$	$\boxdot(K=i \ \supset\ \Diamond K=2i) \wedge \mathit{fin}(K=n) \ \supset\ \Box(K=i \wedge i \neq n \ \supset\ \Diamond K=2i)$
9.	$7,8 \Rightarrow$	$\vdash$	$(\mathit{stable}\,K; K \lessapprox 2K)^* \wedge \mathit{fin}(K=n) \ \supset\ \Box(K=i \wedge i \neq n \ \supset\ \Diamond K=2i).$

Figure 3 depicts a proof outline for the lemma now given about the body of the loop:

$$\vdash \quad (\mathit{stable}\,K; K \lessapprox 2K) \ \supset\ \boxdot(K=i \ \supset\ \Diamond K=2i). \tag{9}$$

The results of this are used in Figure 4 for the proof outline of a lemma about the overall loop:

$$\vdash \quad (\mathit{stable}\,K; K \lessapprox 2K)^* \ \supset\ \boxdot(K=i \ \supset\ \Diamond K=2i). \tag{10}$$

4.1 Markers

Here is a variant of the previous example with the two sequential parts of the loop body exchanged:

$$(K \lessapprox 2K; \mathit{stable}\,K)^* \wedge \mathit{fin}(K=n).$$

Once again we wish to prove that this implies the formula

$$\Box(K=i \wedge i \neq n \ \supset\ \Diamond K=2i).$$

However, the proof is more complicated since we can no longer readily propagate the $\diamondsuit$-fixpoint $\diamondsuit K=2j$ from $K \lessapprox 2K$ back to $\mathit{stable}\,K$. A more powerful technique for analyzing reachability is needed. We now introduce the

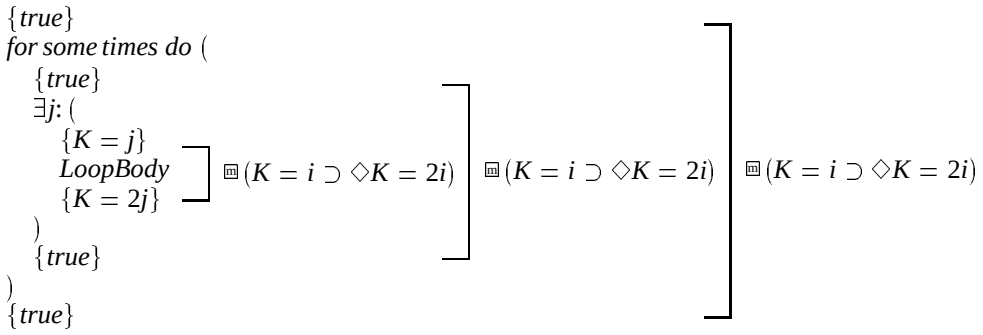

Figure 4: Proof outline for lemma (10).

notion of *marker*. This is a boolean state variable, called here *Mk*, which is true exactly at the start and end of loop iterations. A variant of *chop-star* having a marker can be defined as follows:

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

See Table 1 for the definition of the *halt* construct. The following programming syntax is also used for this kind of *chop-star*:

$$\mathit{for}_{Mk}\ \mathit{some\ times\ do}\ S.$$

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

$$\vdash \quad S^* \equiv \exists Mk{:}\, (Mk \wedge \mathit{chopstar}_{Mk} S) ,$$

where *Mk* does not occur freely in the formula *S*. The marker facilitates postponing reachability of a $\diamondsuit$-fixpoint until a later loop iteration. We originally considered markers in [13, page 127].

In our example, we introduce such a marker and prove the following lemma about liveness:

$$\begin{array}{ll} \vdash & Mk \wedge \mathit{chopstar}_{Mk}(K \lessapprox 2K; \mathit{stable}\, K) \\ \quad \supset & \quad \boxdot\big(K = i \ \supset\ \diamondsuit(K = 2i) \vee \diamondsuit(Mk \wedge K = i)\big) \\ & \quad \wedge \boxdot\big(Mk \wedge K = i \ \supset\ \diamondsuit K = 2i\big) . \end{array} \tag{11}$$

The two commitments are combined using temporal reasoning and ensure that whenever $K = i$, eventually either $K = 2i$ or the overall interval finishes with *K* still equalling *i*:

$$\begin{array}{ll} \vdash & \boxdot\big(K = i \ \supset\ \diamondsuit(K = 2i) \vee \diamondsuit(Mk \wedge K = i)\big) \\ & \wedge \boxdot(Mk \wedge K = i \ \supset\ \diamondsuit K = 2i) \\ & \qquad \supset\ \boxdot\big(K = i \ \supset\ (\diamondsuit K = 2i \vee \mathit{fin}\, K = i)\big) . \end{array} \tag{12}$$

After hiding the marker, we obtain the next lemma:

$$\vdash \quad (K \lessapprox 2K; \mathit{stable}\, K)^* \quad \supset \quad \boxdot\big(K = i \ \supset\ (\diamondsuit K = 2i \vee \mathit{fin}\, K = i)\big) .$$

The overall reduced commitment together with the originally given postcondition $\mathit{fin}(K = n)$ imply the desired formula:

$$\vdash \quad \boxdot\big(K = i \ \supset\ (\diamondsuit K = 2i \vee \mathit{fin}\, K = i)\big) \wedge \mathit{fin}(K = n) \quad \supset \quad \Box(K = i \wedge i \neq n \ \supset\ \diamondsuit K = 2i) .$$

$$
\begin{array}{l}
\{Mk \wedge K = j\} \\
K \lessapprox 2K \\
\{K = 2j\} \\
stable\, K \\
\{Mk \wedge K = 2j\}
\end{array}
\left[\begin{array}{l}
\boxdot K = j \\
\wedge\ \boxdot \Diamond K = 2j \\
\wedge\ \Diamond K = 2j
\end{array}\right]
\left[\begin{array}{l}
\boxdot K = 2j \\
\wedge\ \boxdot \Diamond (Mk \wedge K = 2j)
\end{array}\right]
\left[\begin{array}{l}
\boxdot (K = j \vee K = 2j) \\
\wedge\ \boxdot \Diamond \big(K = 2j \\
\qquad \vee\ (Mk \wedge K = j)\big) \\
\wedge\ \Diamond K = 2j
\end{array}\right]
Co \wedge DI
$$

Figure 5: Proof outline for lemma (17).

The proof of lemma (11) requires a $\diamondsuit$-fixpoint called here *DA* and a fixpoint of the temporal operator $\Box$ referred to this formula as *BT*. In our treatment of loops, *BT* is always $fin\, Mk$. Here is the generalized version of proof rule (5) for *chop*:

$$\frac{\begin{array}{l} \vdash\ w \wedge As \wedge Sys\ \supset\ fin\, w', \\ \vdash\ w' \wedge BT \wedge As \wedge Sys'\ \supset\ \boxdot (x \supset DA) \wedge DA \wedge fin\, w'' \end{array}}{\vdash\ w \wedge BT \wedge As \wedge (Sys; Sys')\ \supset\ \boxdot (x \supset DA) \wedge DA \wedge fin\, w'.} \tag{13}$$

The proof rule now given is for the version of *chop-star* with a marker. It uses a $\diamondsuit$-fixpoint *DI*:

$$\frac{\vdash\ w \wedge fin\, Mk \wedge As \wedge Sys\ \supset\ DI \wedge fin\, w}{\vdash\ w \wedge As \wedge chopstar_{Mk} Sys\ \supset\ \boxdot (Mk \supset DI) \wedge fin\, w.} \tag{14}$$

The following variants of these rules permit an additional commitment *Co* and are in fact the ones used for proving lemma (17) shown above.

$$\frac{\begin{array}{l} \vdash\ w \wedge As \wedge Sys\ \supset\ Co \wedge fin\, w', \\ \vdash\ w' \wedge BT \wedge As \wedge Sys'\ \supset\ Co \wedge \boxdot (x \supset DA) \wedge DA \wedge fin\, w'' \end{array}}{\vdash\ w \wedge BT \wedge As \wedge (Sys; Sys')\ \supset\ Co \wedge \boxdot (x \supset DA) \wedge DA \wedge fin\, w'.} \tag{15}$$

$$\frac{\vdash\ w \wedge fin\, Mk \wedge As \wedge Sys\ \supset\ Co \wedge DI \wedge fin\, w}{\vdash\ w \wedge As \wedge chopstar_{Mk} Sys\ \supset\ Co \wedge \boxdot (Mk \supset DI) \wedge fin\, w.} \tag{16}$$

The analysis of the loop body $K \lessapprox 2K; stable\, K$ involves the use of rule (15) to deduce the lemma below:

$$\begin{array}{l} \vdash\quad true \wedge fin\, Mk \wedge true \wedge (K \lessapprox 2K; stable\, K) \\ \qquad \supset\quad \boxdot \big(K = i\ \supset\ \Diamond (K = 2i) \vee \Diamond (Mk \wedge K = i)\big) \\ \qquad\qquad \wedge\ (K = i \supset \Diamond K = 2i) \wedge fin\, true. \end{array} \tag{17}$$

Figure 5 and Figure 6 show proof outlines for establishing lemmas (17) and (11), respectively. In these, *Co* stands for the exportable commitment

$$\boxdot \big(K = i\ \supset\ \Diamond K = 2i \vee \Diamond (Mk \wedge K = i)\big)$$

and *DI* stands for the $\diamondsuit$-fixpoint $K = i \supset \Diamond K = 2i$.

5 ITL with Infinite Time

The semantics and proof system 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 this means $S; T$ is true on such 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

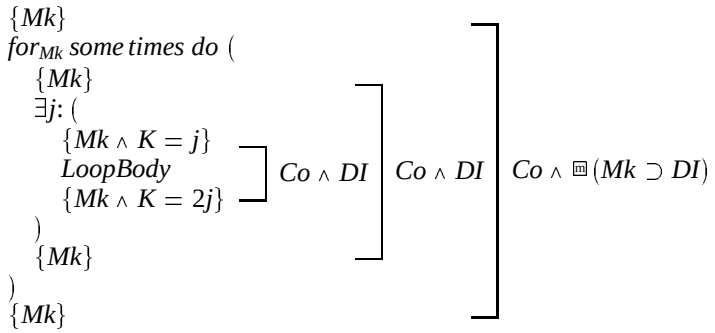

Figure 6: Proof outline for lemma (11).

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 $\Diamond$:

$$\begin{array}{llll} \mathit{inf} & \stackrel{\text{def}}{\equiv} & \mathit{true}; \mathit{false} & \qquad \mathit{finite} \stackrel{\text{def}}{\equiv} \neg \mathit{inf} \\ \Diamond S & \stackrel{\text{def}}{\equiv} & \mathit{finite}; S & \qquad \mathit{sfin}\, S \stackrel{\text{def}}{\equiv} \Diamond(\mathit{empty} \wedge S). \end{array}$$

Here $\mathit{sfin}\, S$ is a strong version of $\mathit{fin}\, S$ and is true only on finite intervals. In contrast, $\mathit{fin}\, S$ is vacuously true on all infinite intervals. As we have noted, the formula S^* can be true on an infinite interval where S occurs infinitely often in successive subintervals each having finite length. We denote this by S^ω (read "*chop-omega*") and define it in the following way:

$$S^\omega \stackrel{\text{def}}{\equiv} (S \wedge \mathit{finite})^* \wedge \mathit{inf}.$$

The other possibility for S^* on an infinite interval involves a formula S being true for a finite number of successive subintervals, but where the last one has infinite length, and is possibly the interval itself. The construct S^∞ (read "*chop-infinity*") is used to denote the union of both kinds of behavior for *chop-star* on infinite intervals:

$$S^\infty \stackrel{\text{def}}{\equiv} S^* \wedge \mathit{inf}.$$

The syntax *forever do S* is sometimes used as an alternative programming-language based notation for S^∞. In addition, the following two variant notations permit referencing a marker such as *Mk*:

$$\mathit{chopinf}_{Mk} S, \qquad \mathit{forever}_{Mk}\ \mathit{do}\ S.$$

The first-order operators for *temporal assignment* and *padded temporal assignment* are redefined to be true only on finite intervals:

$$\begin{array}{lll} e \leftarrow e' & \stackrel{\text{def}}{\equiv} & \mathit{finite} \wedge (\mathit{fin}\, e) = e', \\ e \mathrel{\ll\!\sim} e' & \stackrel{\text{def}}{\equiv} & \mathit{finite} \wedge (\mathit{fin}\, e) = e' \wedge \mathit{padded}\, e. \end{array}$$

Once this is done, all the axioms and basic inference rules remain sound. 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 $\vdash \big(\forall v: \exists v': (v = v' \wedge S)\big)^* \supset \forall v: \exists v': (v = v' \wedge S^*),$
where v and v' are state variables and v does not occur freely S.

It seems likely that completeness in the sense of Theorem 3.1 can only be achieved with a nonconventional inference rule. This is not central to our approach.

$$\begin{array}{l}\{K=j\}\\ \mathit{stable}\,K \wedge \mathit{finite}\\ \{K=j\}\\ K \mathrel{\leftsquigarrow} 2K\\ \{K=2j\}\end{array}\Bigg] \; \boxdot(K=j) \wedge \mathit{finite} \; \Bigg[\begin{array}{l}\boxdot(K=j)\\ \wedge\ \boxdot\Diamond K=2j\\ \wedge\ \Diamond K=2j\end{array}\Bigg] \; \boxdot(K=i \supset \Diamond K=2i)$$

Figure 7: Proof outline for lemma (19).

5.1 A Simple Example Involving Infinite Time

The following example requires reasoning about infinite intervals:

$$\vdash \quad \big((\mathit{stable}\,K \wedge \mathit{finite}); K \mathrel{\leftsquigarrow} 2K\big)^{\infty} \;\supset\; \Box(K=i \;\supset\; \Diamond K=2i).$$

As it shows, when analyzing such intervals, one must sometimes explicitly specify or prove that certain subintervals have finite length.

Proofs can use the fact that the operators $\Box$ and $\boxdot$ are equivalent on infinite intervals:

$$\vdash \quad \mathit{inf} \quad\supset\quad \Box S \equiv \boxdot S.$$

Once infinite intervals are permitted, one must use rules such as the following variant of derived rule (18) for reachability proofs:

$$\frac{\begin{array}{l}\vdash\ w \wedge \mathit{As} \wedge \mathit{Sys} \;\supset\; \mathit{finite} \wedge \mathit{fin}\,w',\\ \vdash\ w' \wedge \mathit{As} \wedge \mathit{Sys}' \;\supset\; \boxdot(x \supset \mathit{DA}) \wedge \mathit{DA} \wedge \mathit{fin}\,w''\end{array}}{\vdash\ w \wedge \mathit{As} \wedge (\mathit{Sys}; \mathit{Sys}') \;\supset\; \boxdot(x \supset \mathit{DA}) \wedge \mathit{DA} \wedge \mathit{fin}\,w''.} \tag{18}$$

Here we ensure that the interval satisfying *Sys* is finite, thus guaranteeing that *DA* does indeed occur in $\mathit{Sys}; \mathit{Sys}'$. Note that the formula *finite* is not a fixpoint of *chop-star*. See Figures 7 and 8 for respective proof outlines for the following lemmas needed for the above example:

$$\vdash \quad \big((\mathit{stable}\,K \wedge \mathit{finite}); K \mathrel{\leftsquigarrow} 2K\big) \;\supset\; \boxdot(K=i \;\supset\; \Diamond K=2i), \tag{19}$$

$$\vdash \quad \big((\mathit{stable}\,K \wedge \mathit{finite}); K \mathrel{\leftsquigarrow} 2K\big)^{\infty} \;\supset\; \Box(K=i \;\supset\; \Diamond K=2i). \tag{20}$$

Without loss of generality, Figure 8 uses existential quantification to introduce a static variable j equally K's value at the beginning of each loop iteration. A modified version of rule (18) permitting other exportable commitments as well is used:

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

Sometimes each of a sequence of components implies *finite*. This can be reduced by means of the derived rule now given:

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

5.2 Compositionally Proving Absence of Deadlock

Let us now present a compositional analysis proving the absence of deadlock. This small example illustrates shared write access and involves fewer concepts in its specification and analysis than the mutual exclusion examples considered later in Section 6. For instance, no markers or other auxiliary variables are required here. Figure 9 shows two simple

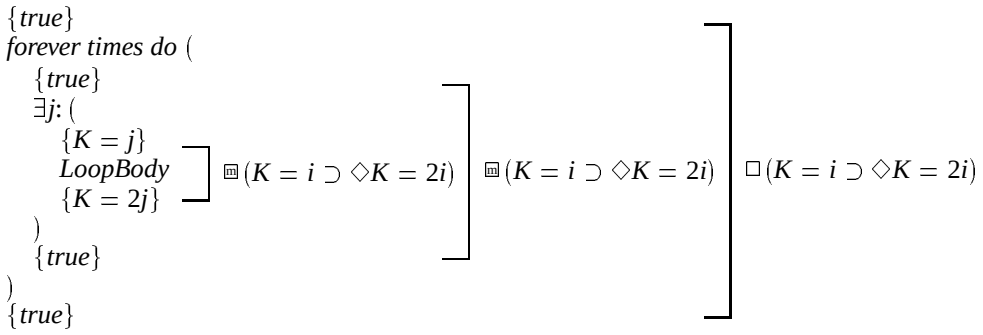

Figure 8: Proof outline for lemma (20).

$$F1 \stackrel{\text{def}}{\equiv} \textit{for some times do} \; (\; K \ll\sim K+1; \; \textit{halt even}(K) \;) \qquad F2 \stackrel{\text{def}}{\equiv} \textit{for some times do} \; (\; \textit{halt odd}(K); \; K \ll\sim K+1 \;)$$

Figure 9: A simple parallel system

processes *F1* and *F2* which alternately modify a single variable K. The iterating in *F1* and *F2* is expressed by means of the *chop-star* operator in the notation of a *for-loop*. The predicates *even* and *odd* are simple arithmetic tests. Here is the overall system together with K initially equal to 0:

$$K = 0 \wedge F1 \wedge F2.$$

When K is even, *F1* keeps it stable for a while and then eventually increments it, thus making it odd. At this time, *F2* keeps K stable and then increments it, thus handing responsibility for it back to *F1*. This continues for some unspecified, possibly infinite number of times. We use padded temporal assignments in order to ensure proper communication between *F1* and *F2*.

Here is a theorem describing correctness of the overall system:

$$\vdash \quad \textit{even}(K) \wedge F1 \wedge F2 \quad \supset \quad \textit{keep}(K \leq \bigcirc K \leq K+1) \wedge \textit{fin even}(K).$$

The theorem uses the *keep* operator defined earlier to state that K is always stable or increases by 1 over pairs of adjacent states. In addition, K's final value is even. In [16] we consider how to compositionally prove this safety property. The proof holds for both finite and infinite intervals.

The discussion so far only deals with showing that the variable K continues to remain stable or increase. It remains for us to ensure that when the combined system operates over an infinite interval, K never gets stuck at some value. Here is an ITL theorem which expresses this:

$$\vdash \quad \textit{inf} \wedge \textit{even}(K) \wedge F1 \wedge F2 \quad \supset \quad \Box\neg \, \textit{stable}\, K.$$

The next lemma plays an important role in our overall analysis:

$$\vdash \quad A \neq B \wedge A \ll\sim B \quad \supset \quad \text{▣}\neg \, \textit{stable}\, A.$$

This states that if the state variable A is padded and its initial and final values in the (finite) interval differ, then A is not stable in all nonempty terminal subintervals. Here is a slightly simplified substitution instance of this that is used when the variable K increases by 1:

$$\vdash \quad K \ll\sim K+1 \quad \supset \quad \text{▣}\neg \, \textit{stable}\, K.$$

We have omitted the subformula $K \neq K+1$ since it is trivially true. The following lemmas consider the behavior of *F1* and *F2*:

$$\vdash \quad even(K) \wedge F1 \supset \boxdot\big(even(K) \supset \neg\, stable\, K\big) \wedge fin\, even(K), \tag{21}$$
$$\vdash \quad even(K) \wedge F2 \supset \boxdot\big(odd(K) \supset \neg\, stable\, K\big) \wedge fin\, even(K). \tag{22}$$

Note that *fin* is defined to be weak and is therefore trivially true for infinite intervals. The proofs compositionally use lemmas such as those below for the sequential parts of *F1*:

$$\vdash \quad even(K) \wedge K \lessapprox K+1 \supset \boxdot\big(even(K) \supset \neg\, stable\, K\big) \wedge fin\, odd(K),$$
$$\vdash \quad odd(K) \wedge halt\, even(K) \supset \boxdot\big(even(K) \supset \neg\, stable\, K\big) \wedge fin\, even(K).$$

We combine lemmas (21) and (22) for *F1* and *F2* in parallel to obtain the following:

$$\vdash \quad even(K) \wedge F1 \wedge F2 \quad \supset \quad \boxdot \neg\, stable\, K. \tag{23}$$

Our assumption about infinite time is then introduced in the following lemma:

$$\vdash \quad inf \wedge \boxdot \neg\, stable\, K \quad \supset \quad \Box \neg\, stable\, K.$$

From this and lemma (23) we obtain the desired theorem:

$$\vdash \quad inf \wedge even(K) \wedge F1 \wedge F2 \quad \supset \quad \Box \neg\, stable\, K.$$

In [17] we deduce this without the use of $\diamondsuit$-fixpoints by introducing an auxiliary variable. The resulting proof has more steps.

6 Two Examples Involving Mutual Exclusion

We now analyze the safety and liveness properties of two systems for mutual exclusion. They have served as our main case studies during the development of the methods described in this work. Both are more complex than any of the examples discussed earlier and together require all of the technical machinery presented in the previous sections. The first example is the simpler of the two to describe but can sometimes deadlock. The second, modified system found later in Subsect. 6.3 remedies this problem but requires the introduction of extra variables which act as time stamps.

The first system, known as *GSys*, is composed of two parallel parts *G1* and *G2* shown in Figure 10. The combined specification together with initialization is as follows:

$$S1{=}C1{=}S2{=}C2{=}false \wedge G1 \wedge G2.$$

Here *G1* sets *S1* to *true* when requesting entry into *G1*'s critical region. Upon entry, *G1* sets *C1* to true and keeps it true until departure from the critical region. At this time *C1* and then *S1* are reset to *false* and remain so until the next attempt for entry is made. The usage by *G2* of variables *S2* and *C2* for controlling its critical region is analogous. We can verify the safety property that *C1* and *C2* are never both true at the same instant:

$$\vdash \quad S1{=}C1{=}S2{=}C2{=}false \wedge G1 \wedge G2 \quad \supset \quad \Box(\neg C1 \vee \neg C2). \tag{24}$$

Liveness of the form $\Box\diamondsuit C1 \wedge \Box\diamondsuit C2$ cannot be proved because *G1* and *G2* might both simultaneously attempt to enter their respective critical regions and deadlock. To some extent, one can deal with this by including the following assumption asserting that *G1* and *G2* do not initiate requests at the same time:

$$keep\big(\neg S1 \wedge \neg S2 \supset \bigcirc(\neg S1 \vee \neg S2)\big).$$

We instead show a weaker liveness property stating that *C1* and *C2* are always eventually false:

$$\vdash \quad S1{=}C1{=}S2{=}C2{=}false \wedge G1 \wedge G2 \quad \supset \quad \Box\diamondsuit(\neg C1 \wedge \neg C2).$$

$$
\begin{array}{lll}
G1 \stackrel{\text{def}}{\equiv} & \textit{forever do} \; (& \\
& \quad S1 \leftsquigarrow \textit{true} \wedge \textit{stable}\, C1; & \\
& \quad \textit{halt}\, \neg S2 \wedge \textit{stable}(S1, C1); & \\
& \quad C1 \leftsquigarrow \textit{true} \wedge \textit{stable}\, S1; & \\
& \quad C1 \leftsquigarrow \textit{false} \wedge \textit{stable}\, S1; & \\
& \quad S1 \leftsquigarrow \textit{false} \wedge \textit{stable}\, C1 & \\
&) &
\end{array}
\qquad
\begin{array}{lll}
G2 \stackrel{\text{def}}{\equiv} & \textit{forever do} \; (& \\
& \quad S2 \leftsquigarrow \textit{true} \wedge \textit{stable}\, C2; & \\
& \quad \textit{halt}\, \neg S1 \wedge \textit{stable}(S2, C2); & \\
& \quad C2 \leftsquigarrow \textit{true} \wedge \textit{stable}\, S2; & \\
& \quad C2 \leftsquigarrow \textit{false} \wedge \textit{stable}\, S2; & \\
& \quad S2 \leftsquigarrow \textit{false} \wedge \textit{stable}\, C2 & \\
&) &
\end{array}
$$

Figure 10: Simple mutual exclusion system

6.1 Safety of *GSys*

Here is an overview of the proof of safety:

(a) Compositionally show that whenever *G2* makes no entry request and *S2* is false, *C2* is also false:

$$\vdash \quad \neg S2 \wedge \neg C2 \wedge G2 \quad \supset \quad \Box(\neg S2 \supset \neg C2).$$

A theorem for the behavior imposed by *G1* on *S1* and *C1* is analogous but not needed for the proof of theorem (24).

(b) Without loss of generality, introduce an auxiliary boolean variable *P2* which is true whenever *G2* has been given *permission* to enter its critical region. More precisely, when *S2* and *P2* are both true, we know that *G2* has advanced passed its step containing the *halt* formula. Note that *C2* is not yet necessarily true. Associated with *P2* is an assumption *P2_As* which we later define.

(c) Prove that whenever *S2* is true but *P2* is false, *G2* forces *C2* to be false since *G2* has requested approval to enter its critical region but not yet received it:

$$\vdash \quad \neg S2 \wedge \neg C2 \wedge P2\_As \wedge G2 \quad \supset \quad \Box(\neg P2 \supset \neg C2).$$

This uses the assumption *P2_As* which characterizes *P2*.

(d) Import the result of (c) into *G1* to show that whenever *C1* is true, *C2* must be false.

(e) Combine everything together to complete the proof of the main safety theorem (24).

We now look at the proof's steps in more detail.

6.1.1 Step (a).

We desire to prove the following lemma:

$$\vdash \quad \neg S2 \wedge \neg C2 \wedge G2 \quad \supset \quad \Box(\neg S2 \supset \neg C2). \tag{25}$$

The consequent $\Box(\neg S2 \supset \neg C2)$ is not itself exportable (although it is importable) so we recast the lemma using the formula $\boxplus(\neg S2 \supset \neg C2)$ as the commitment *Co* instead. Figure 11 shows a proof outline for this.

6.1.2 Step (b).

We use the following general theorem about the *gets* construct (see its definition in Section 3 for introducing auxiliary variables:

$$\vdash \quad \exists A\text{: } (A \textit{ gets } e).$$

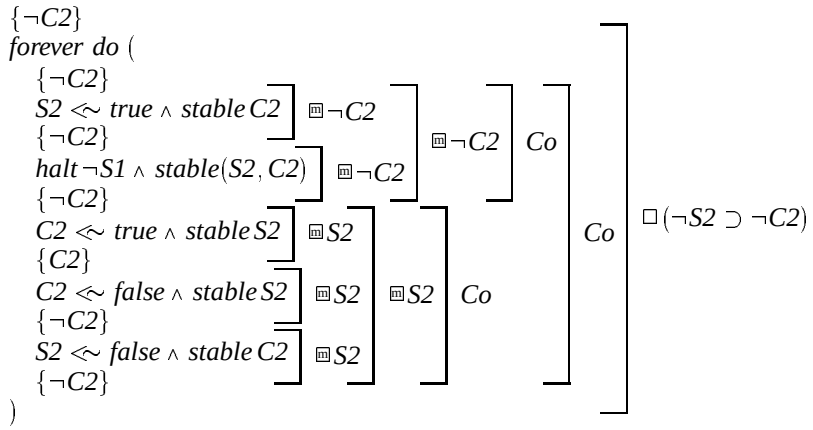

Figure 11: Proof outline for lemma (25).

Here the state variable A and the expression e have the same sort. We allow A to occur in e but not within the context of temporal operators. These restrictions ensure that A is not circularly defined. Here is an instance of this for the variable *P2*:

$$\vdash \quad \exists P2\colon \Big(P2 \textit{ gets } \big((P2 \wedge S2) \vee \bigcirc \neg S1\big)\Big). \tag{26}$$

This constructs an auxiliary boolean variable *P2* which monitors whether *G1* has permission to enter its critical region. In order to do this, *P2* keeps track about whether *S1* has ever been false since *S2* last become true. Between each pair of adjacent states, the next value of *P2* is determined based on the current value of *P2* and *S2* and the next value of *S1*. If *S2* is *false*, then the next value of *P2* equals the next value of $\neg S1$. Otherwise, it is the logical-or of the current value of *P2* and the next value of $\neg S1$. This ensures that after *G2* requests entry and *S2* becomes true, *P2* is also true iff *S1* has been *false* in at least in one state from then to now, inclusively. When *G2* has a successful request, both *S2* and *P2* are simultaneously true. It is possible to include information about the initial value of *P2* but this does not seem to be necessary for our proof.

In what follows, we use the *gets* subformula in an assumption about *P2*'s behavior called *P2_As*:

$$P2\_As \quad \stackrel{\text{def}}{\equiv} \quad P2 \textit{ gets } \big((P2 \wedge S2) \vee \bigcirc \neg S1\big).$$

This is importable because the *gets*-formula is a $\boxdot$-fixpoint since it is defined in terms of *keep*. In much of the remainder of the overall proof of safety, we refer to *P2* as a free variable. Towards the end, we eliminate it through the use of lemma (26) above.

6.1.3 Step (c).

We wish to show that whenever during *G2*'s operation *P2* is false, *C2* is also false:

$$\vdash \quad \neg S2 \wedge \neg C2 \wedge P2\_As \wedge G2 \quad \supset \quad \Box(\neg P2 \supset \neg C2). \tag{27}$$

The exportable commitment *Co* used here is $\boxed{m}(\neg P2 \supset \neg C2)$. Note that as in step (a), *Co* uses $\boxed{m}$ instead of $\Box$:

$$\vdash \quad \neg S2 \wedge \neg C2 \wedge P2\_As \wedge G2 \quad \supset \quad \boxed{m}(\neg P2 \supset \neg C2).$$

The proof outline for lemma (27) is given in Figure 12.

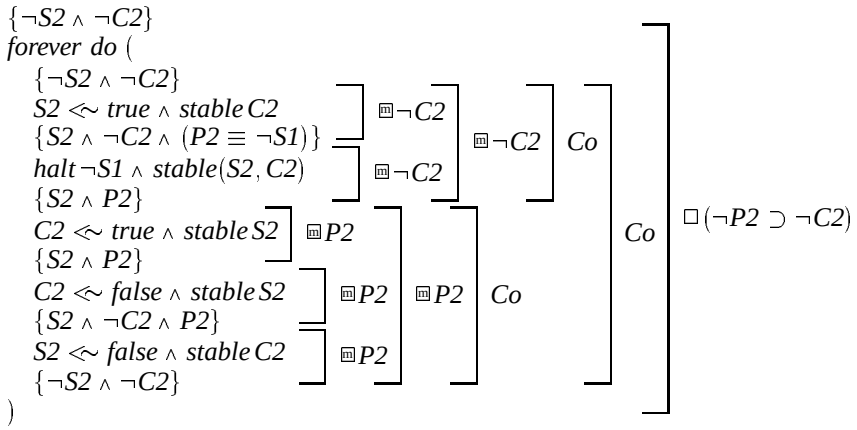

Figure 12: Proof outline for lemma (27).

6.1.4 Step (d).

Our goal in this step is to import into *G1* an assumption about the results regarding *G2* in steps (a), (b) and (c). Let us define the assumption *G1_As*:

$$G1\_As \quad \stackrel{\text{def}}{\equiv} \quad \boxed{a}\big((\neg S2 \vee \neg P2) \wedge \Box S1 \quad \supset \quad \Box\neg C2 \wedge \mathit{fin}(\neg S2 \vee \neg P2)\big).$$

It states that in any interval where initially no successful request is being made by *G2* (i.e., $\neg S2 \vee \neg P2$) *S1* is always true ($\Box S1$), then *G2* never enters its critical region and finishes the interval as it started without a successful request ($\mathit{fin}(\neg S2 \vee \neg P2)$) The assumption *G1_As* is obtained from *P2_As* and the two commitments exported from *G2*:

$$\vdash \quad P2\_As \wedge \Box(\neg S2 \supset \neg C2) \wedge \Box(\neg P2 \supset \neg C2) \quad \supset \quad G1\_As.$$

We then prove the following about *G1*:

$$\vdash \quad \neg S1 \wedge \neg C1 \wedge G1\_As \wedge G1 \quad \supset \quad \Box(\neg C1 \vee \neg C2). \tag{28}$$

See Figure 13 for a proof outline of this in which the commitment *Co* is $\boxed{m}(\neg C1 \vee \neg C2)$. Note that *G1_As* is only actually needed in the proof of the step where *C1* is assigned *true* since we trivially have $\boxed{m}\neg C1$ and hence $\boxed{m}(\neg C1 \vee \neg C2)$ true everywhere else.

6.1.5 Step (e).

We consolidate the previous lemmas into the desired safety theorem (24). Here are the main lemmas so far dealt with:

$$\vdash \quad \neg S2 \wedge \neg C2 \wedge G2 \quad \supset \quad \Box(\neg S2 \supset \neg C2),$$
$$\vdash \quad \exists P2{:}\, P2\_As,$$
$$\vdash \quad \neg S2 \wedge \neg C2 \wedge P2\_As \wedge G2 \quad \supset \quad \Box(\neg P2 \supset \neg C2),$$
$$\vdash \quad P2\_As \wedge \Box(\neg S2 \supset \neg C2) \wedge \Box(\neg P2 \supset \neg C2) \quad \supset \quad G1\_As,$$
$$\vdash \quad \neg S1 \wedge \neg C1 \wedge G1\_As \wedge G1 \quad \supset \quad \Box(\neg C1 \vee \neg C2).$$

We now combine most of these to obtain the following lemma for *G1* and *G2* together:

$$\vdash \quad S1{=}C1{=}S2{=}C2{=}\mathit{false} \wedge P2\_As \wedge G1 \wedge G2 \quad \supset \quad \Box(\neg C1 \vee \neg C2).$$

Let us now existentially quantify *P2* in *P2_As* since *P2* does not occur elsewhere in the implication:

$$\vdash \quad S1{=}C1{=}S2{=}C2{=}\mathit{false} \wedge (\exists P2{:}\, P2\_As) \wedge G1 \wedge G2 \quad \supset \quad \Box(\neg C1 \vee \neg C2).$$

We can then eliminate *P2* to obtain the final theorem:

$$\vdash \quad S1{=}C1{=}S2{=}C2{=}\mathit{false} \wedge G1\_As \wedge G1 \wedge G2 \quad \supset \quad \Box(\neg C1 \vee \neg C2).$$

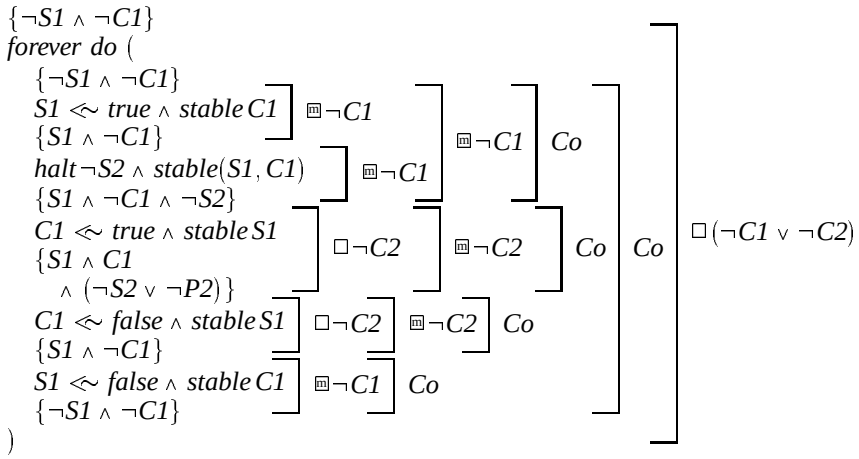

Figure 13: Proof outline for lemma (28).

6.2 Liveness of *GSys*

Recall that our liveness theorem for *GSys* is limited to ensuring that both *G1* and *G2* are always eventually outside their critical regions at the same time:

$$\vdash \quad S1{=}C1{=}S2{=}C2{=}false \wedge G1 \wedge G2 \quad \supset \quad \Box\Diamond(\neg C1 \wedge \neg C2). \tag{29}$$

To establish this, we first prove that *G2* is always eventually outside its critical region:

$$\vdash \quad \neg S2 \wedge \neg C2 \wedge G2 \quad \supset \quad \Box\Diamond\neg C2. \tag{30}$$

A proof outline for this is shown in Figure 14.

Lemma (25) mentioned earlier states that *G2* implies $\Box(\neg S2 \supset \neg C2)$. This is combined with lemma (30) to obtain an assumption:

$$\vdash \quad \Box\Diamond\neg C2 \wedge \Box(\neg S2 \supset \neg C2) \quad \supset \quad \boxed{a}(halt\,\neg S2 \supset \Box\Diamond\neg C2). \tag{31}$$

The following lemma imports this assumption into the process *G1*:

$$\vdash \quad S1{=}C1{=}S2{=}C2{=}false \wedge \boxed{a}(halt\,\neg S2 \supset \Box\Diamond\neg C2) \wedge G1 \quad \supset \quad \Box\Diamond(\neg C1 \wedge \neg C2). \tag{32}$$

In order to prove this, we use a slight variant of *G1* called *G1′* which includes an explicit marker *Mk* in its *forever* construct (*forever*$_{Mk}$) and we then deduce the following:

$$\begin{array}{l}\vdash \quad \neg S1 \wedge \neg C1 \wedge Mk \wedge \boxed{a}(halt\,\neg S2 \supset \Box\Diamond\neg C2) \wedge G1' \\ \qquad \supset \quad \Box\big(\Diamond(\neg C1 \wedge \neg C2) \vee \Diamond Mk\big) \wedge \Box\big(Mk \supset \Diamond(\neg C1 \wedge \neg C2)\big).\end{array} \tag{33}$$

A proof outline for this containing *G1′* is given in Figure 15. It uses for the $\diamond$-fixpoint *DA* the formula $\Diamond(\neg C1 \wedge \neg C2)$ and for *Co* the formula $\boxed{m}(DA \vee \Diamond Mk)$. The two $\Box$-formulas can be combined using conventional temporal reasoning:

$$\vdash \quad \Box\big(\Diamond(\neg C1 \wedge \neg C2) \vee \Diamond Mk\big) \wedge \Box\big(Mk \supset \Diamond(\neg C1 \wedge \neg C2)\big) \quad \supset \quad \Box\Diamond(\neg C1 \wedge \neg C2).$$

We then hide the marker *Mk* to obtain lemma (32). Theorem (29) can then be deduced from lemmas (30), (31) and (32).

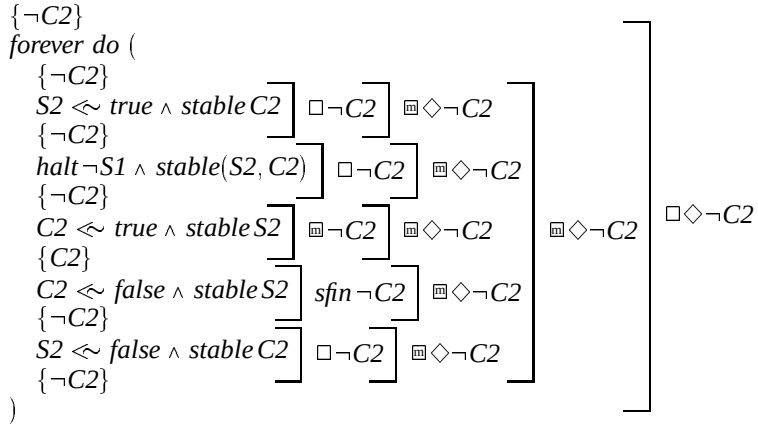

Figure 14: Proof outline for lemma (30).

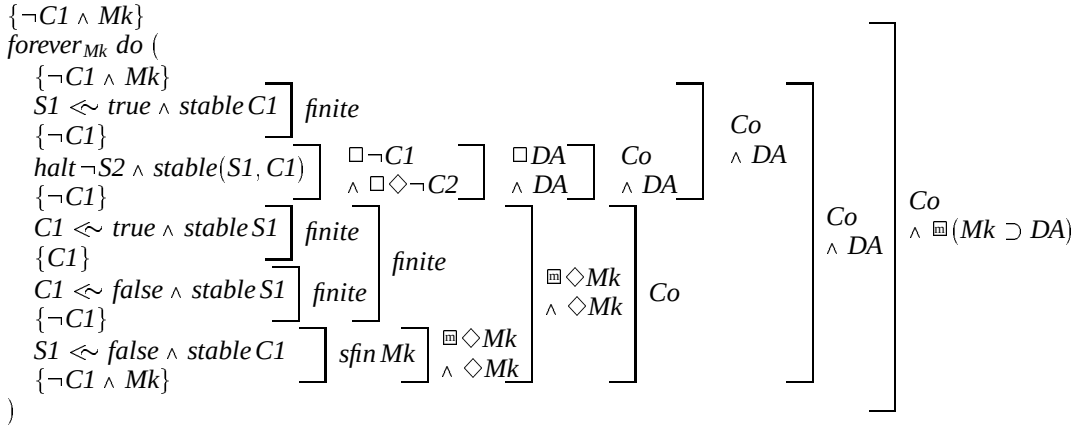

Figure 15: Proof outline for lemma (33).

$$
\begin{array}{lll}
H1 \stackrel{\text{def}}{\equiv} & \textit{forever do} \; (& \\
& \quad S1 \mathrel{\leftsquigarrow} \textit{true} \wedge \textit{stable}\, C1; & \\
& \quad \textit{halt}(\neg S2 \vee T1 \leq T2) & \\
& \qquad \wedge \textit{stable}(S1, C1); & \\
& \quad C1 \mathrel{\leftsquigarrow} \textit{true} \wedge \textit{stable}\, S1; & \\
& \quad C1 \mathrel{\leftsquigarrow} \textit{false} \wedge \textit{stable}\, S1; & \\
& \quad S1 \mathrel{\leftsquigarrow} \textit{false} \wedge \textit{stable}\, C1 & \\
&) &
\end{array}
\qquad
\begin{array}{lll}
H2 \stackrel{\text{def}}{\equiv} & \textit{forever do} \; (& \\
& \quad S2 \mathrel{\leftsquigarrow} \textit{true} \wedge \textit{stable}\, C2; & \\
& \quad \textit{halt}(\neg S1 \vee T2 < T1) & \\
& \qquad \wedge \textit{stable}(S2, C2); & \\
& \quad C2 \mathrel{\leftsquigarrow} \textit{true} \wedge \textit{stable}\, S2; & \\
& \quad C2 \mathrel{\leftsquigarrow} \textit{false} \wedge \textit{stable}\, S2; & \\
& \quad S2 \mathrel{\leftsquigarrow} \textit{false} \wedge \textit{stable}\, C2 & \\
&) &
\end{array}
$$

Figure 16: Mutual exclusion system with time stamps

6.3 Mutual Exclusion Using Time Stamps

We now turn to a variant of the mutual exclusion example called here *HSys* and based on a suggestion of Xu, Cau and Zedan [25] to maintain time stamps (referred to here as *T1* and *T2*) which record when the most recent changes to *S1* and *S2* have occurred. Figure 16 shows the parallel parts *H1* and *H2* which can both try to enter their respective critical regions. In the event that *H1* and *H2* both make requests (i.e., $S1 \wedge S2$), the one with the older request has priority. If there is a tie (i.e., $T1 = T2$), then *H1* has preference. This explains why *H1* uses $\leq$ in its *halt* construct whereas *H2* uses $<$. The maintenance of time stamps presumes the existence of a state variable we call *Timer* which always increases:

$$\textit{keep}(\textit{Timer} < \bigcirc \textit{Timer}).$$

The time stamp variables *T1* and *T2* are initially less than or equal to *Timer* and always record the last time *S1* and *S2* changed, respectively:

$$
\begin{array}{l}
T1 \leq \textit{Timer} \wedge \big(T1 \textit{ gets } (\textit{if stable}\, S1 \textit{ then } T1 \textit{ else } \bigcirc \textit{Timer})\big), \\
T2 \leq \textit{Timer} \wedge \big(T2 \textit{ gets } (\textit{if stable}\, S2 \textit{ then } T2 \textit{ else } \bigcirc \textit{Timer})\big).
\end{array}
$$

Note that a time stamp equals the last time immediately *after* a change (i.e., $\bigcirc \textit{Timer}$).

The formulas *H1_init*, *H2_init* and *HSys_init* denote the respective initial conditions for *H1*, *H2* and *HSys*:

$$
\begin{array}{ll}
\textit{H1\_init} & \stackrel{\text{def}}{\equiv} \neg S1 \wedge \neg C1 \wedge T1 \leq \textit{Timer}, \\
\textit{H2\_init} & \stackrel{\text{def}}{\equiv} \neg S2 \wedge \neg C2 \wedge T2 \leq \textit{Timer}, \\
\textit{HSys\_init} & \stackrel{\text{def}}{\equiv} \textit{H1\_init} \wedge \textit{H2\_init}.
\end{array}
$$

Below are definitions for assumptions about the timer and time stamps:

$$
\begin{array}{ll}
\textit{Timer\_As} & \stackrel{\text{def}}{\equiv} \textit{keep}(\textit{Timer} < \bigcirc \textit{Timer}) \\
\textit{T1\_As} & \stackrel{\text{def}}{\equiv} T1 \leq \textit{Timer} \wedge \textit{Timer\_As} \wedge \big(T1 \textit{ gets } (\textit{if stable}\, S1 \textit{ then } T1 \textit{ else } \bigcirc \textit{Timer})\big), \\
\textit{T2\_As} & \stackrel{\text{def}}{\equiv} T2 \leq \textit{Timer} \wedge \textit{Timer\_As} \wedge \big(T2 \textit{ gets } (\textit{if stable}\, S2 \textit{ then } T2 \textit{ else } \bigcirc \textit{Timer})\big), \\
\textit{T12\_As} & \stackrel{\text{def}}{\equiv} \textit{T1\_As} \wedge \textit{T2\_As}.
\end{array}
$$

The lemmas now given assist for general reasoning about the time stamps *T1* and *T2*. In particular, they establish that *T1_As*, *T2_As* and *T12_As* are indeed $\boxdot$-fixpoints and can be used as assumptions.

$$
\begin{array}{ll}
\vdash & \textit{T1\_As} \supset \Box T1 \leq \textit{Timer}, \\
\vdash & \textit{T2\_As} \supset \Box T2 \leq \textit{Timer}, \\
\vdash & \textit{T1\_As} \equiv \boxdot \textit{T1\_As}, \\
\vdash & \textit{T2\_As} \equiv \boxdot \textit{T2\_As}, \\
\vdash & \textit{T12\_As} \equiv \boxdot \textit{T12\_As}.
\end{array}
$$

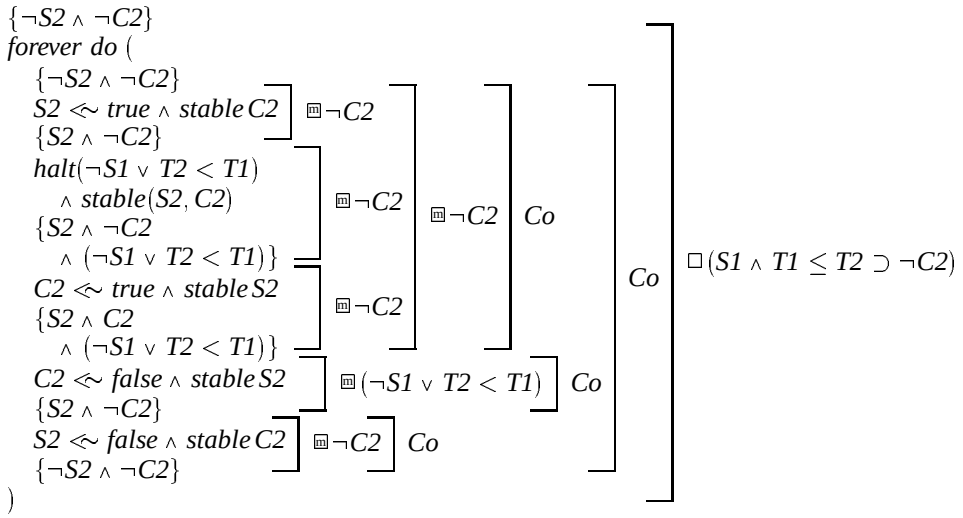

Figure 17: Proof outline for lemma (37).

Here are theorems for safety and liveness:

$$\vdash \quad \mathit{HSys\_init} \wedge \mathit{T12\_As} \wedge \mathit{H1} \wedge \mathit{H2} \quad \supset \quad \Box(\neg \mathit{C1} \vee \neg \mathit{C2}), \tag{34}$$

$$\vdash \quad \mathit{HSys\_init} \wedge \mathit{T12\_As} \wedge \mathit{H1} \wedge \mathit{H2} \quad \supset \quad \Box\Diamond \mathit{C1} \wedge \Box\Diamond \mathit{C2}. \tag{35}$$

The proof of each is considered separately.

6.4 Safety for *HSys*

We first consider proving the theorem about safety. This does not require introducing a new auxiliary variable such as *P2* since the time stamps turn out to be sufficient. An assumption imported into *H1* and called *H1_As* is used:

$$\mathit{H1\_As} \quad \stackrel{\text{def}}{\equiv} \quad \boxdot\big((\neg \mathit{S2} \vee \mathit{T1} \leq \mathit{T2}) \wedge \Box \mathit{S1} \ \supset \ \Box\neg \mathit{C2} \wedge \mathit{fin}(\neg \mathit{S2} \vee \mathit{T1} \leq \mathit{T2})\big).$$

It is analogous to *G1_As* and states that in any interval where *H1* maintains an active request ($\Box S1$) and initially *H2* is not requesting ($\neg S2$) or has a request which is not older than *H1*'s ($T1 \leq T2$) then *H2* never enters its critical region ($\Box\neg C2$) and ends up with the same general request status ($\mathit{fin}(\neg S2 \vee T1 \leq T2)$).

Below are the major lemmas needed:

$$\vdash \quad \neg \mathit{S2} \wedge \neg \mathit{C2} \wedge \mathit{H2} \quad \supset \quad \Box(\neg \mathit{S2} \supset \neg \mathit{C2}), \tag{36}$$

$$\vdash \quad \mathit{H2\_init} \wedge \mathit{T2\_As} \wedge \mathit{H2} \quad \supset \quad \Box(\mathit{S1} \wedge \mathit{T1} \leq \mathit{T2} \supset \neg \mathit{C2}), \tag{37}$$

$$\vdash \quad \mathit{T12\_As} \wedge \Box(\neg \mathit{S2} \supset \neg \mathit{C2}) \wedge \Box(\mathit{S1} \wedge \mathit{T1} \leq \mathit{T2} \supset \neg \mathit{C2}) \ \supset \ \mathit{H1\_As}, \tag{38}$$

$$\vdash \quad \mathit{H1\_init} \wedge \mathit{T1\_As} \wedge \mathit{H1\_As} \wedge \mathit{H1} \quad \supset \quad \Box(\neg \mathit{C1} \vee \neg \mathit{C2}). \tag{39}$$

The proof of lemma (36) is similar to that of lemma (25) and omitted. The proof outline for lemma (37) is shown in Figure 17. It uses for the exportable commitment *Co* the formula $\boxplus(S1 \wedge T1 \leq T2 \supset \neg C2)$.

Lemma (38)'s proof involves straightforward temporal reasoning and is not given. See Figure 18 for a proof of lemma (39). The exportable commitment *Co* used by it is $\boxplus(\neg C1 \vee \neg C2)$. From these we can deduce theorem (34) about safety.

6.5 Liveness for *HSys*

The proof of liveness theorem (35) for *HSys* involves four key lemmas about the processes *H1* and *H2*:

$$\vdash \quad \mathit{H2\_init} \wedge \mathit{T12\_As} \wedge \mathit{H2} \ \supset \ \Box\Diamond \mathit{T1} \leq \mathit{T2}, \tag{40}$$

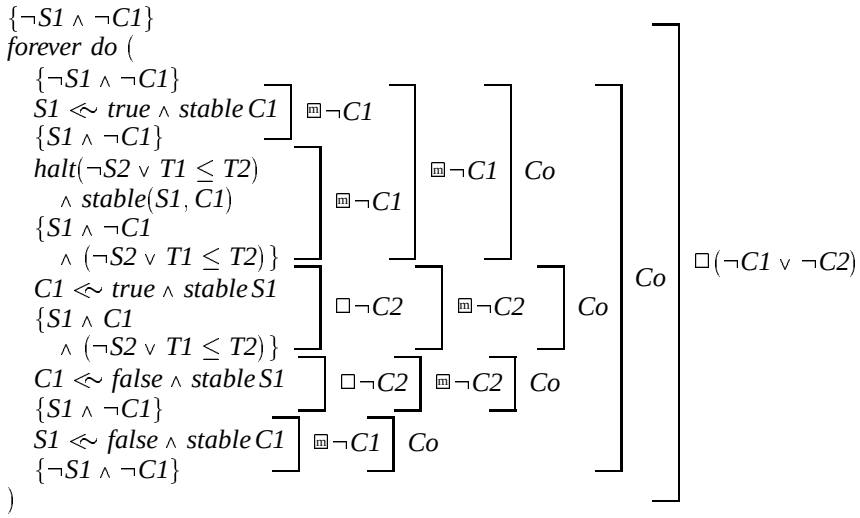

Figure 18: Proof outline for lemma (39).

$$\vdash \quad H1\_init \wedge \big(T12\_As \wedge \boxdot(halt\, T1 \leq T2 \supset finite)\big) \wedge H1 \quad \supset \quad \Box\Diamond C1, \tag{41}$$
$$\vdash \quad H1\_init \wedge H1 \ \supset \ \Box(C1 \supset \Diamond\neg S1), \tag{42}$$
$$\vdash \quad H2\_init \wedge \big(T12\_As \wedge \boxdot(halt\neg S1 \supset finite)\big) \wedge H2 \quad \supset \quad \Box\Diamond C2. \tag{43}$$

The first of these states that *H1*'s request is at least as old as *H2*'s infinitely often. It ensures that *H2* is not making too many requests without giving *H1* a chance. The second lemma uses the the first one's results (converted to an importable assumption) to establish that *H1* enters its critical region infinitely often. The consequent of the first lemma is converted to a $\boxdot$-fixpoint which is used as part of the second lemma's assumption. Recall that $\boxdot$-fixpoints are closed under conjunction so a number of them can be imported together. The third lemma states that when *H1* enters its critical region, it also eventually leaves it and resets the request variable *S1* to *false.* The consequents of the second and third lemmas are combined to form an assumption $\boxdot(halt\neg S1 \supset finite)$ specifying that *H1* is never infinitely long requesting entry without success. This assumption is used in the fourth lemma to show that *H2* enters its critical region infinitely often.

A proof outline of lemma (40) is in Figure 19 and uses *DA* to represent the formula $\Diamond(T1 \leq T2)$. Figure 20 shows a proof for lemma (42) with *Co* being the exportable commitment $\boxdot(C1 \supset \Diamond\neg S1)$. We also include proof outlines for versions of the second and fourth lemmas with explicit markers. They use slight variants of *H1* and *H2* called $H1'$ and $H2'$, respectively, containing loops referencing markers (i.e., $forever_{Mk}$ instead *forever*). The lemmas for $H1'$ and $H2'$ are now shown:

$$\vdash \quad H1\_init \wedge Mk \wedge \big(T12\_As \wedge \boxdot(halt\, T1 \leq T2 \supset finite)\big) \wedge H1' \quad \supset \quad \Box\Diamond(C1 \vee Mk) \wedge \Box(Mk \supset \Diamond C1), \tag{44}$$
$$\vdash \quad H2\_init \wedge Mk \wedge \big(T12\_As \wedge \boxdot(halt\neg S1 \supset finite)\big) \wedge H2' \quad \supset \quad \Box\Diamond(C2 \vee Mk) \wedge \Box(Mk \supset \Diamond C2). \tag{45}$$

The proof outline for lemma (44) is in Figure 21 and uses *Co* to represent the formula $\boxdot(\Diamond C1 \vee \Diamond Mk)$ and *DA* to represent $\Diamond C1$. Figure 22 contains the proof outline for lemma (45). It uses *Co* to stand for $\boxdot\Diamond(C2 \vee Mk)$ and *DA* to stand for $\Diamond C2$. The two corresponding lemmas (41) and (43) for *H1* and *H2*, respectively, are obtained from these with the help of the following lemmas and existential elimination of existential quantifiers:

$$\vdash \quad \Box\Diamond(C1 \vee Mk) \wedge \Box(Mk \supset \Diamond C1) \quad \supset \quad \Box\Diamond C1,$$
$$\vdash \quad \Box\Diamond(C2 \vee Mk) \wedge \Box(Mk \supset \Diamond C2) \quad \supset \quad \Box\Diamond C2.$$

Here are some simple temporal lemmas for converting consequents of some of the lemmas into assumptions imported by others.

$$\vdash \quad \Box\Diamond T1 \leq T2 \ \supset \ \boxdot(halt\, T1 \leq T2 \supset finite),$$

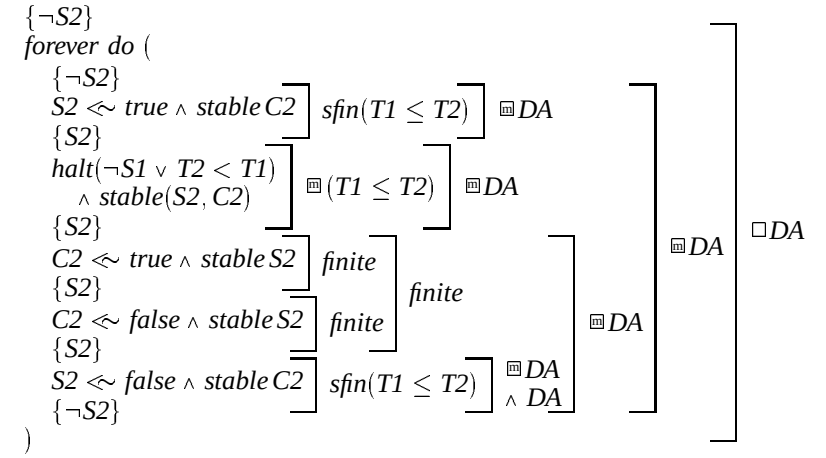

Figure 19: Proof outline for lemma (40).

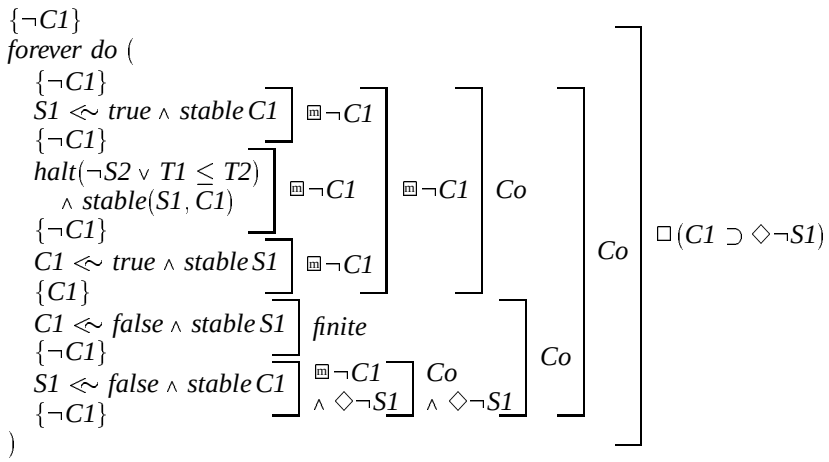

Figure 20: Proof outline for lemma (42).

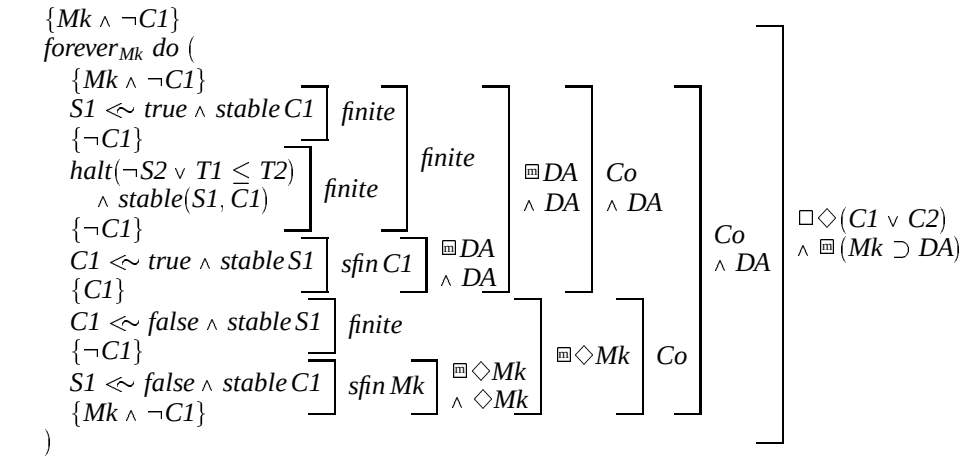

Figure 21: Proof outline for lemma (44).

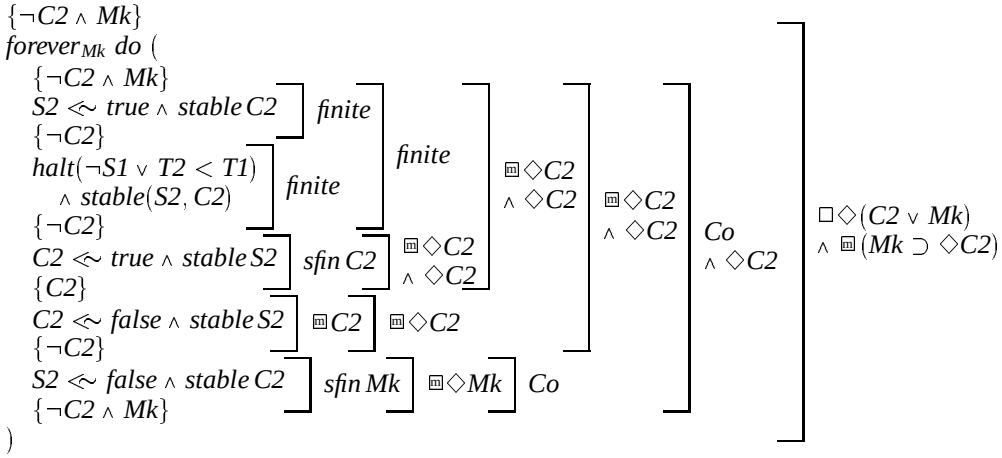

Figure 22: Proof outline for lemma (45).

$\vdash \quad \Box\Diamond C1 \wedge \Box(C1 \supset \Diamond\neg S1) \quad \supset \quad \Box\Diamond\neg S1,$
$\vdash \quad \Box\Diamond\neg S1 \quad \supset \quad \boxdot(halt\,\neg S1 \supset finite).$

7 Conclusion

We have presented some of our experience with using temporal fixpoints for compositional reasoning about safety and liveness. Our plans include applying these and other methods to the formal specification and analysis of various conceptual layers of the EP/3 multithreaded computer [1] being built by Dr. J. N. Coleman at the University of Newcastle.

Acknowledgements

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

References

[1] Coleman JN. A high speed dataflow processing element and its performance compared to a von Neumann mainframe. In: IEEE 7th international parallel processing symposium. IEEE Computer Society Press, Los Alamitos, California, USA, 1993, pp 24–33

[2] 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, pp 36–43

[3] Francez N, Pnueli A. A proof method for cyclic programs. Acta Inf 1978; 9:133–157

[4] 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). Springer-Verlag, Heidelberg, 1983, pp 278-291 (Lecture Notes in Computer Science No. 154)

[5] Hoare CAR. An axiomatic basis for computer programming. Comm ACM 1969; 12:576–580,583

[6] Jones CB. Specification and design of (parallel) programs. In: Mason REA (ed) Proceedings of information processing '83. North Holland Publishing Co, Amsterdam, 1983, pp 321–332

[7] Jonsson B, Tsay Y-K. Assumption/guarantee specifications in linear-time temporal logic. In: Mosses PD et al (eds) Proceedings of TAPSOFT '95: Theory and practice of software development. Springer-Verlag, Heidelberg, 1995, pp 262–276, (Lecture Notes in Computer Science No. 915)

[8] Kleene SC. Mathematical logic. John Wiley & Sons, Inc., New York, 1967

[9] Kono S. A combination of clausal and non clausal temporal logic programs. In: Fisher M, Owens R (eds) Executable modal and temporal logics. Springer-Verlag, Heidelberg, 1995, pp 40–57 (Lecture Notes in Computer Science No. 897)

[10] Kröger F. Temporal logic of programs. Springer-Verlag, Berlin, 1987

[11] Lamport L. 'Sometimes' is sometimes 'not never': on the temporal logic of programs. In: Proceedings of the 7th ACM Symposium on principles of programming languages. ACM Press, New York, 1980, pp 174–185

[12] Manna Z. Verification of sequential programs: temporal axiomatization. In: Broy M, Schmidt G (eds), Theoretical Foundations of Programming Methodology. D. Reidel Publishing Co, 1982, pp 53–102

[13] Moszkowski B. Reasoning about digital circuits. PhD thesis, Stanford University, Stanford, California, 1983

[14] Moszkowski B. A temporal logic for multilevel reasoning about hardware. IEEE Computer 1985; 18:10–19

[15] Moszkowski B. Executing temporal logic programs. Cambridge University Press, Cambridge, England, 1986

[16] Moszkowski B. Some very compositional temporal properties. In: E.-R. Olderog (ed) Programming concepts, methods and calculi. IFIP Transactions, Vol. A-56, North-Holland, 1994, pp 307–326

[17] 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, pp 238–245

[18] Owicki S. Axiomatic proof techniques for parallel programs. PhD thesis, Cornell University, Ithaca, New York, 1975

[19] Owicki S and Gries D. An axiomatic proof technique for parallel programs: part I. Acta Inf 1976; 6:319–340

[20] Paech B. Gentzen-systems for propositional temporal logics. In: Börger E et al (eds) Proceedings of the 2nd workshop on computer science logic. Springer-Verlag, Heidelberg, 1988, pp 240–253 (Lecture Notes in Computer Science No. 385)

[21] Pandya PK, Joseph M. P-A logic: A compositional proof system for distributed programs. Distributed Computing 1991; 5:37–54

[22] 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, pp 306–314

[23] Stølen K. Proving total correctness with respect to a fair (shared-state) parallel language. In: Proceedings of the 5th BCS-FACS refinement workshop. Springer-Verlag, London, 1992, pp 320–341

[24] Xu Q, Cau A, Collette P. On unifying assumption-commitment style proof rules for concurrency. In: Concur'94. Springer-Verlag, 1994, pp 267–282 (Lecture Notes in Computer Science No. 836)

[25] Xu Q, Cau A, Zedan H. Semantics and verification of infinite temporal agent model programs using interval temporal logic. In preparation

[26] Xu Q, de Roever W-P, He J. Rely-guarantee method for verifying shared variable concurrent programs. Technical Report 9502, Institute of Computer Science II, Kiel University, Kiel, Germany, 1995

[27] Xu Q, He J. A theory of state-based parallel programming: part 1. In: Morris J (ed) Proceedings of the 4th BCS-FACS refinement workshop. Cambridge, UK, Springer-Verlag, London, 1991