Published in *Proc. First IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, IEEE Computer Society Press, 1995, pp. 238–245, (Fort Lauderdale, Florida, November 6–10, 1995).
Received award for best paper at the conference in formal methods track.

Compositional Reasoning about Projected and Infinite Time

Ben Moszkowski[1]
Department of Electrical and Electronic Engineering
University of Newcastle upon Tyne
Newcastle NE1 7RU
Great Britain
Internet: Ben.Moszkowski@ncl.ac.uk

Abstract

Modularity is of fundamental importance in computer science. The need for a formal theory of modularity in the design and maintenance of large systems is especially pronounced. In recent work [1] on Interval Temporal Logic (ITL) [2, 3, 4] we gave an axiom system in which proofs of sequential and parallel systems can be decomposed into proofs for the syntactic subcomponents. This provides a precise framework for describing and generalizing the insights of Francez and Pnueli [5] and Jones [6] for modular reasoning about concurrency using what are often called assumptions *and* commitments. *It combines the benefits of these ideas with temporal logic. We now show that such techniques can be used to analyze temporal projection operators developed by us for describing systems with multiple time granularities. In addition, we consider how to compositionally reason in ITL about the absence of deadlock in systems running for infinite time. This demonstrates that our generalization of Jones' techniques for assumptions and commitments handles not only safety properties but also liveness ones [7].*

1 Introduction

One of the best known ways to modularly specify and reason about a concurrent system is through the use of *assumptions* and *commitments*. This can simplify the specification and verification of a dynamic system built out of interacting sequential and parallel components. Let us consider assumptions and commitments from the standpoint of temporal logic. Suppose we have a system described by a temporal formula Sys. System interaction with the world can be described by one or more formulas of the form

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

Here w and w' are formulas about a single state whereas As, Sys and Co are arbitrary formulas which can contain temporal constructs. The implication says that if w is true in the first state and As is true for the period when Sys is active, then Co also holds and that w' is true in the final state. We formally define *fin* and related constructs later. The formula As is often called an *assumption* and Co a *commitment.* Francez and Pnueli [5] called these *interface predicates* and were perhaps the first to study how they could relate parts of a parallel system. If we can represent the parallel behavior of two systems Sys and Sys' as the conjunction $Sys \wedge Sys'$, much of the interaction of the assumptions and commitments of Sys and Sys' can be reasoned about using conventional logical deduction.

Jones [6] used assumptions and commitments (called by him *rely-conditions* and *guarantee-conditions*, respectively) in proof rules for a generalized Hoare logic [8] in order to decompose the specification of a sequential system communicating via shared variables with its environment. Here is a proof rule of this kind expressed in temporal logic:

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

The rule uses the operator ";" (*chop*) of Interval Temporal Logic (ITL) [2, 3, 4] to combine the formulas Sys and Sys' in series.

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

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

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. Similar rules are possible for *if*, *while* and other constructs. Rules (1) and (2) are not in general sound since they do not work for arbitrary As's and Co's. For instance, try the rules with the assumption "K's

[1]Supported by EPSRC Research Grant GR/K25922

values in the initial and final states are equal." Jones proposed obtaining soundness by restricting the As's and Co's to properties which hold for all pairs of adjacent states in the computation. An example is "*K's value remains the same or increases between each pair of states.*" In ITL we can characterize Jones' approach as formulas of the form *keep* S for some arbitrary subformula S. A formula *keep* S is defined to be true on an interval iff S is true on every subinterval consisting of exactly two (adjacent) states. These are the smallest intervals in which change can occur since we model time as being discrete. Once a *keep*-formula has been established as a commitment guaranteed by one activity, it can be used as an assumption for other activities operating in parallel.

In [1] we showed that the notion of assumptions and commitments fits naturally within the ITL formalism. In addition, assumptions and commitments can be more general than those of Jones while still preserving the soundness of rules (1) and (2). An assumption As is suitable if it can be *imported* into the scope of *chop* and *chop-star*. We can achieve this be requiring that if As is true for an interval, it is also true in all of the interval's subintervals. This can be formally expressed in ITL as follows:

$$\vdash As \supset \boxdot As.$$

Here $\boxdot As$ is defined to be true for an interval iff As is true for every subinterval (including the interval itself). An example is "*K always equals 1.*" The following show how an assumption imports into *chop* and *chop-star* and can be used in proofs of rules (1) and (2):

$\vdash As \wedge (Sys; Sys') \supset (As \wedge Sys); (As \wedge Sys')$

$\vdash As \wedge Sys^* \supset (As \wedge Sys)^*.$

A commitment Co can be *exported* from *chop* and *chop-star* if the following is provable:

$$\vdash Co^* \supset Co.$$

Here Co^* denotes zero or more sequential iterations of the ITL formula Co. An example is "*K's values in the initial and final states are equal.*" Any such commitment can be exported from *chop* and *chop-star* as follows:

$\vdash (Sys \wedge Co); (Sys' \wedge Co) \supset (Sys; Sys') \wedge Co$

$\vdash (Sys \wedge Co)^* \supset Sys^* \wedge Co.$

We can obtain closed forms which syntactically characterize these desired qualities. The set of importable assumptions can be proved to be those formulas expressible in the form $\boxdot S$, for some arbitrary ITL subformula S. Similarly, the set of exportable commitments are those of the form S^*.

The *keep* formulas which characterize Jones' original approach can be proved to be exactly the intersection of importable assumptions and exportable commitments. We presented an ITL proof system developed for reasoning about compositionality. It can be used to formally deduce rules (1) and (2) and many of the other details described here for the kinds of assumptions and commitments we have mentioned. In addition, we applied it to reasoning about concurrent systems with shared writable variables, asynchronous communication and timing constraints.

Earlier work by us [2, 3, 22] proposed some temporal projection operators for mapping between intervals representing different granularities of time. These operators can be used to describe digital systems and other processes with multiple clock rates. We present here an ITL proof system which facilitates compositional reasoning about projection. This permits us to prove ITL theorems involving discrete-time temporal operators such as $\bigcirc$ (*next*) in a modular way and without committing ourselves to any particular rate of time passage. A formal notion of importability and exportability for projection is discussed and used in an associated compositional proof rule.

The presentation in [1] briefly showed how to modify the ITL syntax, semantics and proof system to handle infinite time. Earlier research by Rosner and Pnueli [9] and Paech [10] also considered infinite intervals. One of our contributions in the present work is to demonstrate that the compositional proof techniques described above can deal with liveness properties and not just safety ones [7]. We use a $\boxdot$-assumption, which is not permitted in Jones's original formulation. Stølen [11] deals with liveness by adding a *wait-condition* to Jones' approach. See also Xu and He [12], and Xu, Cau and Collette [13] and a survey by Xu, de Roever and He [14]. Jonsson and Tsay [15] use linear time temporal temporal logic with past time in assumptions and commitments involving liveness.

2 Preliminaries

We now describe Interval Temporal Logic. The presentation is rather brief and the reader should refer to references such as [3, 4] for more details. ITL is a linear-time temporal logic with a discrete model of time. An interval is in general a finite, nonempty sequence of states, each mapping 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.

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 interval's start. Thus, the formula $L = K+1$ is true on an interval σ iff the L's value in σ's first state is one more that K's value in that state.

Conventional linear-time temporal logic operators					
$\bigcirc S$	$\stackrel{\text{def}}{\equiv} skip; S$	Next	$\Diamond S$	$\stackrel{\text{def}}{\equiv} true; S$	Sometimes
$\textcircled{w} S$	$\stackrel{\text{def}}{\equiv} \neg \bigcirc \neg S$	Weak next	$\Box S$	$\stackrel{\text{def}}{\equiv} \neg \Diamond \neg S$	Always
Some other important operators					
more	$\stackrel{\text{def}}{\equiv} \bigcirc true$	Nonempty interval	*fin* S	$\stackrel{\text{def}}{\equiv} \Box(empty \supset S)$	Final state
empty	$\stackrel{\text{def}}{\equiv} \neg more$	Empty interval	*halt* S	$\stackrel{\text{def}}{\equiv} \Box(S \equiv empty)$	Exactly final
More interval-oriented operators					
$\textcircled{a} S$	$\stackrel{\text{def}}{\equiv} true; S; true$	Some subinterval	$\textcircled{1} S$	$\stackrel{\text{def}}{\equiv} S; true$	Some initial subinterval
$\boxed{a} S$	$\stackrel{\text{def}}{\equiv} \neg \textcircled{a} \neg S$	All subintervals	$\boxed{1} S$	$\stackrel{\text{def}}{\equiv} \neg \textcircled{1} \neg S$	All initial subintervals
keep S	$\stackrel{\text{def}}{\equiv} \boxed{a}(skip \supset S)$	All *unit* subintervals	*keepnow* S	$\stackrel{\text{def}}{\equiv} \textcircled{1}(skip \wedge S)$	First unit

Figure 1: Some definable ITL operators

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

$$skip \qquad S; T \qquad S^*.$$

The formula *skip* has no operands and is true on an interval iff the interval has exactly two states. Both *chop* and *chop-star* permit evaluation within various subintervals. A formula $S; T$ is true on an interval iff the interval can be chopped into two parts sharing a single state and in which S is true on the left part and T is true on the right part. Thus, the formula $skip; (L = K + 1)$ is true on an interval σ iff σ has at least two states and $L = K+1$ is true in the second one. A formula S^* is true on an interval iff the interval can be chopped into zero or more sequential parts and S is true on each. An empty interval (one having exactly one state) trivially satisfies any formula of the form S^* (including $false^*$). We generally use w, w' and so forth to denote *state formulas* with no temporal operators in them. Expressions are denoted by e, e' and so on. Figure 1 shows some useful operators definable in ITL.

In [1] we made use of the conventional logical notion of definite descriptions as a way to give 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).

The state variables A and B as well as the static variable a range over arbitrary values whereas the variables K, L and i range over the natural numbers. Expressions and formulas with no temporal operators and only static variables are also called static.

2.1 Other first-order constructs

The next few definitions provide a way of observing the values of one or two expressions at various points in time. These constructs are used extensively in the parallel systems described later. As before, e and e' are arbitrary expressions. In addition, a is any static variable of the same data domain as e and not occurring freely in e itself:

$$\begin{aligned}
e \leftarrow e' &\stackrel{\text{def}}{\equiv} (fin\ e) = e' \\
e\ gets\ e' &\stackrel{\text{def}}{\equiv} keep\ (e \leftarrow e') \\
goodcounter\ e &\stackrel{\text{def}}{\equiv} keep\ (e \leq \bigcirc e) \\
goodindex\ e &\stackrel{\text{def}}{\equiv} keep\ (e \leq \bigcirc e \leq e + 1) \\
padded\ e &\stackrel{\text{def}}{\equiv} \exists a{:}\, keep\ (e = a) \\
e \lessapprox e' &\stackrel{\text{def}}{\equiv} (e \leftarrow e') \wedge padded\ e \\
stable\ e &\stackrel{\text{def}}{\equiv} \exists a{:}\, \Box(e = a)
\end{aligned}$$

The operator *goodcounter* checks that an expression continues to remain stable or increase. The operator *goodindex* tests that an expression remains unchanged or increases by exactly 1 over every unit subinterval. The operator *padded* ensures that the expression's value remains unchanged except possibly in the interval's last state. This is used in the definition of the operator $\lessapprox$ to describe a *padded temporal assignment* which can be used for synchronous assignments in parallel systems.

3 A practical proof system

We now present a very powerful and practical compositional proof system for ITL. Our experience in rigorously developing hundreds of proofs has helped us refine the axioms and convinced us that they are sufficient for a very wide range of purposes.

The axioms and inference rules shown below deal mainly with *chop*, *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. For example, it enables us to deduce theorems about the left side of *chop* as well as the operators $\boxdot$ and $\boxminus$ (all *initial* subintervals). This is exceedingly important for the kinds of proofs we do. In contrast, most temporal logics cannot handle initial subintervals and even other proof systems for ITL largely neglect them.

Rosner and Pnueli [9] and Paech [10] give propositional proof systems for ITL with infinite intervals and prove completeness. Kono [16] has developed a complete propositional ITL proof system for finite time. Our finite-time 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 $\boxminus$, *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 $\vdash (S;T);U \equiv S;(T;U)$
$\supset (S;T) \supset (S';T')$
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 $\vdash empty;S \equiv S$
P6 $\vdash S;empty \equiv S$
P7 $\vdash w \supset \boxminus w$
P8 $\vdash \boxminus(S \supset S') \wedge \Box(T \supset T')$
P9 $\vdash \bigcirc S \supset \neg\bigcirc\neg S$
P10 $\vdash keepnow\ S \supset \neg\, keepnow\ \neg S$
P11 $\vdash S \wedge \Box(S \supset \circledcirc S) \supset \Box S$
P12 $\vdash S^* \equiv empty \vee (S \wedge more);S^*$

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

In [1] we overview a proof of completeness for the axiom system and give a first-order ITL proof system which is not essential for the discussion here. Dutertre [17] gives a complete first-order ITL proof system but with a nonstandard semantics of intervals. Hale and He [18] and Hale [19] look at reasoning about ITL in the HOL proof system. Skakkebæk and Shankar [20] discuss using the PVS proof system for verifying theorems of the Duration Calculus [21], a variant of ITL for continuous time.

4 Temporal projection

Various aspects of digital circuits, automata and other dynamic systems can be naturally specified using discrete intervals in which each state corresponds to one clock cycle or atomic step. A problem arises when we attempt to relate specifications which assume different granularities of time. In earlier work on ITL [2, 3] we proposed a temporal projection operator for mapping between intervals with varying degrees of atomicity. Subsequently in [22] we gave a more general definition of the operator and showed how it could be used as a programming language construct. We now examine this operator and others based on it. A proof system is then given for compositional reasoning.

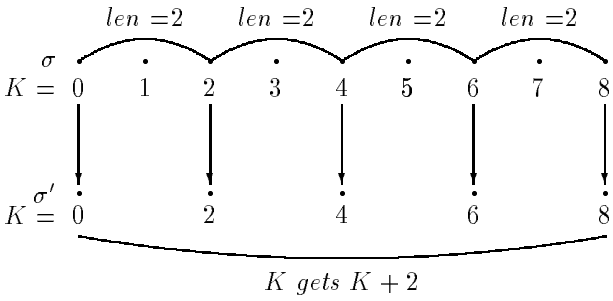

Figure 2: Example of temporal projection

Roughly speaking, the formula $S \triangle T$ is defined to be true on an interval σ iff two conditions are met. First, the formula T must be true on some interval σ' obtained by projecting some states from σ. Second, the formula S must be true on each of the subintervals of σ bridging the gaps between the projected states. Before giving a formal definition of $\triangle$, let us consider figure 2 which illustrates a sample projection. The example depicts an interval σ in which the state variable K increases from 0 to 8 by steps of 1. Now σ satisfies the ITL formula shown below:

$$(len = 2) \triangle (K\ gets\ K + 2).$$

Here $len = 2$ is true of intervals with length 2 (i. e., 3 states) and $K\ gets\ K + 2$ is true iff K increases by 2 from state to state. We can construct a projected interval σ' satisfying $K\ gets\ K + 2$. The gaps in the original interval σ are bridged by the formula $len = 2$.

Below is a more precise definition of $\triangle$:

$\mathcal{M}_\sigma[\![S \triangle T]\!] = true$ iff
for some $n \geq 0$, σ' and $l_0, \ldots, l_n$:
$0 = l_0 < \cdots < l_n = |\sigma|$,
and for each $i < n$, $\mathcal{M}_{\sigma_{l_i:l_{i+1}}}[\![S]\!] = true$,
and $\mathcal{M}_{\sigma'}[\![T]\!] = true$,
where $|\sigma'| = n$ and for all $i \leq n$, $\sigma'_i = \sigma_{l_i}$.

Here $\sigma_{l_i:l_{i+1}}$ denotes the subinterval having $l_{i+1} - l_i + 1$ consecutive states starting with state σ_{l_i}, where σ_0 is σ's initial state. Note that n can equal 0 and that σ' always contains exactly $n + 1$ states.

We are now in a position to define the following operators in terms of $\triangle$:

$$S \bigtriangledown T \stackrel{\text{def}}{\equiv} \neg(S \triangle \neg T)$$

$$\diamondsuit S \stackrel{\text{def}}{\equiv} \mathit{true} \triangle S \qquad \boxdot S \stackrel{\text{def}}{\equiv} \mathit{true} \bigtriangledown S$$

The universal projection $S \bigtriangledown T$ is true iff T is true for *all* projections bridged by S. Thus, $(\mathit{len} \bmod 2 = 0) \bigtriangledown \mathit{stable}\ A$ is true on an interval iff no matter how we break it up into consecutive parts each having even length, the variable A is stable on the projected interval. This formula is vacuously true for intervals with odd length and is logically equivalent to

$$(\mathit{len} \bmod 2 = 1) \vee (\mathit{len} = 2 \wedge A \leftarrow A)^*.$$

The operators $\diamondsuit$ and $\boxdot$ are simple forms of projection with the trivial bridge formula *true* and are duals ($\boxdot S \equiv \neg \diamondsuit \neg S$). Consider the formula $\diamondsuit(K\ \mathit{gets}\ K+2)$ which is true iff K periodically increases by 2. It is equivalent to $(K \leftarrow K+2)^*$.

The following axioms and inference rule provide a basis for formal reasoning about projection. We add them to our conventional propositional ITL proof system presented earlier in §3.

PJ1 $\vdash S \triangle (T \vee U) \supset S \triangle T \vee S \triangle U$

PJ2 $\vdash S \triangle \mathit{empty} \equiv \mathit{empty}$

PJ3 $\vdash S \triangle \mathit{skip} \equiv (S \wedge \mathit{more})$

PJ4 $\vdash S \triangle (T;U) \equiv (S \triangle T);(S \triangle U)$

PJ5 $\vdash S \triangle w \supset w$

PJ6 $\vdash \mathit{skip} \triangle S \equiv S$

PJ7 $\vdash (S \triangle T) \triangle U \equiv S \triangle (T \triangle U)$

PJ8 $\vdash \boxdot(S \supset S') \supset S \triangle T \supset S' \triangle T$

PJ9 $\vdash S \bigtriangledown (T \supset T') \supset S \triangle T \supset S \triangle T'$

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

Axiom **PJ4** lets us decompose a projection of a sequential formula into its parts. Other axioms also facilitate modular reasoning by permitting us to move formulas in and out of projected intervals. Kono [16] discusses a complete proof system for propositional ITL including our projection operator and also looks at a decision procedure. However he does not handle compositional reasoning.

Some propositional theorems are shown below:

$\vdash \boxdot S \equiv \boxdot\boxdot S, \qquad \vdash \Box w \equiv \boxdot \Box w,$

$\vdash S \triangle (T^*) \equiv (S \triangle T)^*, \qquad \vdash S^* \equiv S \triangle \mathit{true}.$

$\vdash S \triangle (w \wedge \bigcirc T) \equiv w \wedge (S \wedge \mathit{more});(S \triangle T)$

The last theorem involving *chop-star* shows how to express this operator using projection. Thus, if we wish, we can take *skip*, *chop*, and $\triangle$ to be our three basic temporal operators.

The axioms and inference rule are complete for formulas with propositional state variables. This is because we can first transform a formula containing projection operators into a logically equivalent one without them and then exploit the completeness of the basic ITL proof system. We note that our first-order ITL proof system given in [1] can be augmented with the following useful axiom for moving existential quantification out of projection:

FOLJ1 $\vdash S \triangle \exists v{:}T \supset \exists v{:}(S \triangle T)$,

where v is a variable not occurring freely in S.

Suppose we have a projected system $S \triangle \mathit{Sys}$. One way to reason about it is by proving that it is equivalent to some projection-free formula. We will illustrate this later in §4.1. Alternately we can employ the following derivable inference rule which provides a way to compositionally reason about the assumptions and commitments of a projected formula:

$$\frac{\vdash w \wedge \mathit{As} \wedge \mathit{Sys} \supset \mathit{Co} \wedge \mathit{fin}\ w}{\vdash w \wedge \mathit{As} \wedge (S \triangle \mathit{Sys}) \supset \mathit{Co} \wedge \mathit{fin}\ w.} \qquad (3)$$

As with the compositional proof rules (1) and (2) mentioned earlier, this is not sound for arbitrary assumptions and commitments. Here is what we need:

$$\vdash \mathit{As} \supset S \bigtriangledown \mathit{As}, \qquad \vdash S \triangle \mathit{Co} \supset \mathit{Co}.$$

This requires that the assumption As can be imported into the projected interval and that the commitment Co can be exported from it. Here are some instances:

$\vdash \mathit{goodindex}\ K \supset \mathit{padded}\ K \bigtriangledown \mathit{goodindex}\ K$

$\vdash \mathit{padded}\ K \bigtriangledown \mathit{stable}\ K \supset \mathit{stable}\ K.$

The importability of As ensures that we can conjoin it with Sys in the projected interval for further analysis:

$\vdash \mathit{As} \wedge (S \triangle \mathit{Sys}) \supset S \triangle (\mathit{As} \wedge \mathit{Sys}).$

In some cases, Co's constraint needs to include As in order to establish Co's exportability:

$\vdash \mathit{As} \wedge (S \triangle \mathit{Co}) \supset \mathit{Co}.$

Here is a small example:

$\vdash \mathit{stable}\ K \wedge (\mathit{padded}\ L \triangle \mathit{stable}\ K+L) \supset \mathit{stable}\ K+L$

We need $\mathit{stable}\ K$ when exporting $\mathit{stable}\ K+L$ out of the projection. In general, this can be viewed as using $\mathit{As} \supset \mathit{Co}$ instead of Co as the commitment in rule (3). A variant proof rule based on this can easily be derived as a corollary of the original proof rule. It is important to observe that these constraints do not depend on the projected formula Sys. Once we establish them, any projections involving S, As and Co can modularly use inference rule (3).

4.1 An application using projection

Consider the following definition of a simple system:

$$\begin{array}{l} \mathit{IndexSkel}(K,L) \stackrel{\text{def}}{\equiv} \\ \quad \mathit{for\ some\ times\ do}\ (\\ \quad\quad \mathit{stable}\ K \wedge \mathit{fin}\ (K \neq L) \\ \quad\quad K \ll K+1; \\ \quad); \\ \quad \mathit{stable}\ K \end{array}$$

The *for-loop* used here is a programming language construct representing *chop-star.* The definition repeatedly increments the variable K as determined by K's interaction with the variable L. Two such systems can be combined in parallel to obtain compositionally provable theorems such as the following:

$$\vdash\ K = L \wedge \mathit{IndexSkel}(K, L) \wedge \mathit{IndexSkel}(L, K + n) \supset\ \Box(K \leq L \leq K + n).$$

Let us define a projected version of *IndexSkel*:

$$\mathit{ProjIndexSkel}(K, L) \stackrel{\text{def}}{\equiv} \mathit{padded}\ K \triangle \mathit{IndexSkel}(K, L).$$

By using sequential decomposition and analysis, we can prove the equivalence of both definitions:

$$\vdash\ \mathit{ProjIndexSkel}(K, L) \equiv \mathit{IndexSkel}(K, L) \qquad (4)$$

Here are some typical general purpose lemmas used in the proof:

$$\vdash\ S \triangle (T \wedge \mathit{fin}\ w) \quad \equiv \quad (S \triangle T) \wedge \mathit{fin}\ w$$
$$\vdash\ \mathit{stable}\ A \quad \equiv \quad \mathit{padded}\ A \triangle \mathit{stable}\ A$$
$$\vdash\ A \lessapprox B \quad \equiv \quad \mathit{padded}\ A \triangle (A \lessapprox B)$$

There are other ways to express *IndexSkel* which are logically equivalent to the definition given above but where the decomposition for projection is not so straightforward. In addition, larger systems might have no provable equivalence corresponding to lemma (4). Let us therefore now look at a more modular way to handle the projected version using the compositional proof rule (3). We consider part of the analysis of the following conjunction:

$$\mathit{ProjIndexSkel}(K, L) \wedge \mathit{ProjIndexSkel}(L, K + n)$$

Our focus is on proving the lemma

$$\vdash\ K \leq L \wedge \mathit{goodcounter}\ L \wedge \mathit{ProjIndexSkel}(K, L) \supset\ \mathit{keep}\,(K \leq \bigcirc K \leq L) \wedge \mathit{fin}\,(K \leq L).$$

The formula $\mathit{goodcounter}\ L$ is the assumption and the formula $\mathit{keep}\,(K \leq \bigcirc K \leq L)$ is the commitment. Roughly speaking, this states that K patiently increases with its next value never exceeding the current value of L. We assume the following similar nonprojected lemma has already been deduced for *IndexSkel*:

$$\vdash\ K \leq L \wedge \mathit{goodcounter}\ L \wedge \mathit{IndexSkel}(K, L) \supset\ \mathit{keep}\,(K \leq \bigcirc K \leq L) \wedge \mathit{fin}\,(K \leq L).$$

The lemma shown below demonstrates that the assumption $\mathit{goodcounter}\ L$ can be imported into the projection:

$$\vdash\ \mathit{goodcounter}\ L\ \supset\ (\mathit{padded}\ K \bigtriangledown \mathit{goodcounter}\ L).$$

The following must be established for the commitment $\mathit{keep}\,(K \leq \bigcirc K \leq L)$:

$$\vdash \mathit{goodcounter}\ L \wedge \big(\mathit{padded}\ K \triangle \mathit{keep}\,(K \leq \bigcirc K \leq L)\big) \supset\ \mathit{keep}\,(K \leq \bigcirc K \leq L).$$

We can now apply the corollary version of proof rule (3). Further details are omitted here.

5 Infinite time

The semantics and proof system so far presented is suitable for reasoning about finite intervals. We briefly discuss some modifications needed to permit both finite and infinite intervals. This enables us to extend the techniques of Jones to the analysis of liveness properties [7]. First, we apply our semantics of $S;T$ and S^* to infinite intervals. As before this means $S;T$ is true 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 even if it is false. Furthermore, we also let S^* be true on an infinite interval which can be split 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 $\Diamond$:

$$\mathit{inf} \stackrel{\text{def}}{\equiv} \mathit{true};\mathit{false} \qquad \mathit{finite} \stackrel{\text{def}}{\equiv} \neg \mathit{inf} \qquad \Diamond S \stackrel{\text{def}}{\equiv} \mathit{finite};S$$

The definition of *inf* is true iff *chop* completely ignores the right subformula *false* and *chop* does this precisely on infinite intervals. Using these definitions, all the axioms and basic inference rules remain sound. We also include two new axioms:

PI1 $\vdash\ (S \wedge \mathit{inf});T\ \equiv\ S \wedge \mathit{inf}$

PI2 $\vdash\ S \wedge \Box\big(S \supset (T \wedge \mathit{more});S\big)\ \supset\ T^*$

Most likely, completeness can only be achieved with a nonconventional inference rule. This is not central to our approach. It is helpful to redefine the temporal assignment operator $\leftarrow$ to be limited to finite intervals:

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

The variant operator $\lessapprox$ is similarly redefined.

5.1 Compositionally proving absence of deadlock

Let us now present a compositional analysis proving the absence of deadlock. Only the overall structure of the proof is given.

Below are descriptions of two simple processes Q and R which alternately modify a single variable K:

$Q(K) \stackrel{\text{def}}{\equiv}$
for some times do (
$K \lessapprox K + 1$;
halt even(K)
)

$R(K) \stackrel{\text{def}}{\equiv}$
for some times do (
halt odd(K);
$K \lessapprox K + 1$
)

The iterating in Q and R 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 with K initially equal to 0:

$$K = 0 \wedge Q(K) \wedge R(K).$$

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

Here is a theorem for system correctness:

$$\vdash even(K) \wedge Q(K) \wedge R(K) \supset goodindex\ K \wedge fin\ even(K).$$

The theorem uses the *goodindex* operator defined earlier to state that K is always stable or increases by 1. In addition, K's final value is even. In [1] we consider how to compositionally prove this. 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. We still must ensure that when the system operates over an infinite interval, K never gets stuck:

$$\vdash\ inf \wedge even(K) \wedge Q(K) \wedge R(K)\ \supset\ \Box\neg\, stable\ K.$$

In order to permit a compositional proof of this, we introduce an auxiliary boolean variable. This is done without loss of generality since the following is deducible from our ITL proof system:

$$\vdash\ \exists CF{:}\,\Box(CF \equiv \neg\, stable\ K). \tag{5}$$

Here CF stands for *change flag* and is defined to be true iff K changes some time in the future from its current value. This lemma is an instance of the following lemma for existentially constructing state variables:

$$\vdash\ \exists A{:}\,\Box(A = e),$$

where A does not occur in freely in e. Here we view the operator $\equiv$ in (5) as a special case of equality for boolean values. However, the formula $\Box(CF \equiv \neg\, stable\ K)$ is itself not suitable as an assumption in proof rules (1) and (2) for sequential composition. This is because its truth like that of most $\Box$-formulas does not guarantee truth in all subintervals, only terminal ones. Therefore it is not expressible as a $\boxdot$-formula usable in proof rules (1) and (2) given in the introduction. However, it does imply another formula which is suitable:

$$\vdash\ \Box(CF \equiv \neg\, stable\ K) \supset \boxdot(\neg\, stable\ K \supset CF). \tag{6}$$

The consequent $\boxdot(\neg\, stable\ K \supset CF)$ states that in any subinterval where K is not stable, CF is true in the initial state. This was not permitted as an assumption in Jones' original formulation. The next lemma plays an important role in our overall analysis:

$$\vdash A \neq B \wedge \boxdot(\neg stable\ A \supset CF) \wedge A \ll B\ \supset\ keep\ CF.$$

This states that if A is padded and its initial and final values in the (finite) interval differ, then CF is always true except possibly in the last state. Here is a slightly simplified substitution instance of this of use when the variable K increases by 1:

$$\vdash\ \boxdot(\neg\, stable\ K \supset CF) \wedge K \ll K+1\ \supset\ keep\ CF.$$

We have omitted the subformula $K \neq K+1$ since it is trivially true. Here are theorems about Q and R:

$$\begin{aligned}
&\vdash\ even(K) \wedge \boxdot(\neg\, stable\ K \supset CF) \wedge Q(K) \\
&\qquad \supset\ \ keep\,(even(K) \supset CF) \wedge fin\ even(K), \\
&\vdash\ even(K) \wedge \boxdot(\neg\, stable\ K \supset CF) \wedge R(K) \\
&\qquad \supset\ \ keep\,(odd(K) \supset CF) \wedge fin\ even(K).
\end{aligned}$$

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 Q:

$$\begin{aligned}
&\vdash\ even(K) \wedge \boxdot(\neg\, stable\ K \supset CF) \wedge K \ll K+1 \\
&\qquad \supset\ \ keep\,(even(K) \supset CF) \wedge fin\ odd(K), \\
&\vdash\ odd(K) \wedge \boxdot(\neg\, stable\ K \supset CF) \wedge halt\ even(K) \\
&\qquad \supset\ \ keep\,(even(K) \supset CF) \wedge fin\ even(K).
\end{aligned}$$

We combine the theorems for Q and R in parallel to obtain the following:

$$\begin{aligned}
&\vdash\ even(K) \wedge \boxdot(\neg\, stable\ K \supset CF) \wedge Q(K) \wedge R(K) \\
&\quad \supset\ \ keep\ CF.
\end{aligned}$$

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

$$\begin{aligned}
&\vdash\ inf \wedge keep\ CF\ \supset\ \Box CF \\
&\vdash\ \Box(CF \equiv \neg\, stable\ K) \wedge \Box CF\ \supset\ \Box\neg\, stable\ K.
\end{aligned}$$

From these and (6) we get the lemma below:

$$\begin{aligned}
&\vdash\ inf \wedge even(K) \wedge \Box(CF \equiv \neg\, stable\ K) \\
&\qquad \wedge Q(K) \wedge R(K)\ \ \supset\ \ \Box\neg\, stable\ K.
\end{aligned}$$

The variable CF can be existentially quantified:

$$\begin{aligned}
&\vdash\ inf \wedge even(K) \wedge \exists CF{:}\,\Box(CF \equiv \neg\, stable\ K) \\
&\qquad \wedge Q(K) \wedge R(K)\ \ \supset\ \ \Box\neg\, stable\ K.
\end{aligned}$$

Finally, we completely hide CF using lemma (5):

$$\vdash\ inf \wedge even(K) \wedge Q(K) \wedge R(K)\ \supset\ \Box\neg\, stable\ K.$$

6 Discussion

We have presented the basis of a methodology for modular specification and verification of concurrent systems with multiple granularities of time and infinite time. The ideas seem promising for both small and big applications. Our approach can be seen as a logical outgrowth of earlier work on compositional reasoning using assumptions and commitments. All of the proof rules can be formally derived from a rather low-level axiom system for ITL. Our plans include applying these and other methods to the formal specification and

analysis of various conceptual layers of the EP/3 multithreaded computer [23] being built by Dr. J. N. Coleman at the University of Newcastle. Another aim is to extend the ideas described here to also handle conventional imperative programming constructs expressed directly in ITL. We have done many proofs by hand and it would be nice to use a mechanized tool. Another aim is to study a first-order variant of the projection operator having the form $S \triangle_i T$. This binds the static variable i in each scope of the bridge formula S to the current step number in the projected formula T. Research on temporal projection from continuous time intervals found in the Duration Calculus [21], a variant of ITL, to discrete intervals might help our understanding of the logical relationship between analog and digital models of dynamic systems.

Acknowledgements

We wish to thank Antonio Cau, Nick Coleman, Roger Hale, Tony Hoare, Shinji Kono, Maciej Koutny, Anders Ravn, Hussein Zedan and the ProCoS working group for discussions. The Engineering and Physical Sciences Research Council kindly funded our research.

References

[1] B. Moszkowski. Some very compositional temporal properties, in: Programming Concepts, Methods and Calculi, E.-R. Olderog (ed.), IFIP Transactions, Vol. A-56, North-Holland, 1994, 307–326.

[2] B. Moszkowski. Reasoning about Digital Circuits, Ph. D. thesis, Stanford Univ., 1983.

[3] J. Halpern, Z. Manna, and B. Moszkowski. A hardware semantics based on temporal intervals, in: ICALP83, LNCS 154, Springer, '83, 278-291.

[4] B. Moszkowski. A temporal logic for multilevel reasoning about hardware, IEEE Computer, Vol. 18, No. 2, 1985, 10–19.

[5] N. Francez and A. Pnueli. A proof method for cyclic programs, Acta Inf., Vol. 9, 1978, 133–157.

[6] C. B. Jones. Specification and design of (parallel) programs, in: Proc. Information Processing '83, R. E. A. Mason (ed.), Elsevier, 1983, 321–332.

[7] L. Lamport. Proving the correctness of multiprocess programs, IEEE Trans. Software Eng., Vol. SE-3, No. 2, 1977, 125–143.

[8] C. A. R. Hoare. An axiomatic basis for computer programming, Comm. ACM, Vol. 12, No. 10, 1969, 576–580, 583.

[9] R. Rosner and A. Pnueli. A choppy logic, in: Proc. 1st Ann. IEEE Symp. Logic in Comp. Sci., '86, 306–314.

[10] B. Paech. Gentzen-systems for propositional temporal logics, in: Proc. 2nd Workshop on Comp. Sci. Logic, LNCS 385, Springer-Verlag, Berlin, 1988, 240–253.

[11] K. Stølen. Proving total correctness with respect to a fair (shared-state) parallel language, in: Proc. 5th BCS FACS Refinement Workshop, London, Springer-Verlag, 1992, 320–341.

[12] Q. Xu and J. He. A theory of state-based parallel programming: Part 1, in: Proc. 4th BCS-FACS Refinement Workshop, J. Morris, ed., Cambridge, UK, Springer, '91.

[13] Q. Xu, A. Cau and P. Collette. On unifying assumption-commitment style proof rules for concurrency, in: Concur'94, LNCS 836, Springer, '94, 267–282.

[14] Q. Xu, W.-P. de Roever and J. He. Rely-guarantee method for verifying shared variable concurrent programs, report 9502, Inst. of Comp. Sci. II, Kiel University, Germany, 1995.

[15] B. Jonnson and Y.-K. Tsay. Assumption/-guarantee specifications in linear-time temporal logic, report DoCS95/58, Dept. Comp. Sys., Uppsala Univ., Sweden. (Presented at TAPSOFT'95).

[16] S. Kono. A combination of clausal and non clausal temporal logic program, in: Executable Modal and Temporal Logics, M. Fisher and R. Owens (ed.), Lect. Notes in Artificial Intelligence 897, Springer-Verlag, Berlin, 1995, 40–57.

[17] B. Dutertre. On first order interval temporal logic, in: 10th Ann. IEEE Symp. Logic in Comp. Sci., '95, 36–43.

[18] R. W. S. Hale and J. He. A real-time programming language, in: Towards Verified Systems, J. Bowen (ed.), Elsevier, 1994.

[19] R. W. S. Hale. Program compilation, in: Towards Verified Systems, J. Bowen (ed.), Elsevier, '94.

[20] J. U. Skakkebæk and N. Shankar. Towards a Duration Calculus proof assistant in PVS, in: Proc. Symp. Formal Techniques in Real-Time and Fault Tolerant Sys., LNCS 863, Springer-Verlag, Berlin, 1994, 660–679.

[21] Zhou Chaochen, C. A. R. Hoare and A. P. Ravn. A calculus of durations, Inf. Proc. Let., Vol. 40, No. 5, 1991, 269–276.

[22] B. Moszkowski. Executing Temporal Logic Programs, Cambridge U. Press, Cambridge, 1986.

[23] J. N. Coleman. A high speed dataflow processing element and its performance compared to a von Neumann mainframe, in: IEEE 7th Int'l. Parallel Processing Symp., 1993, 24–33.