Mexitl: Multimedia in Executable Interval Temporal Logic*

Howard Bowman[1], Helen Cameron[2], Peter King[2] and Simon Thompson[1]

[1] Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent, CT2 7NF, United Kingdom
[2] Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, R3T 2N2, Canada
Email Contacts: {H.Bowman,S.J.Thompson}@ukc.ac.uk and {prking,hacamero}@cs.umanitoba.ca

Abstract. This paper explores a formalism for describing a wide class of multimedia document constraints, based on an interval temporal logic. We describe the requirements on temporal logic specification that arise from the multimedia documents application area. In particular, we highlight a canonical specification example. Then we present the temporal logic formalism that we use. This extends existing interval temporal logic with a number of new features: actions, framing of actions, past operators, a projection-like operator called filter and a new handling of interval length. A model theory, logic and satisfaction relation are defined for the notation, a specification of the canonical example is presented, and a proof system for the logic is introduced.

1 Introduction

This paper explores a formalism for describing a wide class of multimedia document constraints. The formalism is based on an interval temporal logic. The term *multimedia* indicates that a document may contain continuous or time-dependent entities [10] known as *media items* [6], in addition to the more traditional spatial items in paper documents. Part of the task facing the author of such a document, therefore, is to describe the dynamic temporal relationships that are to hold between media items. We are interested in documents with rich sets of such relationships, and as a consequence are keenly interested in issues of consistency verification, modelling, proto-typing, and specification refinement. While a number of authoring systems for multimedia documents are extant [3, 9], little work has been done on the question of suitable formalisms for the specification and manipulation of such temporal constraints in the context of multimedia.

This paper introduces an interval temporal logic specification notation called *Mexitl* which stands for *Multimedia in Executable Interval Temporal Logic*. While

* Travel grants to support the research presented here have been provided by the British Council. The second and third authors are supported by individual research grants from the Natural Sciences and Engineering Research Council of Canada.

the anticipated application area for this notation is multimedia, it is also relevant to other areas of real-time specification. King [11, 12] proposed the use of Interval Temporal Logic (ITL) to specify multimedia documents and the theory presented here stems from that earlier work.

A major difference between ITL and standard linear time temporal logic [15], is that ITL is interpreted over finite state sequences, called intervals, rather than over infinite models. A number of authors have investigated ITLs, including [4, 8, 13, 16]. The restriction to finite states prompts consideration of a number of temporal operators not typically found in non-interval temporal logics, such as *chop* and *projection*, which will be discussed in Section 3.1.

We anticipate that complete specifications of multimedia documents will have a number of elements. Firstly, an *abstract data typing* notation will be used in order to describe the primitive *operations/actions* (we use these terms interchangeably in the sequel) to be used in specifying the document. Actions such as *displayCaption* or *playVideo* are typical examples. A suitable abstract data typing notation is suggested in [11]. We will not consider this notation here, rather the specification language that we present takes the primitive actions as given. In addition, mechanisms to define composite actions out of primitive actions can be added.

We introduce a methodology for developing multimedia artefacts using *Mexitl*. Specifications are written in the logic, and refined according to the rules of the language. Implementations can be developed in various ways, including either by means of deterministic refinements or as proofs of *Mexitl* formulas interpreted in a constructive logic.

Structure of the Paper. The paper is structured as follows. Section 2 reviews the requirements associated with multimedia documents. In particular, an example of a typical presentation from the field is highlighted. Section 3 presents the specification notation that we are advocating. The operators of the core language are presented, the model theory is highlighted and the satisfaction relation is defined. Section 4 contains an account of how the operators used in the paper are derived from the core operators of the language. Then Section 5 applies the defined notation to the requirements highlighted in Section 2. Section 6 gives an initial account of the logic of the *Mexitl* operators. Related work is discussed in Section 7 and concluding remarks are presented in Section 8.

Note that a shorter version of this paper has been published as [2].

2 Multimedia Documents

2.1 Multimedia Document Requirements

There is no extant formal or standard taxonomy of functional requirements for temporal constraints among media items in multimedia documents. However, the multimedia literature displays a good deal of commonality and agreement in this regard. Indeed, in [6], Erfle presents a set of eighteen issues, or functional requirements, which he regards as being sufficient to describe multimedia documents. This set was obtained by a study of what is provided in a number of existing authoring systems and standards. In [11], King presents an alternative set of eight requirements, which encompasses Erfle's set. We will not repeat this work in any detail here, but for the sake of completeness we will provide a summary of the requirements. We will divide our summary into two sets of requirements, a set of three general requirements, which are dictated mainly by the authoring aspect of this area of application, followed by eight individual functional requirements.

This summary of requirements will be illustrated by a fairly complex example. In Section 5.2, we will show how each component of this example is expressible in the formalism to be described in Sections 3 and 4.

General Requirements The most basic requirement is to represent the *display* of a media item. Indeed, we require various forms of display. We first require what we might term *standard* display, where a media item is simply presented in its normal fashion at its normal rate. In addition, to the extent that the physical properties of the medium in question permit them, we must allow in our formalism for variations of this standard display such as presenting at half speed, rewind, fast-forward, and so forth.

Secondly, we need to represent the notion of *reader intervention* during a multimedia display. This intervention will frequently require *early termination* of a media item.

Thirdly, we require facilities for *composing* sets of constraints. Documents, just like programs, are rarely static or fixed, and new documents are often created by adding to existing ones. An author will frequently use a section of an existing document as part of a new, larger document. We thus require the ability to express both serial and parallel composition of sets of media item constraints. Parallel composition is also required to permit independent development of what are termed *channels* in the multimedia literature [3, 9], which may then be combined so that they occur in the same time interval – that is, in the same multimedia presentation.

In this work we have chosen to generalise somewhat the notion of *channel* as it is used in the multimedia literature. See [3, 9], for example, where the term channel refers to a single type of medium, and one may refer, for example, to the video channel or the audio channel. By way of contrast, our notion of channel is a higher-level one, corresponding more closely to the notion of an independent author. In our work, therefore, a channel will usually contain items of differing physical media types, since a single author may well wish to make use of such a variety of media types in creating part of a complete multimedia document. Our use of the term channel is akin to the use of such terms as "thread" in other areas of computer science.

Functional Requirements The following eight individual items are required, although arbitrary combinations of these items are also to be permitted:

1. Temporal placement of a media item at an absolute (time) point;
2. Specification of the duration of a media item;
3. Determination of the start and finish points of a media item;
4. Relative placement of two or more media items;
5. Repetitive display of a media item;
6. Conditional display of a media item;
7. Scripting, that is, using events or conditions occurring in one media item to control the display of a second; and
8. Exception handling, that is controlling error situations which may occur during the display of a multimedia document.

2.2 Example

We will now illustrate the requirements list given in the previous section by introducing an informal specification of a fairly elaborate example of a multimedia document. We will annotate the various parts of this example with references to the corresponding items in the list given above. This entire example will be expressed in the ITL formalism in Section 5 below. We refer to this example as the *Beethoven Problem*. The Beethoven Problem requires the development of a multimedia document consisting of an audio of Beethoven's Fifth Symphony, Opus 67 in C minor, together with various other media items which are designed to illustrate the music as it is played. King [12] presents an earlier version of a portion of this example.

Beethoven's Fifth Symphony comprises four movements.

1. Play the four movements of the symphony in sequence with a gap of 20 seconds between each movement.
   This simple example uses sequential composition and duration specification; it involves Requirements 2 and 4. Note that King [11] shows in detail how n-ary temporal relations can be represented in (a subset of) the formalism described herein; we will not repeat the details of this. Strictly speaking, this example also involves Requirement 1, since we are assuming that the symphony is to begin at time zero. Note also that in what follows we will not be very concerned about temporal units or the granularity of real time. We may, for example, simply assume that one clock tick equals one second, or any other convenient unit.
2. Before the first movement begins, play an audio which announces the name of the symphony, the composer, and the orchestra. Two seconds after this audio starts, display a video still of Beethoven. Stop the video still display as the first movement starts. After the last movement, wait 3 seconds and display for 30 seconds a video of Ludwig van Beethoven and then, after a further 5 seconds, display information about how to order this presentation.
   The various parts of this specification may be composed in a variety of ways. For example, each might occupy a separate channel and all channels then be composed in parallel. Alternatively, they may be incorporated by

serial composition into one of the existing channels. The choice to be made is largely a matter of taste. As individual items, they involve serial composition together with Requirements 2 and 4.

3. At the start of each movement display an appropriate title for 5 seconds. Each title is a video still. Repeat the 5 second display of each title every 3 minutes during the corresponding movement.

   We may regard the display of the four video stills as comprising a second channel, which is to be composed in parallel with the first: the four movements. We have an instance of Requirement 5, a repetitive display. The actual display of the video stills themselves requires duration specification as in Requirement 2.

4. The audio introduced in Example 2 is actually in three parts, corresponding to the name of the symphony, the composer, and the orchestra. During this audio display,

   either display a rolling text item containing the same information (assume this item has the same length as the audio)

   or display, in sequence, three video stills containing the same information for the appropriate time lengths.

   This example is a further instance of parallel composition, but also introduces a conditional specification (Requirement 6), and also involves some simple sequential composition.

5. The twenty second gap between the second and third movements should be replaced by a video/audio display describing the third movement. If the video/audio display takes longer than twenty seconds, truncate the display to twenty seconds.

   This example illustrates the need for an exception handling facility; we must allow for truncation of the display in question if it does in fact require longer that 20 seconds, and decide what happens if it requires less that 20 seconds. Note, therefore, that we require the start and finish points of the item, an illustration of Requirement 3.

6. At the first sound of a viola, display a purple band for 3 seconds.
7. During the first crescendo passage, display a mammoth.
8. During each crescendo passage of the first movement, display a looped video tape of a bug climbing an inclined plane. For each crescendo passage after the first one, continue playing the video from the point at which it was previously stopped.
9. Count the number of C minor chords in the symphony.
10. During each staccato note of the first movement, flash the screen red.

    Examples 6 to 10 illustrate the need for what we have chosen to call *scripting*, using events or conditions in one media item to control the display of a second. Examples 6 and 7 refer to *a particular* instance of such conditions, the first occurrence in both these cases. Examples 8, 9 and 10 refer to *all* instances of such conditions. Note that these conditions that may hold at a specific time point (staccato) or over an interval (crescendo). Example 9 is a somewhat different example of scripting, where the operation to be performed is a housekeeping task, counting the number of chords, rather than the display of a media item.

This definition of the Beethoven problem illustrates each of the eight items listed under functional requirements. It also illustrates some, but not all, of the general requirements appearing in Section 2.1. Specifically, the notions of parallel and sequential composition are illustrated, but the notions of variable speed display and of reader intervention are not included. In order to illustrate these latter points, we now present two additions to the Beethoven problem, though these additions will not form part of the formal *Mexitl* specification to be presented in Section 5.

1. The reader may interrupt and stop the audio occurring at the beginning of the presentation (introduced in Part 2 above) and proceed directly to the start of the first movement.
2. The reader may stop and rewind the tape containing any movement. The reader may then replay that movement, and at any subsequent point may fast-forward to the end of that movement.

3 Introduction to *Mexitl*

We present a *core language* for *Mexitl*, which contains the primitive constructs of the notation. Then we describe the model theory underlying the language; this theory is based upon finite sequences of states (called *intervals*) and we define the satisfaction relation, which characterises the intervals that satisfy a logical formula.

3.1 The Core Language

Expressions The core language uses a simple expression notation. Expressions have the following form:

$$E \ ::= \ c \mid v \mid V \mid f(E) \mid \mathbf{mylen}$$

where $c \in \mathcal{N}$, $v \in \mathbf{Var}_{static}$, the set of static variables, $V \in \mathbf{Var}_{state}$, the set of state variables, and f is in a set of assumed functions (in some areas of the literature static and state variables are called respectively rigid and flexible variables [15]). We also assume a set $\mathbf{Var} = \mathbf{Var}_{static} \cup \mathbf{Var}_{state}$ of variables.

In addition, **mylen** is a distinguished variable which denotes the length of the current interval. This operator reflects the finiteness of models for *Mexitl*. The inclusion of **mylen** is a departure from standard interval temporal logic. We discuss the motivation for its inclusion in the next subsection.

The Logic The domain of logical propositions is denoted $\mathcal{P}$ and $P \in \mathcal{P}$ is constructed as follows:

$$P ::= \mathbf{read}_X(V) \mid a_X(E) \mid a_X \mid p(E_1, ..., E_n) \mid E = E \mid \mathbf{False} \mid P \Rightarrow P \mid$$
$$P \ ; \ P \mid P \ \ \mathbf{proj} \ \ P \mid P \ \tilde{;} \ P \mid (\exists x \leq E)P \mid P \ \ \mathbf{filter} \ \ P$$

Among the above propositions, **read** and a belong to the set **Act** of actions and X is a set of such actions. Also p is in a set of given predicates; V is a state variable and E is an expression, as defined above.

Much of this logic will be well known to a reader familiar with interval temporal logic.

- **False** and $P \Rightarrow P$ are the familiar connectives of classical propositional logic.
- $p(E_1, ..., E_n)$ denotes application of a predicate to n expressions. In standard fashion, we will often write binary predicates infix.
- $E = E$ gives equality of expressions.
- **;** is the sequencing operator, *chop*, familiar from [16]. An interval satisfies $P \ ; \ Q$ if the interval can be divided into two contiguous sub-intervals, such that P holds over the first subinterval and Q holds over the second.
- **proj** is the projection operator, also described in [16]. An interval satisfies P **proj** Q if it can be sub-divided into a series of sub-intervals each of which satisfies P - we call P *the projection formula* - and the new interval formed from the end points of these sub-intervals satisfies Q, which we call *the projected formula*.

The reader who requires a more detailed discussion of these operators is referred to [16]. The remainder of our operators are not standard, and thus, require a little more explanation.

Actions. Actions in *Mexitl* are atomic, in the sense that they cannot be analysed into simpler components. Time is discrete, and an action is thought of as taking place in a single state; composite actions are built as *Mexitl* combinations of atomic actions using operators such as **;**.

read(V), $a(E)$ and a define the forms that actions can take. **read**(V) is a distinguished action; it is the only input action. **read**(V) enables new values to be bound to the variable V in the current state. User-defined actions, over which a ranges in the syntax above, will be defined in the separate abstract data typing notation, with their associated type information. Simple non-parametrised actions can be specified by referencing an action, a say, without any data parameter. Thus, this action can occur at a state without accessing the bindings at that state. In contrast, when actions of the form $a(E)$ occur at a state, the value of E with respect to the bindings at that state is associated with the occurrence. One example of such an action is the output of the value of an expression.

From a logical point of view we can think of an action a as an atomic proposition and of $a(E)$ as an application of an atomic predicate.

An action can appear a number of times in an interval; however, each of these represents a different instance of the action. In particular, action occurrences cannot span two states in an interval. Thus, at the level of interval states, actions do not have duration. However, durational behaviour can be obtained by defining composite actions, which are a shorthand for the occurrence of multiple primitive actions. In particular, primitive actions may correspond to indexing into a composite action. For example, an action $playFrame[500]$ might be a constituent of the composite action $playVideo$.

Although actions do not have duration, sets of (distinct) actions can occur at the same state. Such sets reflect simultaneous lock-step occurrence of the actions. In this sense, the model employs synchronous parallelism.

Framing of actions. One aspect which distinguishes our usual perception of logical propositions and actions is the idea of framing. An assertion of a, where a is a particular action, is often interpreted as '*a and no other action* happens' whereas a logical interpretation simply reads this as a happening. The former interpretation, in which the action a is "framed", would lead to a non-monotonic logic were we to adopt it.

Instead of this in our system we subscript the actions with sets X of actions. a_X is interpreted as 'a happens and none of the other actions in X happens'. The set X thus provides an explicit frame within which the action a takes place. Logically the interpretation of a_X is the conjunction of a and $\neg b$ for all b in $X - \{a\}$.

We add a distinguished action – **null** – to the set of actions. This action has null effect, but can be used for framing purposes thus: $\mathbf{null}_X$.

The reason for adding this local framing as primitive is that it supports the composition of separate channels in a modular way as will be seen in the examples below.

A restriction of our model is that it does not support *auto-concurrency*, i.e. multiple instances of the same action occurring at the same state. This slightly restricts generality, however the extra expressiveness obtained does not seem to be needed in our particular area of application.

Length and Next. In contrast to the standard approach to ITL we have not included the next operator, $\bigcirc$, directly in *Mexitl*. However, standard length operators and $\bigcirc$ can be derived from the expression **mylen**, as follows:

$$\mathbf{lesseq}(E) \equiv \mathbf{mylen} \leq E$$

$$\mathbf{len}(E) \equiv \mathbf{mylen} = E$$

$$\bigcirc P \equiv (\mathbf{mylen} = 1) \ ; \ P$$

Thus, only intervals whose length is less than or equal to the value of E will satisfy $\mathbf{lesseq}(E)$, while only intervals of length E will satisfy $\mathbf{len}(E)$. The latter of these is the more common operator, which typically arises as a derived operator in interval temporal logic.

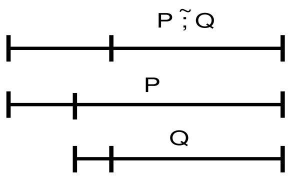
We have included **mylen** as primitive for three reasons:

- It is useful to be able to refer directly to the length of an interval when defining various properties of component parts, such as the two halves into which it is cut by the 'chop' operator.
- Moreover, we use **mylen** as a bound for existential quantifications in derivations of many temporal operators from the core language.
- Finally, including **mylen** allows us to avoid a 'next' operator over expressions, $\bigcirc E$, as in [16], which returns the value of E in the next state. However, $\bigcirc E$ is not well behaved when applied to an empty interval, i.e. one without a next state, since it is not clear what value should be returned. An undefined value could be included in order to cope with this situation. Our approach, however, avoids this extra complexity by incorporating **mylen** in the core language and deriving $\bigcirc P$ (next over propositions) from it.

Past Operators. We include a single past operator in *Mexitl*, chop in the past, denoted $\tilde{;}$. An implication of the inclusion of past operators is that the past history of a computation must be recorded; we will show how this is done in Section 3.4. Using a more sophisticated model theory, $P \mathbin{\tilde{;}} Q$ is satisfied by an interval such that

1. P holds over the larger interval resulting from moving the start of the interval into the past, and
2. Q holds over the original interval according to a past history that is truncated at the start of the interval over which P holds.

The following diagram illustrates this situation.



Intervals are depicted as line segments with three reference points. The leftmost point is the start of time, then moving to the right, the next point is the start of the current interval and the final point is the end of the current interval. Thus, time is divided into a past history and a current interval. The effect of $P \mathbin{\tilde{;}} Q$ is to make P hold over some interval arising from moving the start of the current interval into the past and Q holds over the current interval with a correspondingly truncated past. So, amongst other things, $\tilde{;}$ has the effect of chopping up the past, in a dual manner to which ; divides up the future. This duality is examined further in Section 6.7.

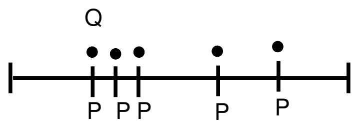
The more familiar past operators, since and previous, can be derived from chop in the past; see Section 4.

The incorporation of past operators is a significant departure from standard interval temporal logic. The motivation for their inclusion is that a number of examples are simplified by their use.

Quantification. In our core language we have included a limited form of existential quantification, namely quantification in which the value of the variable is *bounded* by (the value of) an expression: $(\exists x \leq E)P$. The effect of an existential quantification is to introduce a new variable in the scope of the quantifier. A bounded quantification has the property that its satisfaction over a given interval remains decidable; this is not the case for unbounded quantification over the natural numbers.

Finally note that it is implicitly assumed that all static variables are universally quantified in the outermost context of a proposition. This assumption does not limit decidability.

Filter. The filter operator provides a variant of projection (**proj**). P **filter** Q has the effect of selecting all points which begin an interval over which the property P holds; the formula holds if over this new interval the property Q is true.



filter generalises **when** as defined in [8]; in Hale's definition of **when** (based on **proj**) the property P needs to be a local property, that is, one without temporal operators. We see the generalisation as valuable – it allows us to express properties of the form 'whenever a musical performance is within 5 minutes of the end of a movement, show some more of a video sequence' (see for example Part 8 of the Beethoven example presented in Section 2). We see no way of deriving **filter** from the other *Mexitl* operators.

3.2 Types and definitions in *Mexitl*

The definition mechanism explained here allows for the naming of types and of values. Note that recursive definitions are not allowed, so it is simply a means by which expressions can be made more readable; all (type) definitions can be expanded away should the reader so wish.

Definitions of types take the form

$$\textbf{type}\ \textit{symphony4} = \ldots \textit{definition of symphony4} \ldots$$

where types are either base types (in this case the type of intervals) or are tuples of types. For instance,

$$\textbf{type}\ \textit{mvt} = \textit{interval}$$

$$\textbf{type } \mathit{symphony4} = (mvt, mvt, mvt, mvt)$$

These definitions are not recursive, nor mutually recursive. One way of ensuring this is to ask for each definition to precede its use; the restriction is eased in short examples.

To form an element of a tuple type elements of the component types are combined; so, for example if

$$mv1 :: mvt \qquad mv2 :: mvt$$

then

$$(mv1, mv2) :: (mvt, mvt)$$

where we use '::' to mean 'is of type'. Components are extracted by '$.n$', so that

$$(mv1, mv2).1 = mv1$$

$$(mv1, mv2).2 = mv2$$

Note that with this convention pattern matching need not be used to extract the components of tuples in, for instance, function definitions. Simple definitions of values take the form

$$name :: type$$

$$name = value$$

For example,

$$mv1, mv2, mv3, mv4 :: mvt$$

$$mv1 = \ldots\ldots$$

$$B5 :: symphony4$$

$$B5 = (mv1, mv2, mv3, mv4)$$

We also allow function definitions which take the form

$$name :: type$$

$$name\ x1\ x2\ ..\ xn = \ldots result \ldots$$

For instance,

$$arrange :: symphony4 \rightarrow interval$$

$$arrange\ m = m.1\ ;\ pause\ ;\ m.2\ ;\ pause\ ;\ m.3\ ;\ pause\ ;\ m.4$$

Again, note that neither simple nor function definitions can be recursive.

3.3 Intervals

Mexitl is defined over finite sequences of states. Each sequence is called an interval and $\mathcal{I}$ denotes the set of all possible intervals; $\sigma \in \mathcal{I}$ has the form:

$$\sigma_0, \sigma_1, ..., \sigma_{|\sigma|}$$

where $|\sigma|$ denotes the length of an interval and σ_i denotes the ith state in an interval. By convention the length of an interval is the number of states minus one and all intervals must have at least one state. We use $[\sigma]^i$ to denote the ith prefix of an interval and $(\sigma)^i$ to denote the ith suffix of an interval. Formally,

$$[\sigma]^i = \sigma_0, ..., \sigma_i$$
$$(\sigma)^i = \sigma_i, ..., \sigma_{|\sigma|}$$

Since we have actions, our states are slightly more sophisticated than those of standard interval temporal logic. Specifically, each state is a pair; the first component of the pair records the actions that occur at that state and the second component records the current data state. More formally, we define Σ the set of possible states as:

$$\Sigma = A \ \times \ D$$

where,

$$A = fin(\mathbf{Act} \ \rightarrow \ \mathcal{N} \cup \{\#\}) \quad and$$
$$D = fin(\mathbf{Var} \ \rightarrow \ \mathcal{N})$$

Thus, all states σ_i in σ are in Σ and have the general form (A_i, D_i), where A_i is the action component at state i and D_i is the data component at state i. A_i is a finite function from actions to data values. Thus, data passing can be associated with the occurrence of an action. Some actions may not have an associated data value, in which case they are mapped by A_i to $\#$ (undefined). This is a technical convenience, which avoids having two representations of actions. D_i is a finite function from variables to data values which defines the bindings of values to variables at the ith state.

We will refer to A_i and D_i directly, when we want to reference a specific component of the ith state.

The construction of our states implies that a particular action can only happen once at a given instant. This effectively prevents auto-concurrency.

3.4 Satisfaction

The model for a *Mexitl* proposition is a set of intervals. Thus, the logic is linear time. However in contrast to the work, for example, of [4, 15], all intervals are finite.

The order of presentation of this section is as follows. First we describe *computation structures*, which enhance intervals in order that past operators can be handled. Then we consider the interpretation of expressions and finally, we describe the satisfaction relation for the main temporal operators.

Computation Structures The major difference between our interpretation of *Mexitl* and the standard interpretations of interval temporal logics, to be found in [8, 16], for example, is that our satisfaction relation embraces past operators. Consequently, the sequence of states that have already been passed through must be recorded. This record is obtained by defining our satisfaction relation over what we call *computation structures*. These computation structures are pairs,

$$(\sigma, i)$$

where, in the normal way, σ is a sequence of states, i.e.,

$$\sigma = \sigma_0, \sigma_1, ..., \sigma_{|\sigma|}$$

(we refer to this as the interval) and i is the *point of reference*. The point of reference identifies the past states of the computation and the interval under current consideration. Specifically, the subsequence:

$$\sigma_0, \sigma_1, ..., \sigma_{i-1}$$

denotes the sequence of states that the system has passed through, we call this subsequence the *history* of the computation, while the subsequence,

$$\sigma_i, \sigma_{i+1}, ..., \sigma_{|\sigma|}$$

denotes the interval under current consideration, we call this subsequence the *current interval*. Clearly, if the point of reference was always set to zero, standard interval temporal logic models would be obtained. Our computation structures are similar to the linear time models associated with classic (non-interval) temporal logics, see for example [15], except that they are finite and, thus, have a last state.

Interpreting Expressions This subsection defines how expressions are interpreted over intervals. Firstly, we denote the meaning of function symbols as,

$$[\![f]\!]$$

It is assumed that this mapping yields standard interpretations for all the function symbols. We similarly interpret predicate symbols, by writing $[\![p]\!]$ and constants by writing $[\![c]\!]$. The interpretations of functions, predicates and constants are independent of intervals.

The application $[\![E]\!]_{(\sigma,i)}$ denotes the meaning of E in interval σ relative to the point of reference i, defined by induction over the syntax of expressions. Constants are interpreted in the obvious way:

$$[\![c]\!]_{(\sigma,i)} = [\![c]\!]$$

The interpretation of both static and state variables is their value in the current state, i.e. at the first state in the current interval,

$$[\![v]\!]_{(\sigma,i)} = D_i(v)$$

$$[\![V]\!]_{(\sigma,i)} = D_i(V)$$

Clearly, it would not matter at which state the value of a static variable was accessed. However, the choice of the current state is significant for state variables, as their values can change over the states of the current interval. This interpretation of state variables corresponds to that classically employed in interval temporal logics.

Function applications are interpreted by applying the interpretation of the function symbol to the interpretation of the expression parameter.

$$[\![f(E)]\!]_{(\sigma,i)} = [\![f]\!]([\![E]\!]_{(\sigma,i)})$$

As indicated earlier the one non-standard aspect of our expression language is the operator **mylen**, which is interpreted as follows:

$$[\![\mathbf{mylen}]\!]_{(\sigma,i)} = |\sigma| - i$$

Thus, the meaning of **mylen** is the length of the current interval.

Interpreting *Mexitl* Our satisfaction relation, $\models$, interprets *Mexitl* propositions over computation structures. The notation $(\sigma, i) \models P$ denotes that the computation structure (σ, i) satisfies (or models) the proposition P.

We interpret *Mexitl* propositions in the usual way, by induction over the structure of propositions. We consider the satisfaction rule for each class of construct in turn.

Actions. The distinguished action **read** is modelled differently to other actions. We have the following rule:

$$\begin{array}{lll} (\sigma, i) \models \mathbf{read}_X(V) & \text{iff} & \mathbf{read}(V) \in dom(A_i) \text{ and} \\ & & \exists x \in \mathcal{N}\ A_i(\mathbf{read}(V)) = D_i(V) = x \text{ and} \\ & & \forall a \in X - \{\mathbf{read}(V)\}\ a \notin dom(A_i) \end{array}$$

Thus any computation structure (σ, i) models $\mathbf{read}_X(V)$ if it satisfies the following:

- the action $\mathbf{read}_X(V)$ appears in the ith action set and is bound to a value x;
- the variable V is bound to x in the ith state; and
- none of the other actions in the set X appears in the ith action set.

Thus, $\mathbf{read}_X(V)$ causes an input action to occur, which binds a new value to the variable V in the current state.

All actions other than $\mathbf{read}_X(V)$ behave like output actions. The satisfaction rule for these is:

$$(\sigma, i) \models a_X(E) \text{ iff } a(E) \in dom(A_i),\ A_i(a(E)) = [\![E]\!]_{(\sigma,i)} \text{ and } \\ \forall x \in X - \{a(E)\}\ x \notin dom(A_i)$$

The value of E is determined by applying $[\![\ ]\!]$ and this value is associated with the occurrence of action $a_X(E)$ at the ith state. It is important to note that E is evaluated according to the bindings of values to variables found in the ith state.

The simple case of an action with no data attributes occurring is interpreted in the following straightforward rule:

$$(\sigma, i) \models a_X \text{ iff } a \in dom(A_i), A_i(a) = \# \text{ and } \forall x \in X - \{a\}\ x \notin dom(A_i)$$

Predicates and Equality of Expressions. Satisfaction of predicates is defined by interpreting the components of the predicate in the expected manner.

$$(\sigma, i) \models p(E_1, ..., E_n) \text{ iff } [\![p]\!]([\![E_1]\!]_{(\sigma,i)}, ..., [\![E_n]\!]_{(\sigma,i)})$$

Interpreting equality of expressions is also straightforward.

$$(\sigma, i) \models E_1 = E_2 \text{ iff } [\![E_1]\!]_{(\sigma,i)} = [\![E_2]\!]_{(\sigma,i)}$$

Non-temporal Logical Operators. The following rules reflect the standard logical interpretation of **False** and $\Rightarrow$.

$$(\sigma, i) \not\models \mathbf{False}$$

$$(\sigma, i) \models P_1 \Rightarrow P_2 \text{ iff } (\sigma, i) \models P_1 \text{ implies } (\sigma, i) \models P_2$$

Interval Temporal Operators. Chop subdivides the given interval into two sub-intervals the first of which satisfies P_1 and the second of which satisfies P_2.

$$(\sigma, i) \models P_1; P_2 \text{ iff } \exists k \in \mathcal{N}\ (i \leq k \leq |\sigma| \text{ and } ([\sigma]^k, i) \models P_1 \text{ and } (\sigma, k) \models P_2)$$

The two sub-intervals arising from chop have one common state, the kth state in the above definition. Thus, the last state of the sub-interval over which P_1 holds is the first state of the sub-interval over which P_2 holds.

The semantics of projection are somewhat more complex.

$$\begin{aligned}(\sigma, i) \models P_1 \ \mathbf{proj}\ P_2 \text{ iff } \ & \exists m \in \mathcal{N} \text{ and } \exists \tau_0, \tau_1, ..., \tau_m \in \mathcal{N} \\ & (i = \tau_0 < \tau_1 < ... < \tau_m = |\sigma| \text{ and} \\ & \forall j < m\ (([\sigma]^{\tau_{j+1}}, \tau_j) \models P_1) \text{ and} \\ & ([\sigma]^{i-1}.\sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m}, i) \models P_2)\end{aligned}$$

This rule states that for a computation structure to satisfy P_1 **proj** P_2 there must exist a sequence of m increasing points (or states) in the current interval, $\tau_0, \tau_1, ..., \tau_m$, such that the first and last points bound the current interval. This sequence of points effectively divides the current interval into a series of m sub-current intervals, each of which comprises the states between τ_j and τ_{j+1}. We require that P_1 holds over each of these sub-current intervals and, in addition, that P_2 holds when $\sigma_{\tau_0}, \sigma_{\tau_1}, ..., \sigma_{\tau_m}$ is viewed as a current interval.

Because we see no need in our area of application to allow $\tau_i = \tau_{i+1}$ and because forbidding **empty proj** Q would simplify an implementation of *Mexitl* using tableaux, the sequence $\tau_0, \tau_1, \ldots, \tau_m$ must be strictly increasing. By way of contrast, in [8, 16] (for example), the sequence $\tau_0, \tau_1, \ldots, \tau_m$ only has to be nondecreasing, and therefore $\tau_i = \tau_{i+1}$ is possible in certain pathological cases.

Past Operators. The semantics of chop in the past are as follows:

$$(\sigma, i) \models P_1 \ \tilde{;}\ P_2 \ \text{ iff } \ \exists k\ (0 \leq k \leq i \text{ and } (\sigma, k) \models P_1 \text{ and } ((\sigma)^k, i - k) \models P_2)$$

Bounded existential quantification. Note that an existentially quantified variable need not be static – its value can be different at different points in the interval. We have

$$(\sigma, i) \models (\exists x \leq E)P \ \text{ iff } \ (\sigma', i) \models P \ \text{ for some } \sigma' \simeq_{x,E} \sigma$$

We say that $\sigma' \simeq_{x,E} \sigma$ if σ and σ' are the same length and they have the same value on all actions and variables except (perhaps) x, and at each point the value of x is less than or equal to the corresponding value of E.

Filter. As can be seen, the formula P_1 is used to select points in the interval σ from which is made the interval over which the value of P_2 is found.

$$\begin{array}{lll}(\sigma, i) \models P_1 \ \textbf{filter}\ P_2 & \text{iff} & \textbf{either } \exists \tau_0, \tau_1, ..., \tau_m \in \mathcal{N} \\ & & i \leq \tau_0 < \tau_1 < ... < \tau_m \leq |\sigma| \text{ and} \\ & & \forall j < m\ ((\sigma, \tau_j) \models P_1) \text{ and} \\ & & \forall k, i \leq k \leq |\sigma| \wedge (\sigma, k) \models P_1 \text{ implies } k = \tau_j \\ & & \quad \text{for some } j \text{ and} \\ & & ([\sigma]^{i-1}.\sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m}, i) \models P_2) \\ & & \textbf{or} \text{ no points satisfy } P_1\end{array}$$

Again, notice that in this definition the sequence $\tau_0, \tau_1, \ldots, \tau_m$ is strictly increasing.

Satisfaction and Validity. An arbitrary *Mexitl* proposition is interpreted relative to a zero point of reference. Thus, we define that an interval σ *satisfies* a proposition P if and only if $(\sigma, 0) \models P$. In addition, in the usual way, we state that P is *valid* if and only if for all σ in $\mathcal{I}$, $(\sigma, 0) \models P$. If P is valid we write $\models P$.

This concludes out description of the syntax and semantics of our logic. The following section expands the language by describing a collection of standard definitions which we use in our example which follows.

4 Derived Operators

The primitive operators of *Mexitl* can be used to derive a large spectrum of further operators. In many circumstances these extra operators prove to be more usable than those of the core language. We derive the other connectives of classical propositional logic in the next subsection. Then we consider a number of different classes of derived temporal operator.

4.1 Logical Connectives

The remaining propositional logic connectives are derived in a standard fashion.

$$\neg P \;\equiv\; P \;\Rightarrow\; \mathbf{False}$$

$$P \;\vee\; Q \;\equiv\; (\neg P) \Rightarrow Q$$

$$P \;\wedge\; Q \;\equiv\; \neg(P \Rightarrow \neg Q)$$

$$\mathbf{True} \;\equiv\; \neg\mathbf{False}$$

$$P \;\Leftrightarrow\; Q \;\equiv\; (P \Rightarrow Q) \;\wedge\; (Q \Rightarrow P)$$

Conditional operators are defined from the logical operators.

$$\mathbf{if}\ P\ \mathbf{then}\ Q\ \mathbf{else}\ R \;\equiv\; (P \Rightarrow Q) \;\wedge\; (\neg P \Rightarrow R)$$

$$\mathbf{if}\ P\ \mathbf{then}\ Q \;\equiv\; (P \Rightarrow Q)$$

Bounded universal quantification is defined in the standard way.

$$(\forall x \leq E)P \;\equiv\; \neg(\exists x \leq E)(\neg P)$$

4.2 Basic Derived Temporal Operators

The following derived operators have all been considered previously in the interval temporal logic literature. See, for example, [8, 13, 16]. We present the majority of them without explanation; if an explanation is needed, it follows the definition.

$$\mathbf{lesseq}(E) \;\equiv\; \mathbf{mylen} \le E$$
$$\mathbf{len}(E) \;\equiv\; \mathbf{mylen} = E$$
$$\mathbf{empty} \;\equiv\; \mathbf{len}(0)$$
$$\mathbf{more} \;\equiv\; \neg\mathbf{empty}$$
$$\bigcirc P \;\equiv\; \mathbf{len}(1) \;;\; P$$
$$\text{\textcircled{w}} P \;\equiv\; \bigcirc P \;\vee\; \mathbf{empty}$$
$$\Diamond P \;\equiv\; \mathbf{True} \;;\; P$$

There exists a terminal interval on which P holds.

$$\Box P \;\equiv\; \neg\Diamond\neg P$$

For all terminal intervals P holds.

$$A \;\mathbf{until}\; B \;\equiv\; (\exists x \le \mathbf{myLen})(\Box((\mathbf{myLen} > x) \Rightarrow A) \wedge \Box((\mathbf{myLen} = x) \Rightarrow B))$$

A holds at each point in the future until a point is reached where B holds.

$$\mathbf{skip} \;\equiv\; \bigcirc\mathbf{empty}$$
$$\mathbf{fin}\; P \;\equiv\; \Box(\mathbf{empty} \Rightarrow P)$$
$$\mathbf{halt}\; P \;\equiv\; \Box(P \Leftrightarrow \mathbf{empty})$$
$$\mathbf{beg}\; P \;\equiv\; (\mathbf{empty} \;\wedge\; P) \;;\; \mathbf{True}$$
$$\mathbf{keep}\; P \;\equiv\; \Box(\neg\mathbf{empty} \Rightarrow P)$$
$$P \;;;\; Q \;\equiv\; (P \;\wedge\; \neg\mathbf{empty}) \;;\; Q$$
$$\Diamond_i P \;\equiv\; P \;;\; \mathbf{True}$$

There exists an initial interval on which P holds.

$$\Box_i P \;\equiv\; \neg \Diamond_i \neg P$$

For all initial intervals P holds.

$$\Diamond_a P \;\equiv\; \mathbf{True} \;;\; P \;;\; \mathbf{True}$$

There exists an arbitrary interval on which P holds.

$$\Box_a P \;\equiv\; \neg \Diamond_a \neg P$$

For all arbitrary intervals P holds.

$$P \;||\; Q \;\equiv\; (P \;\wedge\; (Q \;;\; \mathbf{True})) \;\vee\; ((P \;;\; \mathbf{True}) \;\wedge\; Q)$$

P and Q are composed in parallel; this imposes a weaker condition that conjoining the two, since they are not forced to 'terminate' at the same point.

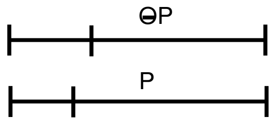
4.3 Past Operators

The following is the complete set of past operator derivations. They are all derived from $\tilde{;}$, with a little help from **mylen**.

$$\begin{aligned}
P_1 \ \tilde{\mathcal{S}} \ P_2 &\equiv x = \mathbf{mylen} \ \Rightarrow \\
&\quad (((P_2 \ \wedge \ \mathbf{mylen} > x) \ \wedge \bigcirc\Box(\mathbf{mylen} > x \Rightarrow P_1)) \ \tilde{;} \ \mathbf{True}) \\
\circleddash P &\equiv \mathbf{False} \ \tilde{\mathcal{S}} \ P \\
P_1 \, \mathcal{S} \, P_2 &\equiv P_2 \ \vee \ (P_1 \ \wedge \ P_1 \ \tilde{\mathcal{S}} \ P_2) \\
\diamondminus P &\equiv \mathbf{True} \, \mathcal{S} \, P \\
\boxminus P &\equiv \neg \diamondminus \neg P
\end{aligned}$$

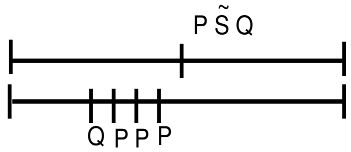
In fact, $\tilde{\mathcal{S}}$ is a strong since operator. The definition of strong since enables previously $\circleddash$ to be derived. Weak since $\mathcal{S}$ is also derived in standard fashion.

Thus, $\circleddash P$ is satisfied by any interval where P holds over the larger interval resulting from moving the start of the interval one place into the past. The following diagram illustrates this situation. It is assumed that the interval P holds over is one state larger than the interval $\circleddash P$ holds over.



If a past history does not exist, that is if the reference point is 0, $\circleddash P$ is false.

Satisfaction of $P \ \tilde{\mathcal{S}} \ Q$ can be illustrated as follows:



In order to simplify this diagram we have not explicitly delimited the intervals that Q and P hold over; we have just indicated the point at which the interval they hold over starts. Implicitly, all intervals satisfying Q or P run through to the end of the indicated time line. In words, $P \ \tilde{\mathcal{S}} \ Q$ holds over an interval if two conditions are satisfied. First, there exists a (strictly) larger interval arising from moving an arbitrary number of states into the past over which Q holds. Second, P holds over all intervals starting at states (strictly) between the start of the larger interval and the current time.

In addition, we can derive Duan's chop in the past [4], which behaves differently in its left argument to $\overset{\sim}{;}$

$$P_1 \;\overset{-}{;}\; P_2 \equiv ((P_1 \wedge \textbf{empty})\ ;\ \textbf{True})\ \overset{\sim}{;}\ P_2$$

and Duan's chop (in the future), denoted $\overset{\wedge}{;}$

$$P_1 \;\overset{\wedge}{;}\; P_2 \equiv P_1\ ;\ (x = \textbf{mylen} \Rightarrow (x = \textbf{mylen}\ \overset{\sim}{;}\ P_2))$$

Before we move on it is worth briefly dwelling on the effect of past operators. Firstly, the combination of ; and past operators implies that histories must be considered. For example, the formula:

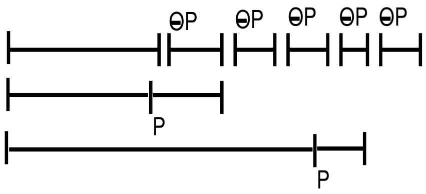
$$P\ ;\ \circleddash Q$$

holds if P is satisfied by an initial sub-interval and Q is satisfied by a final sub-interval, which overlaps by two states with the initial sub-interval (remember that even without past operators ; enforces a one state overlap). Thus, in order to interpret $\circleddash Q$ in this formula we have to consider the history arising from satisfaction of P.

Past operators also interact in an interesting way with projection. Consider first how past operators behave in the projection formula. For example, consider:

$$\circleddash P \ \ \textbf{proj} \ \ \textbf{True}$$

which requires a series of sub-intervals to be found, which satisfy $\circleddash P$. Importantly, the history for one such sub-interval is the sequence of states in the satisfying interval up to but not including the first state of the sub-interval. Consider the following depiction:

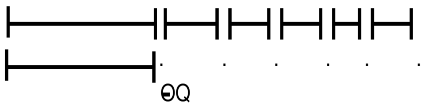


The higher time line indicates the complete satisfying interval, with its history shown to the left of the current interval. The current interval is divided into sub-intervals each of which satisfy $\circleddash P$. The two lower time lines show examples of induced intervals which must satisfy P in order that $\circleddash P$ is satisfied over a sub-interval. One such interval must exist for each sub-interval.

Now consider how past operators behave in the projected formula:

$$\textbf{True} \ \ \textbf{proj} \ \ \circleddash Q$$

A satisfying interval is depicted in the higher time line in the following diagram:



where $\ominus Q$ holds over a current interval resulting from extracting states from the satisfying interval in the usual way. But, importantly, the history of the satisfying interval is inherited by the extracted current interval. Thus, Q must be satisfied in an interval resulting from adding one state from the history to the extracted current interval.

4.4 Assignments

We can also, in a standard fashion derive three basic assignment operators for *Mexitl*. In the following definitions, v is a static variable which does not appear free in E_1 or E_2.

$$E_1 \leftarrow E_2 \;\equiv\; v = E_2 \;\Rightarrow\; \mathbf{fin}(E_1 = v)$$

$$E_1 \mathrel{\bigcirc\!\!=} E_2 \;\equiv\; v = E_2 \;\Rightarrow\; \bigcirc(E_1 = v)$$

$$E_1 := E_2 \;\equiv\; \mathbf{skip} \;\wedge\; E_1 \mathrel{\bigcirc\!\!=} E_2$$

Notice that $\mathrel{\bigcirc\!\!=}$ does not define the length of the interval over which it holds. For example, $x \mathrel{\bigcirc\!\!=} 2$ could hold in an interval of 5 states where x has the value 2 in the second state. This is in contrast to the operator $:=$.

We can also define multiple assignment operators,

$$\begin{array}{rcl} E_1, ..., E_n \; op \; F_1, ..., F_n & \equiv & E_1 \; op \; F_1 \;\wedge\; .... \;\wedge\; E_n \; op \; F_n \\ & & \text{for } op \in \{\leftarrow, \mathrel{\bigcirc\!\!=}, :=, =\} \end{array}$$

4.5 Repeated Assignments

Repeated assignments can be derived from the basic assignment operators.

$$E_1 \;\approx\; E_2 \;\equiv\; \Box(E_1 = E_2)$$

$$E_1 \;\mathbf{gets}\; E_2 \;\equiv\; \mathbf{keep}(E_1 \mathrel{\bigcirc\!\!=} E_2)$$

$$\mathbf{stable}\; E \;\equiv\; E \;\mathbf{gets}\; E$$

4.6 Projection and Loops

We can derive two useful variants of projection, P^* and **uproj**.

$$P^* \equiv P \ \textbf{proj} \ \textbf{True}$$

This operator is called chop star, although it has also been referred to as *loop* P.

$$P \ \textbf{uproj} \ Q \equiv \neg(P \ \textbf{proj} \ \neg Q)$$

uproj , known as universal projection [16], is the dual of projection.

There are a number of iteration operators that are classically found in interval temporal logics. We derive each of these here.

$$\textbf{for } n \textbf{ times do } P \equiv P \ \textbf{proj} \ \textbf{len}(n)$$

Notice P **proj** $\textbf{len}(0) \Leftrightarrow$ **empty**.

$$\textbf{while } P \textbf{ do } Q \equiv (P \wedge Q)^* \wedge \textbf{fin}(\neg P)$$

$$\textbf{for } i := 1 \textbf{ to } E \textbf{ do } P \equiv i := 1 \textbf{ ; while } i \leq E \textbf{ do } (P \wedge i \leftarrow i + 1)$$

assuming i is not changed in P.

$$\textbf{repeat } P \textbf{ until } Q \equiv P \textbf{ ; while } \neg Q \textbf{ do } P$$

$$P \textbf{ when } Q \equiv (\neg \Diamond Q) \vee (\textbf{halt}(Q) \textbf{ ; } (\bigcirc \textbf{halt}(Q) \ \textbf{proj} \ P) \textbf{ ; keep} \bigcirc(\neg Q))$$

$$P \textbf{ upon } Q \textbf{ do } P' \textbf{ else } P'' \equiv (\textbf{if } Q \textbf{ then } P' \textbf{ else } P'') \ \textbf{proj} \ P$$

Duan [4] gives a variant of projection, **prj**, which differs from **proj** in two ways.

1. An explicit sequence of intervals is given into which the main interval must be chopped, rather than chopping the interval into an arbitrary number of intervals all satisfying the same formula as is the case for **proj**.
2. **prj** allows overruns in either the projection sequence or the projected formula.

Duan's **prj** can be defined in *Mexitl*. Consider:

$$(P_1, P_2) \textbf{ prj } Q \equiv (\exists x \leq 2)(x = 0 \wedge (P \textbf{ proj } (Q; \textbf{True})))$$

where the formula P is defined by

$$\begin{aligned} P \equiv \ & (x = 0 \Rightarrow P_1 \wedge \textbf{halt}(x = 1)) \wedge \\ & (x = 1 \Rightarrow P_2 \wedge \textbf{halt}(x = 2)) \wedge \\ & (x \geq 2 \Rightarrow \textbf{len}(1) \wedge \textbf{halt}(x = 2)) \end{aligned}$$

We can see that P is equivalent to P_1 initially, as x is required to be zero initially, it is then equivalent to P_2 and subsequently it is equivalent to $\mathbf{len}(1)$. The result of the projection is thus as required. We expect the projected interval to satisfy $(Q; \mathbf{True})$, which allows overrun.

It is not possible to express the **proj** operation in terms of **prj** however. For instance, we see no way of expressing the fact that an interval is of even length using **prj**; it is expressed by $len(2)$ **proj True**. The reason that **prj** is inadequate is that it preserves boundedness: if there is an upper bound on the size of the intervals satisfying the projection formulas then there is on the size of the interval satisfying the **prj** itself; this is not the case with **proj**, as exemplified by the characterisation of evenness above.

It remains to verify that our definition of **prj** above treats the past in the way that it is defined to by Duan. We believe that methods similar to those which relate his and our 'chop in the past' will also apply here.

4.7 Metric Temporal Logic Operators

It is also possible to derive Metric Temporal Logic [14] style bounded temporal operators. We have the following derivations:

$$\Diamond_{[t,r]} P \;\equiv\; t \leq \mathbf{mylen} \leq r \;;\; P$$

$$\Diamond_t P \;\equiv\; \Diamond_{[t,t]} P$$

$$\Box_{[t,r]} P \;\equiv\; \neg \Diamond_{[t,r]} \neg P$$

$$\diamondminus_{[t,r]} P \;\equiv\; x = \mathbf{mylen} \;\wedge\; \diamondminus(P \;\wedge\; x + t \leq \mathbf{mylen} \leq x + r)$$

$$\diamondminus_t P \;\equiv\; \diamondminus_{[t,t]} P$$

$$\boxminus_{[t,r]} P \;\equiv\; \neg \diamondminus_{[t,r]} \neg P$$

Clearly, in contrast to some other definitions of these operators, such as [5], our definitions are over a discrete time domain.

4.8 Miscellaneous

This subsection contains a number of other miscellaneous interval temporal logic operators.

$$P \;\mathbf{within}\; Q \;\equiv\; \neg \mathop{\Diamond\kern-0.62em{\scriptscriptstyle\mathrm{i}}\kern0.3em}(Q \;\wedge\; \neg \Diamond P)$$

$$P \;\mathbf{timeout}\; t \; Q \;\equiv\; ((\mathbf{len}(t) \;\wedge\; \Box\neg P) \;;\; Q) \;\vee\; (\mathbf{lesseq}(t) \;\wedge\; \mathbf{halt}(P))$$

These two definitions also appear in Hale [8].

The **prefix** operator cannot be derived from any of the other operators, and indeed it is problematic since it makes satisfiability undecidable, since we can reduce the halting problem to a case of **prefix**. On the other hand, **prefix** over sequences of actions is unproblematic. In a similar way, disable (also called interrupt or trap) is derivable if the body of the disable is restricted to be a sequence of indexed actions, disabling reduces to a while loop. For example,

$$i := 1 \; ; \; \textbf{while} \; (i \leq n \;\wedge\; \neg Q) \; \textbf{do} \; (playFrame(i) \;\wedge\; i \leftarrow i + 1)$$

will either play all n frames or disable the playing of frames when Q holds.

We can also obtain fast and slow using for loops. For example,

$$\textbf{for} \; i := 1 \; \textbf{to} \; n \; div \; 2 \; \textbf{do} \; playFrame(2 * i)$$

$$\textbf{for} \; i := 1 \; \textbf{to} \; 2 * n \; \textbf{do} \; playFrame(i \; div \; 2)$$

5 Applying *Mexitl*

In this section we show how the *Mexitl* formalism of the previous section can be employed for multimedia documents. We do this by first showing how the general requirements of Section 2.1 are satisfied, and then by showing how the individual functional requirements are met by providing a complete formal specification of the Beethoven problem of Section 2.2.

5.1 Specifying General Examples

We first refer to the general requirements introduced in Section 2.1. The first of these concerns media item display. What we termed standard display is a fairly trivial consequence of the definition mechanism we have chosen. Since all media items are represented as intervals, we may with perfect consistency regard a, say, either as a name representing a particular media item, or as a formula representing the display of that media item.

King [12] discussed at length the question of displaying media items at different rates. These rate changes can be represented by a particular form of interval projection referred to as a *multiplication*, where the projector is of the form $\textbf{len}(n)$. For example

$$\textbf{len}(2) \;\; \textbf{proj} \;\; a$$

would display a at half speed; each successive interval of length 1 in a is replaced by an interval of length 2, doubling the length of the entire interval. King [12] presents other examples.

With regard to composition, serial composition is based on the chop operator (;), while parallel composition makes use of the conjunction ($\wedge$). For the serial and parallel composition of two items a and b (or indeed sets of constraints), we would have, respectively,

$$a \; ; \; b$$

$$a \wedge b$$

Frequently it is the case that items a and b will not be independent, but b will require some synchronisation information from item a. This synchronisation over parallel composition is illustrated by Example 3 in the Beethoven Problem, where the title displays depend upon temporal information from Example 1, (the endpoints of the respective movements), as well as Examples 6 through 10, which are more complex instances. These matters are all covered in the formal *Mexitl* specification of the next section. Item b may also depend on a in the case of serial composition; this dependency will be illustrated below in our discussion of past operators.

5.2 Specification of Beethoven Example

In this example, we elaborate on the earlier specification of Beethoven's Fifth Symphony.

We begin with a type definition of a symphony with four movements.

Beethoven's Fifth Symphony comprises four movements.

type symphony4 = (movement, movement, movement, movement)
type movement = interval – a finite interval of actions

This definition of a movement allows us to use m_1, m_2, etc. both as intervals in *Mexitl* formulae, and also as sequences of primitive actions into which we can index. In particular, we distinguish two elements of these sequences,

$$m_i[1] \text{ and } m_i[m_i.\mathrm{last}]$$

where last is a predefined constant stating how many actions there are in such a sequence.

Beethoven's Fifth Symphony is a particular instance of a symphony with four movements, which is expressed as follows:

b5 :: symphony4
m_1, m_2, m_3, m_4 :: movement

The name b5 is appropriately initialised.

b5 = (m_1, m_2, m_3, m_4)

The constants m_1, m_2, m_3 and m_4 would, in reality, have to be set to the contents of the corresponding movements; this matter lies outside the scope of this paper.

We now proceed to give a formal specification of each of the ten examples comprising the Beethoven problem.

1. **Requirements:** Play the four movements of the symphony in sequence with a gap of 20 seconds between each movement.

Formal Specification:

$$S_1 \equiv m_{1\,M} \text{ ; } (\textbf{for } i := 2 \textbf{ to } 4 \textbf{ do } \Diamond_{20''\,M}\ m_{i\,M})$$

where M is the set of audio actions forming m_1, m_2, m_3 and m_4.
In this specification, we have used a for-loop for the later movements of the symphony. We have framed each movement against any (other) actions from the set of actions comprising the four movements. We are, in effect, overloading the symbols m_i to be both of type "movement" and "set of actions" – the actions comprising the movements. The bounded temporal operator $\Diamond_{20''}$ is used to provide the 20 second intervals, and it is also framed to prevent any of the actions M from any of the movements from playing during these intervals.

2. **Requirements:** Before the first movement begins, play an audio which announces the name of the symphony, the composer, and the orchestra. Two seconds after this audio starts, display a video still of Beethoven. Stop the video still display as the first movement starts. After the last movement, wait 3 seconds and display for 30 seconds a video of Ludwig van Beethoven and then, after a further 5 seconds, display information about how to order this presentation.

Definitions:

type still = action
type audio = interval
type video = interval
a_1 :: audio
Ludwig :: still
lvb :: video
orderinginfo :: still
$a_1 = \cdots$
Ludwig $= \cdots$
lvb $= \cdots$
orderinginfo $= \cdots$

Formal Specification:

$$\begin{aligned} S_2 \equiv\ & (a_{1\,A} \wedge (\Diamond_{2''\,S}\text{Ludwig}^*_S)) \text{ ; } S_1 \text{ ; } \\ & (\Diamond_{3''\,V}\text{lvb}_V) \text{ ; } (\Diamond_{5''\,S}(\text{OrderingInfo}^*_S \wedge \textbf{len}(30''))) \end{aligned}$$

where A is the set of audio actions, S is the set of still image actions, and V is the set of video image actions.
As mentioned earlier, there are several alternative ways in which these sets of constraints may be composed. In this formal specification, we have arbitrarily chosen to compose the items at the beginning and end of the presentation in a serial fashion. The first of these is a parallel composition of two sub-channels, the audio and the delayed video-still. The video still itself is specified as the repetition (*) of an elementary one-state action displaying a picture of Beethoven. We specify the items to appear after the music as a serial composition using chop ; and a bounded temporal

operator $\Diamond_t$. The various items introduced in the specification are framed against other actions of their respective classes.

3. **Requirements:** At the start of each movement display an appropriate title for 5 seconds. Each title is a video still. Repeat the 5 second display of each title every 3 minutes during the corresponding movement.

Definitions:

t_1, t_2, t_3, t_4 :: still – the four titles

$t_1 = \cdots$

$t_2 = \cdots$

$t_3 = \cdots$

$t_4 = \cdots$

Formal Specification:

$$R_{=5''} \equiv ({t_i^*}_T \ \wedge \ \mathbf{len}(5'')) \quad \text{– repeat title for exactly 5 seconds}$$

$$R_{<5''} \equiv ({t_i^*}_T \ \wedge \ \mathbf{less}(5'')) \quad \text{– repeat title for less than 5 seconds}$$

$$L_{=2'55''} \equiv \mathbf{len}_T(2'55'') \quad \text{– an interval of length exactly 2' 55”}$$

$$L_{\leq 2'55''} \equiv \mathbf{lesseq}_T(2'55'') \quad \text{– an interval of length at most 2' 55”}$$

$$S_3 \equiv S_2 \ \wedge \ \Diamond_T \ \mathbf{for}\ i := 1 \ \mathbf{to}\ 4\ \mathbf{do}\ (\mathbf{halt}_T(m_i[1]) \ ; \\ (((R_{=5''} \ ; \ L_{=2'55''})^* \ ; (R_{<5''} \ \vee \ (R_{=5''} \ ; \ L_{\leq 2'55''}))) \ \wedge \\ \mathbf{halt}(m_i[m_i.\mathrm{last}])))$$

where T is the set $\{t_1, t_2, t_3, t_4\}$.
For each of the four movements m_i, $i = 1, 2, 3, 4$, we first synchronise with the beginning of the movement m_i using $\mathbf{halt}(m_i[1])$. Then, by specifying $(R_{=5''} \ ; \ L_{=2'55''})^* \ \equiv \ (({t_i^*}_T \ \wedge \ \mathbf{len}(5'')) \ ; \ \mathbf{len}_T(2'55''))^*$, we can display the title t_i for 5 seconds every 3 minutes. Next, we handle any leftover time of length less than 3 minutes at the end of the movement using a disjunction of two options. The choice between the two options is based on the length of the remaining time. If we have less than 5 seconds left over, then we play the title during the leftover time using $R_{<5''} \ \equiv \ ({t_i^*}_T \wedge \mathbf{less}(5''))$; otherwise, we play the title for 5 seconds and wait through any remaining time using $(R_{=5''} \ ; \ L_{\leq 2'55''}) \ \equiv \ (({t_i^*}_T \ \wedge \ \mathbf{len}(5'')) \ ; \ \mathbf{lesseq}_T(2'55''))$. The conjunction with $\mathbf{halt}(m_i[m_i.\mathrm{last}])$ ensures that these additions last exactly as long as movement m_i. Then we repeat the display with the next movement. Actions are framed against the corresponding class of actions so as to avoid unwanted behaviour. Note that the earlier framing of actions in S_1 ensures, for example, that the occurrence of the first action in m_1 is uniquely defined.

4. **Requirements:** The audio introduced in Example 2 is actually in three parts, corresponding to the name of the symphony, the composer, and the orchestra. During this audio display,

   either display a rolling text item containing the same information (assume this item has the same length as the audio)

or display, in sequence, three video stills containing the same information for the appropriate time lengths.

Definitions:

type announce = (audio, audio, audio)
type rollingtext = interval
a_1, a_2, a_3 :: audio
a :: announce
rt :: rollingtext
f_1, f_2, f_3 :: still
a = (a_1, a_2, a_3)
$a_1 = \cdots$
$a_2 = \cdots$
$a_3 = \cdots$
rt $= \cdots$
$\vdots$

Formal Specification:

$$S_2' \equiv ((\textbf{for } i := 1 \textbf{ to } 3 \textbf{ do } a_{i\,A}) \wedge \Diamond_{2''}\text{Ludwig}^*) \textbf{ ; } \cdots$$
$$S_4 \equiv S_3 \wedge \diamondplus_{\{\text{rt}\}}(\text{rt}_R \vee \textbf{for } i := 1 \textbf{ to } 3 \textbf{ do } (f_i^* \wedge \textbf{halt}(a_i[a_i.\text{last}])))$$

where $\{\text{rt}\}$ is the set of actions corresponding to rt and R is the set of all rolling text items.

We use disjunction ($\vee$) to choose between the rolling text item rt and the three video stills f_1, f_2 and f_3. The stills immediately follow one another and the end of the display of each still is synchronised with the end of the corresponding audio display.

The framing here is more problematical and fairly subtle. Consider the case in which it is decided to display the three video stills. This will not work as a direct composition with the original S_2 in Example 2 above, since in that specification we deliberately framed against any other video stills occurring in that prefix interval. Hence, we have provided an alternative version S_2' without framing on the item Ludwig. For similar reasons, we have restricted the framing in the rolling text item to actions corresponding to rolling text items. The framing on the $\diamondplus$ is even more restrictive; we frame against just those actions occurring in the specific rolling text item at hand; this is to prevent this specific rolling text item (or any of its constituent actions) from occurring other than where we explicitly say it should.

This example illustrates an important consideration regarding framing and regarding composition of channels. If P is framed against the set S (P_S), then any conjunction $P_S \wedge Q$ is thereby framed against the same set, and therefore Q is also so framed. This places restrictions on the actions that may be specified within Q. This sort of restriction is clearly inappropriate in an authoring system where the author of Q may know nothing about the internals of P. This question is akin to questions of modularity, inheritance, information hiding, and programming in the large, which are familiar ones in the domain of software design and engineering. They are currently under study in our own work.

5. **Requirements:** The twenty second gap between the second and third movements should be replaced by a video/audio display describing the third movement. If the video/audio display takes longer than twenty seconds, truncate the display to twenty seconds.

   Definitions:

   type videoaudio = interval
   va :: videoaudio
   va = $\cdots$

   Formal Specification:

$$
\begin{aligned}
S_5 \equiv S_4 \;\wedge\; & {\Diamond\!\!\!\!\!{\scriptstyle\mathrm{a}}}\,_{\{\mathrm{va}\}}(\mathbf{beg}(\circleddash\; m_2[m_2.\mathrm{last}]) \;\wedge\; \mathbf{halt}(\bigcirc m_3[1]) \wedge \\
& ((\Diamond\!\!\!\!\!{\scriptstyle\mathrm{i}}\;\, \mathrm{va}_{\mathrm{VA}}) \;\vee\; (\exists\, t \leq \mathrm{va.last})\; \mathbf{for}\; i \;:=\; 1 \;\mathbf{to}\; t \;\mathbf{do}\; \mathrm{va}[i]_{\mathrm{VA}}))
\end{aligned}
$$

   where {va} is the set of all actions corresponding to va and VA is the set of all video/audio actions.

   We synchronise the display of the audio-video with an interval where the state previous to the first contains the last action of movement 2, and end the display when the next state contains the first action of the third movement. In the 20 second interval we either play the entire audio-video, which, if it is less than 20 seconds, will require a (strictly) initial interval, or play as much of it as we can, using a for loop with a bounded existential quantifier. This all occurs in some arbitrary interval of the entire presentation.

The remaining five parts of the Beethoven problem examples are illustrations of what we have termed scripting. Since the point of these is to illustrate use of the operators such as filter and when, we have chosen to omit all considerations of framing in these specifications, as they will confuse matters unduly.

We require two other assumptions, both of which are perfectly reasonable given the application area of interest to us.

First, we assume that the various conditions (staccato, viola, etc.) correspond to state variables whose values are supplied by the application. Thus, for example, staccato will be true in exactly those states which correspond to a staccato note in the music[3].

Second, we assume the existence of a number of primitive actions (state constants) such as red for colouring the screen, purple for the display of a purple band, and so forth.

6. **Requirements:** At the first sound of a viola, display a purple band for 3 seconds.

[3] This is a simplification in at least two ways. First, the concept of a staccato note in the context of a symphony orchestra playing a number of differing parts, is troublesome. Second, we are assuming that each note in the music occupies exactly one state in the interval. We apologise to Beethoven for both of these.

Formal Specification:

$$S_6 \equiv S_5 \ \wedge \ \mathbf{if} \diamond \text{viola } \mathbf{then} \ \diamondsuit \ (\mathbf{halt}(\text{viola}) \ ; \ (\text{purple}^* \ \wedge \ \mathbf{len}(3'')))$$

This is a relatively simple composition, and should be self- explanatory.

7. **Requirements:** During the first crescendo passage, display a mammoth.

Formal Specification:

$$C \equiv (\exists x \leq \text{maxVol}) \ ((\text{vol} = x) \ \wedge \ (\ominus(\text{vol} < x) \ \vee \ \bigcirc(\text{vol} > x)))$$
$$S_7 \equiv S_6 \ \wedge \ \mathbf{if} \diamond C \ \mathbf{then} \ \diamondsuit \ (\mathbf{halt}(C) \ ; \ (\mathbf{halt}(\neg C) \ \wedge \ \text{mammoth}^*))$$

First, we need a condition to test if we are in a crescendo passage. In the above formula, C tests if we are in such a passage by comparing the current volume with the volume in the previous and next states.
With this, the composition is fairly self-evident.

8. **Requirements:** During each crescendo passage of the first movement, display a looped video tape of a bug climbing an inclined plane.

Formal Specification:

$$\begin{aligned} S_8 \equiv S_7 \ \wedge \ \diamondsuit \ (\ & \mathbf{beg}(\ominus \ a_3[a_3.\text{last}]) \ \wedge \\ & ((C \ \ \mathbf{filter} \ \ (\text{video}^* \, ; (\exists t \ \leq \ \text{video.last}) \, \mathbf{for} \, i \ := \ 1 \, \mathbf{to} \, t \, \mathbf{do} \, \text{video}[i])) \\ & \wedge \ \mathbf{halt}(\bigcirc m_1[m_1.\text{last}]))) \end{aligned}$$

where C is as defined in Example 7.
We use a similar technique to Example 4 to restrict ourselves to the subinterval comprising the first movement. The filter operation allows us to concatenate the crescendo passages as a single interval, over which we then play the video (of the climbing bug) as often as we can. Note that this formulation allows us to play this video four and a third times, if that is what is required.

9. **Requirements:** Count the number of C minor chords.

Formal Specification:

$$S_9 \equiv S_8 \ \wedge \ \text{C minor} \ \ \mathbf{filter} \ \ (\exists i \leq (\mathbf{mylen} + 1))((i = 1) \ \wedge \ (i \, \mathbf{gets} \, i + 1))$$

This example is similar to Example 8, and should be self-explanatory.

10. **Requirements:** During each staccato note of the first movement, flash the screen red.

Formal Specification:

$$S_{10} \equiv S_9 \ \wedge \ \diamondsuit (\mathbf{beg}(m_1[1]) \ \wedge \ (\text{red} \, \mathbf{when} \, \text{staccato}) \ \wedge \ \mathbf{halt}(m_1[m_1.\text{last}]))$$

The synchronisation to restrict to the first movement is identical to that in Example 8.

5.3 Serial Composition and Past Operators

In the foregoing we have referred to the use of this formalism as the basis of an authoring tool, and commented on the issues associated with parallel composition and framing in this regard. It is worth remarking that these issues are non-existent in the case of serial composition using the ; operator. P_A implies nothing about the framing of Q in the formula P_A ; Q.

Further, the past operators in the formalism allow composition, by allowing a coauthor to create piece a of a document a ; b without needing to know what information about piece a is required by the other piece b that occurs later in the overall document and is created by another author.

For example, suppose that a text display P of the definition of "pianissimo" should be displayed exactly once and it is possible that a displayed the definition and it is also possible that a did not present the definition. Instead of a having to signal b in some way if a presents P, the author of b can use $\diamondminus$(sometime in the past) to specify

$$\mathbf{if}\ \neg\diamondminus P\ \mathbf{then}\ P$$

to ensure that P is displayed if it was not displayed previously.

For a further example, suppose that a map display may have been turned on by a and not been turned off. If b needs the map display on, the author of b can use $\mathcal{S}$ (since) to specify

$$(\neg\ \text{off})\ \mathcal{S}\ \text{on}$$

to check if the map display was left on by a.

$\ominus$ is useful for finding a transition point, that is, a point at which some property P becomes true: $\ominus\neg P \wedge P$. See Example 7 above for a specific example.

6 Towards A Logic For Reasoning About Intervals

This section introduces a proof system for *Mexitl*, whose syntax and semantics were given earlier. The system described here is consistent but we make no claims about its completeness. The axioms listed here are for the core language only, but on the basis of these rules we can deduce rules for the derived operators. Examples of this will be given as we go along.

The rules have a number of purposes:

- Implication formalises refinement, so that a rule such as [$P6$] shows that a sound refinement of $(P \vee Q)$ **proj** R is given by a choice between (P **proj** R) and (Q **proj** R).
- We can use the rules to reason about specifications.
- Using the rules we can find a normal form for *Mexitl* formulas (along the lines of Gabbay's work [7]) which forms a component part of the tableau decision procedure and implementation of the language which is currently under development.

The core constructs are actions and **;**, **mylen**, **proj** , $\mathcal{S}, \ominus, \tilde{;}$, **filter** and we examine properties of these in turn. Properties are of two sorts: some we take as axiomatic, and the rest we prove from the axioms and rules. We argue that the axioms are *sound*, that is that they are validated by the semantics in Section 3.4.

In a number of the rules we state that formulas are *local* in that their truth is determined by the point of reference of the interval over which they are interpreted. Formulas which contain no temporal operators will have this property, for instance. We also introduce the notions of *rigid* formulas which have all their models of the same current length and *deterministic* formulas which have models with unique current intervals.

6.1 Non-Temporal Connectives

In the core language we have the connectives **False**, $\Rightarrow$ and bounded existential quantification from which we can define the other non-temporal propositional logical connectives. We assume that any propositional tautology is valid, and also include the rule of inference *Modus Ponens* which is stated thus

$$[MP] \qquad \frac{A \qquad A \Rightarrow B}{B}$$

Rules written in this form should be read 'if the hypotheses above the line are proven then the conclusion can be inferred'. We also assume standard axioms and rules for bounded existential quantification.

6.2 Chop: Sequencing In Time

Read imperatively, chop (**;**) puts P and Q in sequence: an interval satisfies P**;**Q if it can be split into two contiguous sub-intervals satisfying P and Q respectively.

Chop satisfies a number of algebraic identities:

$$[C1] \qquad (P;(Q;R)) \Leftrightarrow ((P;Q);R)$$

$$[C2] \qquad ((P \vee Q);R) \Leftrightarrow ((P;R) \vee (Q;R))$$

$$[C3] \qquad (P;(Q \vee R)) \Leftrightarrow ((P;Q) \vee (P;R))$$

$$[C4] \qquad P \Rightarrow (P;\mathbf{True})$$

$$[C5] \qquad P \Rightarrow (\mathbf{True};P)$$

$$[C6] \qquad \mathbf{False} \Leftrightarrow (P;\mathbf{False})$$

$$[C7] \qquad \mathbf{False} \Leftrightarrow (\mathbf{False};P)$$

The empty interval provides an identity for chop; this is specified in laws $[L1]$ and $[L2]$ in the following section.

For local formulas S we have the converse of $[C4]$:

$$[C8] \qquad S \Leftrightarrow (S;\mathbf{True}) \qquad [S \text{ local}]$$

Finally we introduce a deduction rule associated with chop. It says that we can replace the components of a chop with any of their logical consequences.

$$[RC1] \quad \frac{P \Rightarrow P' \qquad Q \Rightarrow Q'}{(P;Q) \Rightarrow (P';Q')}$$

If we view '$\Rightarrow$' as a refinement relation, with $A \Rightarrow B$ meaning 'A is a refinement of B', then the rule $[RC1]$ states how a chop can be refined, namely by refining either of its components.
As we look at other logical constructions we shall introduce further rules which explain the ways in which they interact with chop.

The Soundness Of $[C3]$ As an illustration of how we show the soundness of the axioms we look at the axiom $[C3]$ now. The left-hand side, $P;(Q \vee R)$ is true over the computation structure (σ, i) if and only if

$$\exists k \in \mathcal{N} \ (i \leq k \leq |\sigma| \text{ and } ([\sigma]^k, i) \models P \text{ and } (\sigma, k) \models (Q \vee R))$$

Since $(\sigma, k) \models (Q \vee R)$ from the semantics of disjunction we know that either $(\sigma, k) \models Q$ or $(\sigma, k) \models R$. In the first case we have $(\sigma, i) \models (P;Q)$ and in the second $(\sigma, i) \models (P;R)$. In either case $(\sigma, i) \models (P;Q) \vee (P;R)$. This establishes the soundness of the bi-implication $[C3]$ in the forward direction; the verification of soundness in the backward direction is similar.

6.3 Lengths: mylen, lesseq and len

Our formal system involves the variable **mylen**, which is interpreted as the length of the interval in question. We can axiomatise this directly, or we can give rules indirectly if we define

$$\mathbf{lesseq}(E) \Leftrightarrow (\mathbf{mylen} \leq E)$$

and

$$\mathbf{len}(E) \Leftrightarrow (\mathbf{mylen} = E)$$

In what follows we discuss the properties of these derived operations, and leave it as an exercise to translate these rules into rules which explicitly talk about **mylen**.

The formula $\mathbf{lesseq}(E)$ holds over any structure the length of whose current interval is less than or equal to the value of E over that interval, while $\mathbf{len}(E)$ requires that the length of the current interval is exactly that value. Rules for these length formulas express their combination with logical connectives and particularly with chop, '**;**'.

$$[L1] \quad (\mathbf{lesseq}(0);P) \Leftrightarrow P$$

$$[L2] \quad (P;\mathbf{lesseq}(0)) \Leftrightarrow P$$

note that $\mathbf{lesseq}(0)$ and $\mathbf{len}(0)$ are both equivalent to **empty**.

$$[L3] \quad \mathbf{lesseq}(Sn) \Leftrightarrow ((\mathbf{len}(1);\mathbf{lesseq}(n)) \vee \mathbf{lesseq}(0))$$

where $1 \equiv (S0)$. Symmetrically with $[L3]$ we have

$$[L4] \quad \mathbf{lesseq}(Sn) \Leftrightarrow ((\mathbf{lesseq}(n);\mathbf{len}(1)) \vee \mathbf{lesseq}(0))$$

and

$$[L5] \quad \mathbf{lesseq}(n+m) \Leftrightarrow (\mathbf{lesseq}(n);\mathbf{lesseq}(m))$$

$$[L6] \quad (\mathbf{lesseq}(n) \vee \mathbf{lesseq}(0)) \Leftrightarrow \mathbf{lesseq}(n)$$

$[L5]$ follows from $[L1]$ and $[L3]$ by induction on n, and $[L6]$ follows from $[L3]$ by a case analysis over whether n is zero or a successor.

Now we turn to **len**, which we saw defined above. We can introduce **len** axiomatically by the two rules which follow if we wish:

$$[L7] \quad \mathbf{len}(Sn) \Leftrightarrow (\mathbf{len}(n);\mathbf{len}(1))$$

$$[L8] \quad \mathbf{len}(Sn) \Leftrightarrow (\mathbf{len}(1);\mathbf{len}(n))$$

We can express the fact that the length of the current interval is greater than or equal to k by means of the formula $\mathbf{len}(k);\mathbf{True}$; the following axiom expresses the fact that the current interval either has length less than or equal to k or length greater than k. This analysis is useful in characterising the properties we need of 'eventually', $\Diamond$, for instance.

$$[L9] \quad \mathbf{lesseq}(k) \vee (\mathbf{len}(Sk);\mathbf{True})$$

In **len** we have a formula which is *rigid* in that the current interval of every computation structure satisfying it has the same length. This gives us the property that

$$[L10] \quad (\mathbf{len}(k);P) \wedge (\mathbf{len}(k);Q) \Rightarrow (\mathbf{len}(k);(P \wedge Q))$$

The analogous property

$$(R;P) \wedge (R;Q) \Rightarrow (R;(P \wedge Q))$$

holds of any rigid predicate R, where we say that R is rigid if and only if

$$\models (R;\mathbf{len}(a)) \wedge (R;\mathbf{len}(b)) \Rightarrow a = b$$

Our final rule for **len** gives the defining property of the expression **myLen**

$$[L11] \quad \mathbf{len}(\mathbf{myLen})$$

The importance of this formula lies in the fact that it identifies the length of the current interval as being the value of the expression **myLen**; we can then use this expression in other expressions or prove results by induction over its value. Given the definition of **len** this assertion is, of course, a tautology.

6.4 Proving Properties For Defined Operators

Given the definitions of **lesseq**, **len** and **;** we are in a position to prove many familiar properties of operators such as $\Box$, $\Diamond$ and $\bigcirc$. Among these is

$$[T1] \qquad \bigcirc P \Rightarrow \Diamond P$$

which is short for $(\mathbf{len}(1);P) \Rightarrow (\mathbf{True};P)$. As **True** is a consequence of $\mathbf{len}(1)$ the rule $[RC1]$ gives the result. Another result is

$$[T2] \qquad \Box P \Rightarrow \Box(P \Rightarrow Q) \Rightarrow \Box Q$$

which when expanded becomes

$$\neg(\mathbf{True};\neg P) \Rightarrow \neg(\mathbf{True};\neg(P \Rightarrow Q)) \Rightarrow \neg(\mathbf{True};\neg Q)$$

Taking the contrapositive of this, we are required to prove

$$(\mathbf{True};\neg Q) \Rightarrow ((\mathbf{True};\neg P) \vee (\mathbf{True};(P \wedge \neg Q)))$$

We show this now.

$$\begin{array}{ll}
(\mathbf{True};\neg Q) & \\
\Leftrightarrow (\mathbf{True};((\neg P \vee P) \wedge \neg Q)) & \text{by logic} \\
\Leftrightarrow (\mathbf{True};((\neg P \wedge \neg Q) \vee (P \wedge \neg Q))) & \text{by logic} \\
\Leftrightarrow (\mathbf{True};(\neg P \wedge \neg Q)) \vee (\mathbf{True};(P \wedge \neg Q)) & \text{by } [C3] \\
\Rightarrow (\mathbf{True};\neg P) \vee (\mathbf{True};(P \wedge \neg Q)) & \text{by } [RC1]
\end{array}$$

Finally we have

$$[T3] \qquad \Box(P \Rightarrow \bigcirc P) \Rightarrow P \Rightarrow \Box P$$

After translating out the definitions and taking the contrapositive of the resulting formula (as we did for $[T2]$ above) we have the goal

$$(\mathbf{True};\neg P) \Rightarrow (\neg P \vee (\mathbf{True};(P \wedge \neg(\mathbf{len}(1);P))))$$

On the basis of $\neg P$ begin true at some point in the future we need to prove either that it will hold at the current instant or at some later point there will be a last stage at which P is true. The proof of this result – which can be seen as a version of mathematical induction – relies on the characterisation $[L11]$ of the length of the interval before $\neg P$ is true; induction over that length gives the result.

6.5 Projection

A projection P **proj** Q holds of an interval when that interval can be sub-divided into a series of sub-intervals each of which satisfies P – we call P the projection formula – and a new interval formed from the end points of the sub-intervals satisfies Q, which we also call the projected formula. A careful examination of the semantics shows a number of special cases which are apparent in the substantial number of formulas which we see below. Note that in the following it is assumed that **proj** binds more tightly than the non-temporal logical operators and **;**.

$$[P1] \quad P \Rightarrow (\mathbf{True}\ \mathbf{proj}\ P)$$

$[P1]$ holds because the subdivision of P into unit intervals makes the projected interval the same as the original. A similar argument justifies $[P2]$

$$[P2] \quad P \Leftrightarrow (\mathbf{len}(1)\ \mathbf{proj}\ P)$$

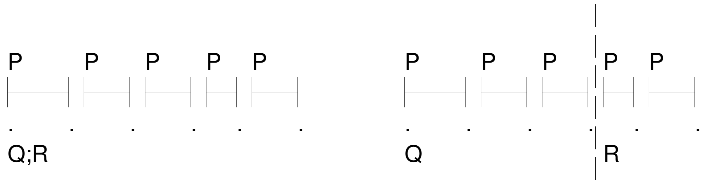
An interval satisfying P can be projected over the formula P – this is expressed by saying that the resulting interval satisfies **True**:

$$[P3] \quad P \Rightarrow (P\ \mathbf{proj}\ \mathbf{True})$$

If the interval resulting from a projection satisfies a chop then we can see this as a chop of projections and vice versa.

$$[P4] \quad (P\ \mathbf{proj}\ (Q\mathbf{;}R)) \Leftrightarrow ((P\ \mathbf{proj}\ Q)\mathbf{;}(P\ \mathbf{proj}\ R))$$

A picture serves to justify $[P4]$:



A similar result holds for disjunction; again we can illustrate the proof or argue from the semantics as we did earlier for $[C3]$.

$$[P5] \quad (P\ \mathbf{proj}\ (Q \vee R)) \Leftrightarrow ((P\ \mathbf{proj}\ Q) \vee (P\ \mathbf{proj}\ R))$$

Any sub-division of an interval in which all sub-intervals satisfy the same formula P (or Q) will serve as a sub-division for the formula $P \vee Q$. This means that we have

$$[P6] \quad ((P\ \mathbf{proj}\ R) \vee (Q\ \mathbf{proj}\ R)) \Rightarrow ((P \vee Q)\ \mathbf{proj}\ R)$$

The converse of this formula fails: a counter-example is instructive. The formula P **proj True** holds if we can sub-divide the interval in question into a series of sub-intervals satisfying P. If we take P as $\mathbf{len}(k)$ then the formula is equivalent to the statement that k is a divisor of the length of the interval. We can therefore see that the formula

$$(\mathbf{len}(2)\ \mathbf{proj}\ \mathbf{True}) \wedge \mathbf{len}(11)$$

can have no models; similarly

$$(\mathbf{len}(3)\ \mathbf{proj}\ \mathbf{True}) \land \mathbf{len}(11)$$

has no models. However

$$((\mathbf{len}(2) \lor \mathbf{len}(3))\ \mathbf{proj}\ \mathbf{True}) \land \mathbf{len}(11)$$

will be satisfiable using an appropriate combination of intervals of length 2 and 3. This shows that the converse of $[P6]$ is not valid.

Resuming the axiomatisation of **proj** we examine the case in which the projection formula is **False**, as in the formula **False proj** Q. We can only satisfy this formula over the computation structure (σ, i) when the projection produces no non-trivial intervals. Recalling the semantic description in Section 3.4 it is apparent that this happens when the number of end-points, m, is zero. In this case we satisfy the clause $i = \tau_0 = |\sigma|$ if and only if the interval is empty. The clause $\forall j < m \ldots$ is vacuously true since m is 0, and the final clause of the condition is equivalent to $(\sigma, i) \models Q$. We therefore have that

$$[P7] \qquad (\mathbf{False}\ \mathbf{proj}\ Q) \Leftrightarrow (\mathbf{len}(0) \land Q)$$

In refining a projection we may refine either the projection or the projected formula, so that we have the rule of inference

$$[RP1] \qquad \frac{P \Rightarrow P' \qquad\qquad Q \Rightarrow Q'}{(P\ \mathbf{proj}\ Q) \Rightarrow (P'\ \mathbf{proj}\ Q')}$$

Using $[P7]$ and $[RP1]$ we have

$$[P8] \qquad \mathbf{len}(0) \Rightarrow (P\ \mathbf{proj}\ \mathbf{True})$$

since by $[P7]$ $\mathbf{len}(0)$ is equivalent to **False proj True** and since $\mathbf{False} \Rightarrow P$ for any P we have $[P8]$ by $[RP1]$.

When can we project an interval and satisfy $\mathbf{len}(0)$? Again we appeal to the semantics, and see that this happens if and only if the number of end points m can be chosen to be 0, which is equivalent to $|\sigma| - i$ being zero. In this case the predicate P need not be satisfied, so we conclude that

$$[P9] \qquad (P\ \mathbf{proj}\ \mathbf{len}(0)) \Leftrightarrow \mathbf{len}(0)$$

When can we project an interval and satisfy $\mathbf{len}(1)$? If the interval has non-zero length it is clear that $P\ \mathbf{proj}\ \mathbf{len}(1)$ will hold exactly when the entire interval satisfies P. In the zero length case this is also the case; we leave the verification to the reader.

$$[P10] \qquad (P\ \mathbf{proj}\ \mathbf{len}(1)) \Leftrightarrow P$$

We next characterise when an interval satisfies P **proj True**. This means that we can subdivide the interval into zero or more subintervals, each of which satisfies P. Using the characterisation $[L9]$ we can give an expansion of such a projection revealing its loop-like nature:

$$[P11] \quad (P \ \mathbf{proj}\ \mathbf{True}) \Leftrightarrow (\mathbf{len}(0) \vee (P;(P \ \mathbf{proj}\ \mathbf{True})))$$

We argue thus:

$$\begin{array}{ll} P \ \mathbf{proj}\ \mathbf{True} & \\ \Leftrightarrow P \ \mathbf{proj}\ (\mathbf{len}(0) \vee (\mathbf{len}(1);\mathbf{True})) & \text{by } [L9] \\ \Leftrightarrow (P \ \mathbf{proj}\ \mathbf{len}(0)) \vee & \\ \qquad (P \ \mathbf{proj}\ (\mathbf{len}(1);\mathbf{True})) & \text{by } [P5] \\ \Leftrightarrow \mathbf{len}(0) \vee & \\ \qquad (P \ \mathbf{proj}\ \mathbf{len}(1));(P \ \mathbf{proj}\ \mathbf{True}) & \text{by } [P8], [P4] \\ \Leftrightarrow \mathbf{len}(0) \vee (P;(P \ \mathbf{proj}\ \mathbf{True})) & \text{by } [P10] \end{array}$$

We conclude with rules for **proj** which attempt to give a normal form for projections in terms of a 'chop normal form'. We can think of decomposing formulas into a form which distinguishes between the present instant and the remainder of the interval – its future. Such an expression takes the form

$$((Q \wedge \mathbf{len}(1));R)$$

At the first instant we require Q to be true; after that R holds. To ensure that Q does apply only to a single instant we require that it is local, as defined above. Now we have

$$[P13] \quad (P \ \mathbf{proj}\ ((Q \wedge \mathbf{len}(1));R)) \Leftrightarrow ((P \wedge Q);(P \ \mathbf{proj}\ R)) \qquad [Q \text{ local}]$$

which shows how the step in the projected formula is unwound to give the first steps $(P \wedge Q)$ followed by a simpler projection $(P \ \mathbf{proj}\ R)$. As a consequence of this we have

$$[P14] \quad (P \ \mathbf{proj}\ (\mathbf{len}(1);R)) \Leftrightarrow (P;(P \ \mathbf{proj}\ R))$$

6.6 Past Operators

A distinguishing feature of this interval logic is the inclusion of the past operators $\ominus$ and $\mathcal{S}$. First we consider the 'previous' operator which has various algebraic properties. In particular disjunction distributes over $\ominus$:

$$[Pr1] \quad (\ominus(P \vee Q)) \Leftrightarrow (\ominus P) \vee (\ominus Q)$$

and we can characterise how negation interacts with $\ominus$:

$$[Pr2] \quad \neg(\ominus P) \Leftrightarrow (\ominus(\neg P) \vee \neg(\ominus \mathbf{True}))$$

This formula states that $(\ominus P)$ fails under two circumstances: either P fails in the previous state or there is *no* previous state, characterised by $\neg(\ominus \mathbf{True})$ holding.

This means that the definition of ‘previous’ is a *strong* one: we can define a weaker version by

$$(\ominus P) \vee \neg(\ominus \mathbf{True})$$

if we wish. Alternatively we could make the weak version primitive and define the strong form from it.

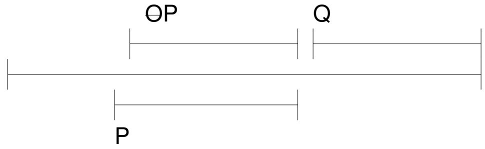
As a final simple identity we have

$$[Pr3] \quad (\ominus \mathbf{False}) \Leftrightarrow \mathbf{False}$$

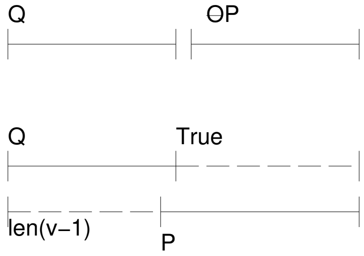
We now examine the interaction of the past time operators with sequencing through the chop operator. If the first formula in a chop is a $\ominus$ formula, we can move the operator outside the chop:

$$[Pr4] \quad ((\ominus P);Q) \Leftrightarrow \ominus(P;Q)$$

We can illustrate this thus:



In examining the case of a $\ominus$ in the second half of a chop we effectively create an overlap between two intervals, thus:



The dotted intervals in the figure also suggest how the overlap may be described.

$$[Pr5] \quad (Q;\ominus P) \Leftrightarrow \exists v \leq \mathbf{myLen}.(((Q \wedge \mathbf{myLen} = v);\mathbf{True}) \wedge (\mathbf{len}(v-1);P))$$

6.7 Temporal Duality

Our logic talks about the future and the past (as well as the present) and we can think of a duality between these two. Under this $\Box$ is dual with $\boxminus$, $\Diamond$ with $\diamondminus$ and so on. We can also introduce a measure of the past (according to the particular interval in question), **myAge**; this is the dual of **myLen**. Note that under this duality the (non-temporal) logical operations are not changed.

This duality gives a set of rules for the operators in question, and particularly we will have rules for **myAge** dual for those of **myLen**. Also dual are chop and chop in the past, $\tilde{;}$. Specifically, we have, writing an overbar for duality,

$$\overline{P\,;Q} \equiv \overline{Q}\ \tilde{;}\ \overline{P}$$

This gives rules for $\tilde{;}$ including its associativity and various relations with logical operators. The two operators interact as follows, if P is a pure (present and) future formula and $\overline{P}$ is its dual:

$$P\,;(R \,\tilde{;}\, \overline{P}) \Leftrightarrow R$$

In a similar way

$$(P;R) \;\tilde{;}\; \overline{P} \Leftrightarrow R$$

In particular we have

$$((\mathbf{myLen} = n);R) \;\tilde{;}\; (\mathbf{myAge} = n) \Leftrightarrow R$$

since the expression $(\mathbf{myLen} = n)$;S shifts S by n instants into the future and $S \;\tilde{;}\; (\mathbf{myAge} = n)$ shifts S by n instants into the past.

6.8 Filter

The filter operation is characterised by means of an expansion using normal forms for the filtered property. If we can express Q in the form of $Q_0 \wedge \bigcirc Q_1$ where Q_0 is a point (and not an interval) property, then we can say

$$P \;\; \mathbf{filter} \;\; Q \;\Leftrightarrow\; (\neg P \,\mathbf{until}\, (P \wedge Q_0 \wedge \bigcirc(P \;\; \mathbf{filter} \;\; Q_1)))$$

which unwinds the formula by one step, as it were. Note that **until** is defined in Section 4.

6.9 Characterising Determinacy and Rigidity

We can give conditions on determinacy and rigidity of formulas. For instance if we write rigid(P) and det(P) for 'P is rigid' and 'P is determinate' then we have the following heuristics

- det(P) and rigid(Q) if and only if det(P **proj** Q).
- rigid(P) and rigid(Q) if and only if rigid(P **proj** Q).
- det(P) and det(Q) implies det(P;Q).
- rigid(P) and rigid(Q) implies rigid(P;Q).

The logic given in this section is designed to be used in verification and refinement. We are also investigating issues of completeness of logical characterisation. This work will be reported elsewhere.

7 Related Work

There has been little previous work on applying temporal logic to the field of electronic publishing, or to multimedia in particular. King [11] justifies the applicability of ITL in this area and also presents [12] a number of illustrative examples. This paper extends that earlier work in a number of important ways, by giving a rigorous definition of the logic used, by considering the underlying proof theory, by considering the issue of framing and its relationship to multi-authoring, and by presenting more complex examples which treat the issue of composition in a more rigorous fashion.

Furuta and Stotts [18, 19] make use of a temporal logic for specifying and tracing dynamic link paths in hypertext documents. Their logic is somewhat different from ours, and does not require, for example, any notion of projection, and their application is also rather different.

In contrast, interval temporal logic has a relatively extensive history, e.g. [8, 13, 16, 17]. Our work builds upon this body of literature: *Mexitl* is derived from the interval temporal logic notation defined in [16]; we have used a number of the derived ITL operators defined in [8] and our proof theory is related to that of [13].

However, in terms of obtaining executability we have taken a rather different approach to other researchers. Ostensibly there are two approaches to obtaining executability. Firstly, a restricted logic (without operators such as eventually) can be used in order that specifications are deterministic and characterise just a single model. Moszkowski [16] employed this approach in defining the language Tempura. Alternatively, a richer set of logical operators can be allowed resulting in non-deterministic specifications having to be accommodated. This is the approach employed by [13] and, to take a non-interval temporal logic example, by [1]. However, the consequence of such an approach is that more complicated execution strategies must be considered, in particular, backtracking must be employed. In contrast to these two alternatives, we are considering multi-levelled development strategies in which abstract *Mexitl* specifications are refined into concrete specifications. In general terms, the abstract specifications will be expressed in the full logic, while refinement entails evolving the specification towards a deterministic executable form.

Duan's [4] approach to interval temporal logic is the closest to ours. Duan's work extends Moszkowski's interval temporal logic in a number of respects: (1) past operators are added, (2) a new projection operator is defined, (3) framing of variables is investigated, (4) infinite models are incorporated and (5) concurrency and communication primitives are considered. The last two of these do not concern us, since neither is immediately relevant to our application domain. In particular with regard to (5), we are using interval temporal logic as a scripting language to describe interaction between a single system and a user.

However, extensions (1), (2) and (3) are of interest; we consider each in turn.

1. *Past Operators.* Firstly, we have incorporated past operators for somewhat different reasons to Duan. He is motivated by the desire to give an operational

definition of assignment in the presence of framed variables. In contrast, our interest in past operators has arisen because they simplify certain of our example specifications. Importantly, all the past operators that Duan uses can be defined from our chop in the past operator which is different from the operator *Chopp* that Duan uses. These derivations are included in Section 4. In addition, Duan's chop (in the future) behaves differently to our chop in the presence of past operators in its second argument. Our reason for defining chop in the way we have is to fit with our motivatory examples and to allow $\Diamond$ and $\Box$ to be defined from it in the usual manner. Duan loses this interderivability. In addition, Duan's chop is derivable from ours as demonstrated in Section 4.

2. *Projection.* Duan defines a new projection operator, denoted *prj*. His motivation for defining this new operator is to model concurrency and interaction. Since we are not interested in such concurrent behaviour we have remained faithful to the original projection operator of Moszkowski [16] which seems much more applicable to our application area. However, we have discovered how to derive Duan's *prj* operator from the standard projection. This derivation is also included in Section 4.6.
3. *Framing of Variables.* Duan motivates the requirement to frame variables, defines a syntax for explicitly framing variables and builds mechanisms by which framing can be handled in satisfaction (in particular, derivation of minimal sets of models). We have not considered framing of variables for two main reasons. Firstly, multimedia document specifications do not seem to be rich in manipulation of data variables. Thus, if the effect of framing is required, it can be obtained using standard logical operators without cluttering specifications too seriously. Secondly, we could import Duan's approach if it was required, but, in the context of this paper, we prefer to concentrate on new aspects which arise from the multimedia documents application domain. An aspect that is new in *Mexitl* is the use of actions, hence we have considered framing of actions. Importantly, framing of actions does not carry the same overhead of theoretical machinery as framing of variables.

8 Concluding Remarks

Our paper here represents the first step in a programme of work looking towards formally based specification, refinement, prototyping and implementation of multimedia systems using interval temporal logic. We have shown how *Mexitl* formalism is given a semantics and the fundamentals of a proof theory and how to use the formalism in the development of a substantial specification example, built in a series of steps using the operations of the logic to combine parts of the specification into a coherent whole.

For the future, we intend to investigate further in a number of directions. We aim to develop the logic of *Mexitl*, working towards a system which is complete relative to the theory of Peano arithmetic. We also intend to look further at a constructive implementation of *Mexitl*, the beginnings of which are reported in [20].

The methodology supported by *Mexitl* is also a topic of active interest for us: we aim to look further at the interaction between conjunction, framing and the separate development of channels in presentations. This will come from the investigation of further case studies as well as from an investigation of the theoretical issues involved. In particular, we aim to look at higher-level languages for multimedia specification and their translation into the *Mexitl* logic reported here.

References

1. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM : A framework for programming in temporal logic. In *Lecture Notes in Artificial Intelligence, vol. 430*. Springer–Verlag, 1989.
2. H. Bowman, H. Cameron, P. King, and S. Thompson. Specification and prototyping of structured multimedia documents using interval temporal logic. In *International Conference on Temporal Logic 1997*, 1997.
3. Dick C.A. Bulterman and Lynda Hardman. Multimedia authoring tools: State of the art and research challenges. In Jan van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 575–591. Springer-Verlag, 1995.
4. Z. H. Duan. *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming*. PhD thesis, University of Newcastle Upon Tyne, May 1996.
5. Z.H. Duan, W.M. Holcombe, and D.A. Linkens. Timed interval temporal logic and modelling of hybrid systems. In *Proceedings of Modelling and Simulation, ESM'94*, june 1994.
6. Robert Erfle. Specification of temporal constraints in multimedia documents using HyTime. *Electronic Publishing*, 6(4):397–411, December 1993.
7. D. Gabbay. The declarative past and imperative future. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*. Lecture Notes In Computer Science 389, Springer-Verlag, 1989.
8. R. Hale. Using temporal logic for prototyping: the design of a lift controller. In *Lecture Notes in Computer Science, vol. 379*, pages 375–408. Springer–Verlag, 1989.
9. Lynda Hardman, Guido van Rossum, and Dick C.A. Bulterman. Structured multimedia authoring. *ACM Multimedia*, pages 283–289, 1993.
10. ISO 19744. *Information Technology – Hypermedia/Time-based Structuring Language (HyTime)*, 1992.
11. P.R. King. A logic based formalism for temporal constraints in multimedia documents. In *PODP 96*, September 1996. Revised version to appear in LNCS, Springer Verlag.
12. P.R. King. Modelling multimedia documents. *Electronic Publishing*, 8(2 and 3):95–110, 1996.
13. S. Kono. A combination of clausal and non-clausal temporal logic programs. In *Lecture Notes in Artificial Intelligence, vol. 897*, pages 40–57. Springer–Verlag, 1993.

14. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.
15. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
16. B. Moszkowski. *Executing Temporal Logic*. Cambridge University Press, 1986.
17. B. Moszkowski. Some very compositional temporal properties. In *Proceedings of Programming Concepts, Methods and Calculi, IFIP Transactions A-56*, pages 307–326. North-Holland, 1994.
18. P. David Stotts, Richard Furuta, and J. Cyrano Ruiz. Hyperdocuments as automata: Trace-based browsing property verification. In D. Lucarella, J. Nanard, M. Nanard, and P. Paolini, editors, *Proceedings of the ACM Conference on Hypertext (ECHT '92)*, pages 272–281. ACM Press, 1992. (November 30–December 4, 1992, Milan, Italy).
19. P. David Stotts, Richard Furuta, and J. Cyrano Ruiz. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *ACM Transactions on Information Systems*, 1997. To appear.
20. S. Thompson. Constructive interval temporal logic in Alf. In *International Conference on Temporal Logic 1997*, 1997.

This article was processed using the LaTeX macro package with LLNCS style