Research Report 201

A Temporal Approach to Requirements Specification of Real-Time Systems*

Yogesh Naik

RR201

This paper describes a specification notation of temporal logic to describe the requirements of real-time systems. The notation is extended by a calculus of occurrences of predicates. Using the logic and the calculus we show that common real-time properties such as durations, number of occurrences, precedence and other properties can be described. It is then used to describe the IEEE 802 Token Bus specification.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

November 1991

A Temporal Approach to Requirements Specification of Real-Time Systems*

Yogesh Naik
Department of Computer Science
University of Warwick
Coventry CV4 7AL, U.K.

Abstract

This paper describes a specification notation of temporal logic to describe the requirements of real-time systems. The notation is extended by a calculus of occurrences of predicates. Using the logic and the calculus we show that common real-time properties such as durations, number of occurrences, precedence and other properties can be described. It is then used to describe the IEEE 802 Token Bus specification.

1 Introduction

In recent years there has been considerable interest in specification of real-time systems. For a real-time system, one needs to specify requirements of responsiveness of events and satisfy timing constraints. The responsive requirement merely states a temporal relationship between events, say p and q. There is no need to specify when p and q must occur or what their durations must be. To specify timing constraints several alternative approaches have been suggested. These can roughly be grouped into two distinct approaches: *explicit clock* and *bounded operator*. In the *explicit clock* approach, no new operators are introduced. Instead to refer to time, a flexible variable T is used to denote the current value of a global clock. Various examples of this style of specification can be found in the literature, for instance RTTL [Ost89], GCTL [PH88], and XCTL [HLP90]. On the other hand, the *bounded operator* approach introduces new temporal operators to describe timing constraints. For each temporal operator (such as $\Diamond$, $\Box$) a new temporal operator is defined. For example, the temporal operator $\Diamond_{\leq\tau}\varphi$ is used to state that φ will eventually be true within τ time units from now. This approach is described in [Koy89]. A similar approach is taken in Timed CTL (TCTL) [ACD90], where for each temporal operator of Computation Tree Logic (CTL) [EC82] a new operator is defined with subscript to restrict its scope in time. There are various

*To be presented at the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems. Nijmegen, The Netherlands, January 1992

other approaches to real-time specification which donot fall into the above classification. A calculus of durations is introduced in [CHR91] to reason about timing constraints of time-critical systems. It uses a calculus of integrals of duration of states in an interval to state timing properties of a system. It is based on an interval temporal logic [MM83] in which integrals of durations of a state are considered as variables. It differs from other approaches because it doesnot make any explicit reference to time. An alternative approach where time is explicit is taken in RTL [JM86]. RTL uses an occurrence function which returns the time of occurrence of an event. Times of different events can be compared to state timing properties.

The main focus of the use of temporal logics and the need to extend its expressive power has been the specification and verification of programs [MP81, BKP84, HW89] but not in describing requirements of systems. Consider the following example.

Example : "The nodes on a token bus communicate by passing frames. The right of access to the network is regulated by using a token frame, which a node may hold for a maximum of 20 time slots. During the period a node holds the token, it can perform general maintenance by sending frames. For example, to grant an opportunity to other nodes to insert themselves in the ring, the token-holder sends a solicit-successor frame. If there is no response, it tries twice before taking any other action."

To describe the above example, one needs additional concepts and operators. For instance, the statement "a node may hold the token for a maximum of 20 time slots" requires a notion of duration of a predicate. Another example is the statement "If there is no response the node tries twice before taking any action" can be expressed more naturally if we can count the number of times the node sends the solicit-successor frame before it takes any action. Moreover, operators such as *during*, *precedes* etc. seem to be more appropriate for describing requirements.

The main contribution of this paper is to extend a temporal logic so that some additional features and operators such as *during*, duration, and number of occurrences can be easily expressed. Most specification approaches in computer science are based on the central notion that events are instantaneous. In specification of real-time systems, we abandon this notion in favour of events which may have extent in time. We can then specify common real-time properties such as duration, precedence, inclusion of events etc. We further introduce a concept of an occurrence of a predicate φ to denote a maximal interval by inclusion in which φ holds everywhere. By relating occurrences of predicates, we show that behaviours of systems can be described. This can be compared with temporal logics where an occurrence of a predicate is the time point at which it is true.

The rest of the paper is organised as follows. The syntax and the semantics of the logic is presented in the next section. The proof theory is given in Section 3. Section 4 gives the calculus of occurrences of predicates including behaviour rules for durations and number of occurrences. Section 5 takes a description of the IEEE 802 Token Bus and uses the logic to formally specify it.

2 The Temporal Framework

2.1 Syntax

The basic symbols of the language consists of constants, variables, propositions, functions and predicate symbols. We use the usual set of propositional connectives: negation ($\neg$), and implication ($\Rightarrow$) and the first-order universal quantifier ($\forall$) which is applied only to variables. The modal connectives we use are weak next ($\odot$), weak previous ($\oslash$), weak until (**U**) and weak since (**S**). Formally,

1. If φ and ψ are formulae, then so are $\neg\varphi$, and $\varphi \Rightarrow \psi$.
2. If x is a variable and φ is a formula, then $\forall x.\varphi$ is also a formula.
3. If φ and ψ are formulae, then so are $\odot\varphi$, $\varphi \mathbf{U} \psi$, $\oslash\varphi$, and $\varphi \mathbf{S} \psi$.

Other operators are defined syntactically using these connectives.

2.2 Semantics

We define a model $M = (I, \sigma, t)$, consisting of an interpretation function I, a mapping σ called behaviour and t is a time point.

- The interpretation function I specifies a nonempty set D and assigns constants, function and predicate symbols to elements in D. For a constant c, I assigns a fixed element in D, for a n-place function symbol I assigns a function $D_1 \times D_2 \ldots D_n$ into D, and for a n-place predicate symbol it assigns n-ary relation in D.
- Let T be an infinite domain of time values which is linearly ordered and closed under addition and $STATE$ be defined as a set of mappings from a set of variables VAR to a set of values VAL. i.e. it assigns values to variables.

  $$STATE = \{s | s : VAR \rightarrow VAL\}$$

  Then a behaviour σ is a mapping from T to $STATE$.

  Definition 2.1 **(Well-formedness)** In the rest of the thesis, we will consider only those behaviours which observe the following properties. These properties are essentially those which rule out the possibility of infinitely many state changes in a finite time i.e. finite variability (cf. [BKP86])

$$\forall t.((\exists t'.(t' > t \wedge \sigma(t') \neq \sigma(t))) \Rightarrow \exists t''.(t < t'' \wedge \sigma(t'') = \sigma(t') \wedge \forall t'''.(t < t''' < t'' \Rightarrow \sigma(t''') = \sigma(t)))$$

The property states that if there is a future instant in which a change in state occurs then there is a first such time point.

$$\forall t.((\exists t'.(t' < t \wedge \sigma(t') \neq \sigma(t))) \Rightarrow \exists t''.(t'' < t \wedge \sigma(t'') = \sigma(t') \wedge \forall t'''.(t'' < t''' < t \Rightarrow \sigma(t''') = \sigma(t)))$$

The second property is similar for the past instances.

- The time point $t \in T$ is the point at which the formula is being interpreted.

Given a well-formed formula A, its meaning is given inductively below. For readability, we have removed I from the definitions. The value of a term τ under M is denoted by $\tau|_t^\sigma$ with I implicit in the definition.

- For a constant

  $c|_t^\sigma = I(c)$

- For a variable

  $x|_t^\sigma = (\sigma(t))(x)$

- For an n-ary function

  $F(t_1, ..., t_n)|_t^\sigma = I(F)(t_1|_t^\sigma, ..., t_n|_t^\sigma)$

  i.e. the value given by the application of $I(F)$ to the value $(t_1, \ldots, t_n)$.

- For an n-ary predicate $\varphi(t_1, \ldots, t_n)$

  $(I, \sigma, t) \models \varphi(t_1, \ldots, t_n)$ iff $I(\varphi)(t_1|_t^\sigma, \ldots, t_n|_t^\sigma)$

  An n-ary predicate is true iff it holds at time t.

- For negation

  $(\sigma, t) \models \neg\varphi$ iff not $(\sigma, t) \models \varphi$

  $\neg\varphi$ holds iff φ does not hold.

- For implication

$(\sigma, t) \models \varphi \Rightarrow \psi$ iff $(\sigma, t) \models \varphi$ implies $(\sigma, t) \models \psi$

$\varphi \Rightarrow \psi$ holds iff φ implies ψ.

- For universal quantification

$(\sigma, t) \models \forall x.\varphi$ iff for every $d \in D$ $(\sigma, t) \models \varphi(d/x)$, where $\varphi(d/x)$ is obtained by substituting d for all free occurrences of x in φ

- For weak next

$$(\sigma, t) \models \odot\varphi \text{ iff } \exists t'.(\, t < t' \wedge (\sigma, t') \models \varphi \wedge \sigma(t) \neq \sigma(t') \wedge (\, \forall t''.(t < t'' < t' \Rightarrow \sigma(t'') = \sigma(t)) \vee \forall t''.(t < t'' < t' \Rightarrow \sigma(t'') = \sigma(t'))))) \vee \forall t'.(t < t' \Rightarrow \sigma(t') = \sigma(t))$$

$\odot\varphi$ holds now iff φ is true at some time t' in the future and the state at that instant is different from what it is now and does not change between now and t' or no such t' exists.

- For weak until

$$(\sigma, t) \models \varphi \mathbf{U} \psi \text{ iff } \exists t'.(t \leq t' \wedge (\sigma, t') \models \psi \wedge \forall t''.(t \leq t'' < t' \Rightarrow (\sigma, t'') \models \varphi)) \vee \forall t'(t \leq t' \Rightarrow (\sigma, t') \models \varphi)$$

$\varphi \mathbf{U} \psi$ is true iff there is a future instant in which ψ is true and φ holds holds continuously until that instant or φ holds in all future instances.

- For weak previous

$$(\sigma, t) \models \oslash\varphi \text{ iff } \exists t'.(\, t' < t \wedge (\sigma, t') \models \varphi \wedge \sigma(t) \neq \sigma(t') \wedge (\, \forall t''.(t' < t'' < t \Rightarrow \sigma(t'') = \sigma(t)) \vee \forall t''.(t' < t'' < t \Rightarrow \sigma(t'') = \sigma(t'))))) \vee \forall t'.(t' < t \Rightarrow \sigma(t') = \sigma(t))$$

$\oslash\varphi$ holds now iff φ is true at some time t' in the past and the state at that instant is different from what it is now and does not change in the interval between now and t' or no such t' exists.

- For weak since

$$(\sigma, t) \models \varphi \mathbf{S} \psi \text{ iff } \exists t'.(\, t' \leq t \wedge (\sigma, t') \models \psi \wedge \forall t''.(t' < t'' \leq t \Rightarrow (\sigma, t'') \models \varphi)) \vee \forall t'(t' \leq t \Rightarrow (\sigma, t') \models \varphi)$$

$\varphi \mathbf{S} \psi$ is true there is an instant in the past in which ψ is true and φ holds continuously since that instant or φ holds in all past instances.

A wff A is logically valid iff A is true for every model. This is denoted by

$$\models A$$

Following are syntactic abbreviations for temporal operators.

DF1. $\Box\varphi \triangleq \varphi \mathbf{U} false$	DP1. $\boxminus\varphi \triangleq \varphi \mathbf{S} false$
DF2. $\Diamond\varphi \triangleq \neg\Box\neg\varphi$	DP2. $\diamondminus\varphi \triangleq \neg\boxminus\neg\varphi$
DF3. $\varphi \mathcal{U} \psi \triangleq \varphi \mathbf{U} \psi \wedge \Diamond\psi$	DP3. $\varphi \mathcal{S} \psi \triangleq \varphi \mathbf{S} \psi \wedge \diamondminus\psi$
DF4. $\oplus\varphi \triangleq \neg \odot \neg\varphi$	DP4. $\ominus\varphi \triangleq \neg \oslash \neg\varphi$
DF5. $\varphi \mathcal{U}^{+} \psi \triangleq \varphi \wedge \varphi \mathcal{U} \psi$	DP5. $\varphi \mathcal{S}^{+} \psi \triangleq \varphi \wedge \varphi \mathcal{S} \psi$
DF6. $\varphi \mathbf{U}^{+} \psi \triangleq \varphi \wedge \varphi \mathbf{U} \psi$	DP6. $\varphi \mathbf{S}^{+} \psi \triangleq \varphi \wedge \varphi \mathbf{S} \psi$

3 Proof Theory

The proof theory consists of seven axioms each for propositional part of the future and past fragments.

Axioms

F1. $\vdash \Box(\varphi \Rightarrow \psi) \Rightarrow (\Box\varphi \Rightarrow \Box\psi)$	P1. $\vdash \boxminus(\varphi \Rightarrow \psi) \Rightarrow (\boxminus\varphi \Rightarrow \boxminus\psi)$
F2. $\vdash \odot(\varphi \Rightarrow \psi) \Rightarrow (\odot\varphi \Rightarrow \odot\psi)$	P2. $\vdash \oslash(\varphi \Rightarrow \psi) \Rightarrow (\oslash\varphi \Rightarrow \oslash\psi)$
F3. $\vdash \odot\neg\varphi \Rightarrow \neg \odot \varphi$	P3. $\vdash \oslash\neg\varphi \Rightarrow \neg \oslash \varphi$
F4. $\vdash \Box\varphi \Rightarrow \varphi \wedge \Box \odot \varphi$	P4. $\vdash \boxminus\varphi \Rightarrow \varphi \wedge \boxminus \oslash \varphi$
F5. $\vdash \Box(\varphi \Rightarrow \odot\varphi) \Rightarrow (\varphi \Rightarrow \Box\varphi)$	P5. $\vdash \boxminus(\varphi \Rightarrow \oslash\varphi) \Rightarrow (\varphi \Rightarrow \boxminus\varphi)$
F6. $\vdash \varphi \mathbf{U} \psi \Leftrightarrow (\psi \vee (\varphi \wedge \odot(\varphi \mathbf{U} \psi)))$	P6. $\vdash \varphi \mathbf{S} \psi \Leftrightarrow (\psi \vee (\varphi \wedge \oslash(\varphi \mathbf{S} \psi)))$
F7. $\vdash \Box\varphi \Rightarrow \varphi \mathbf{U} \psi$	P7. $\vdash \boxminus\varphi \Rightarrow \varphi \mathbf{S} \psi$

Axiom F1 states that if φ always implies ψ in the future then if φ holds in the future then so does ψ. Axiom F2 is similar to F1 for the "weak next" operator. Axiom F3 establishes that if there exists a next state in which $\neg\varphi$ holds then φ does not hold in the next state. Axiom F4 states that the present is part of the future. It also states that henceforth either φ is true in the next state or φ is always true. Axiom F5 states that if it is always the case that φ implies that in the next state φ holds then if φ holds now then φ always holds. Axiom F6 defines " weak until" recursively by stating that either ψ is true now or φ is true and in the next state $\varphi \mathbf{U} \psi$ holds. Axiom F7 states that "henceforth φ" implies $\varphi \mathbf{U} \psi$ holds. Axioms P1-P7 are symmetric to F1-F7 for the past fragment.

There are two inference rules : modus ponens, and $\Box\boxminus$ Introduction

Inference Rules

R1. Modus Ponens

$$\frac{\vdash \varphi, \vdash \varphi \Rightarrow \psi}{\vdash \psi}$$

R2. $\Box$ and $\boxminus$ Introduction

$$\frac{\vdash \varphi}{\vdash \Box\varphi \wedge \boxminus\varphi}$$

4 A Calculus of Occurrences

Let t_1 and t_2 be such that $t_1 \leq t_2$. Then, we say that intervals of the form $[t_1, t_2]$ are closed.

Definition 4.1 **(Occurrence of a predicate)** An occurrence of a predicate φ is defined as a maximal interval (by inclusion) in which φ holds everywhere in that interval.

Given a first order predicate φ, a time point t and a computation sequence σ, we define a function ρ which return a set of occurrences of a predicate φ which occur after the time point t.

$$\begin{aligned}\rho(\varphi, t, \sigma) = \{[t_1, t_2] \mid & t \leq t_1 \leq t_2 \wedge \forall t'.(t_1 \leq t' \leq t_2 \Rightarrow (\sigma, t') \models \varphi) \\ & \wedge \neg\exists t'.(t_2 < t' \wedge \forall t''.(t_2 < t'' < t' \Rightarrow (\sigma, t'') \models \varphi)) \\ & \wedge \neg\exists t'.(t \leq t' < t_1 \wedge \forall t''.(t' < t'' < t_1 \Rightarrow (\sigma, t'') \models \varphi)\}\end{aligned}$$

Given a computation sequence σ, and a time point t, we define number of occurrences of a predicate as the cardinality of the set ρ.

$$\#(\varphi, t, \sigma) = \#\rho(\varphi, t, \sigma)$$

Let ρ' be a sequence obtained from ρ such all its members are in ρ' and they are ordered by precedence relation on intervals.

We define the duration of the i^{th} occurrence of a predicate as the duration of i^{th} member of ρ'.

$$D(\varphi, i, t, \sigma) = \begin{cases} t_2 - t_1 & \text{if } [t_1, t_2] \text{ is the } i^{th} \text{ occurrence of } \varphi \\ 0 & \text{otherwise} \end{cases}$$

An occurrence predicate is a predicate on number of occurrences of predicates $\varphi_1, \ldots, \varphi_m$. If $\mathcal{N}\varphi_1, \ldots, \mathcal{N}\varphi_m$ are free variables corresponding to the number of occurrences of predicates $\varphi_1, \ldots, \varphi_m$ and $OCC[\mathcal{N}\varphi_1, \ldots, \mathcal{N}\varphi_m]$ is a general predicate over the number of occurrences, then the semantics of an occurrence predicate is given by

$$(\sigma, t) \models OCC[\mathcal{N}\varphi_1, \ldots, \mathcal{N}\varphi_m] \text{ iff } OCC[\#(\varphi_1, t, \sigma), \ldots, \#(\varphi_m, t, \sigma)]$$

We can similarly define a predicate for the duration of an occurrence of a predicate and its semantics. If $\mathcal{D}(\varphi_1, i), \ldots, \mathcal{D}(\varphi_m, j)$ are free variables corresponding to predicates $\varphi_1, \ldots, \varphi_m$ and $DUR[\mathcal{D}(\varphi_1, i), \ldots, \mathcal{D}(\varphi_m, j)]$ is a general duration predicate, the semantics of a duration predicate is given by

$$(\sigma, t) \models DUR[\mathcal{D}(\varphi_1, i), \ldots, \mathcal{D}(\varphi_m, j)] \text{ iff } DUR[D(\varphi_1, i, t, \sigma), \ldots, D(\varphi_m, j, t, \sigma)]$$

Occurrence and duration predicates are true for a model $M = (\sigma, t)$ iff M satisfies them and they are false iff M does not satisfy them. They are valid iff they are true for all models.

4.1 Behaviour Rules

The following are valid rules for occurrences of predicates in the logic.

Rule S1 states that the number of occurrences of $false$ is zero.

S1. $\mathcal{N}(false) = 0$

Rule S2 states that the number of occurrences of any predicate is non-negative.

S2. $\mathcal{N}(\varphi) \geq 0$

S3 states that the difference in the number of occurrences of a predicate and its negation is either zero or one.

S3. $0 \leq |\mathcal{N}(\varphi) - \mathcal{N}(\neg\varphi)| \leq 1$

Rule S4 states that if henceforth φ is true then the number times it occurs is one.

S4. $\Box\varphi \Rightarrow (\mathcal{N}(\varphi) = 1)$

Rule S5 states that number of occurrences any predicate either remains the same or decreases in the next state.

S5. $(\mathcal{N}(\varphi) = m \wedge \odot(\mathcal{N}(\varphi) = n) \Rightarrow (m \geq n)$

Rule S6 is similar rule to S5 for the previous state.

S6. $(\mathcal{N}(\varphi) = m \wedge \oslash(\mathcal{N}(\varphi) = n) \Rightarrow (m \leq n)$

The total duration of a predicate is defined as

$$\mathcal{T}(\varphi) \triangleq \sum_{i=1}^{\mathcal{N}(\varphi)} \mathcal{D}(\varphi, i)$$

Rule S7 states the duration of false is zero.

S7. $\forall i.\mathcal{D}(false, i) = 0$

Rule S8 states that duration of any occurrence of a predicate is greater than or equal to zero.

S8. $\forall i.\mathcal{D}(\varphi, i) \geq 0$

Rule S9 states that total duration of any two individual predicates is the sum of durations of their disjunction and their conjunction.

S9. $\mathcal{T}(\varphi) + \mathcal{T}(\psi) = \mathcal{T}(\varphi \vee \psi) + \mathcal{T}(\varphi \wedge \psi)$

Some theorems about occurrences of predicates are given in Appendix A.2.

4.2 Relating Occurrences of Predicates

Given predicates φ and ψ, we use syntactic abbreviations to relate occurrences of predicates. Five operators are defined : *occurs before*, *during*, *occurs immediately before*, *precedes* and *includes*.

$\varphi \lhd \psi$ (*occurs before*) if true iff (i) the current occurrence of φ is followed by an occurrence of ψ with no occurrences of φ in between (ii) or the current occurrence of ψ is preceded by an occurrence of φ with no occurrences of ψ in between. It is described using two disjuncts ω_1 and ω_2. ω_1 states that φ is true now and will become false and remain false until ψ becomes true.

$$\omega_1 \triangleq \varphi \wedge (\varphi \mathcal{U}(\neg\varphi \mathcal{U}((\neg\varphi \wedge \neg\psi)\mathcal{U}^{+}\psi)) \vee (\varphi \mathcal{U}((\varphi \wedge \neg\psi)\mathcal{U}^{+}(\neg\varphi \wedge \psi))))$$

ω_2 states that ψ is true now and φ has been true sometime in the past and became and remained false until ψ became true.

$$\omega_2 \triangleq \psi \wedge (\psi \mathcal{S}(\neg\psi \mathcal{S}((\neg\psi \wedge \neg\varphi)\mathcal{S}^{+}\varphi)) \vee (\psi \mathcal{S}((\psi \wedge \neg\varphi)\mathcal{S}^{+}(\neg\psi \wedge \varphi))))$$

$\varphi \lhd \psi$ is then defined as

$$\varphi \lhd \psi \triangleq \omega_1 \vee \omega_2$$

$\varphi \triangle \psi$ (*during*) is true iff the current occurrence of φ occurs during an occurrence of ψ or the current occurrence of ψ has an occurrence of φ within it. It holds iff ψ is true now and either ω_1, ω_2, or ω_3 is true.

ω_1 is true iff φ is true now and became true after ψ became true and will become false before ψ becomes false.

$$\omega_1 \triangleq \varphi \wedge (\psi \mathbf{U} \neg\varphi) \wedge (\psi \mathbf{S} \neg\varphi)$$

ω_2 is true iff φ is false now and will become true and then false before ψ becomes false.

$$\omega_2 \triangleq \neg\varphi \wedge (\psi \mathcal{U}(((\varphi \wedge \psi)\mathbf{U}^{+}\neg\varphi)))$$

ω_3 is true iff φ is false now and became true and then false before ψ becomes false.

$$\omega_3 \triangleq \neg\varphi \wedge (\psi \mathcal{S}(((\varphi \wedge \psi)\mathbf{S}^{+}\neg\varphi)))$$

$\varphi \triangle \psi$ is then defined as

$$\varphi \triangle \psi \triangleq \psi \wedge (\omega_1 \vee \omega_2 \vee \omega_3)$$

$\varphi \trianglelefteq \psi$ (*occurs immediately before*) is true iff (i) the current occurrence of φ occurs immediately before an occurrence of ψ (ii) or the current occurrence of ψ occurs immediately after an occurrence of φ. It is described using two disjuncts : ω_1 and ω_2.

ω_1 states that φ is true now and will become false when ψ becomes true.

$$\omega_1 \triangleq \varphi \wedge (\varphi \mathcal{U}((\varphi \wedge \neg\psi)\mathcal{U}^+(\neg\varphi \wedge \psi)))$$

ω_2 states that ψ is true now and became true when φ became false.

$$\omega_2 \triangleq \psi \wedge (\psi \mathcal{S}((\psi \wedge \neg\varphi)\mathcal{S}^+(\neg\psi \wedge \varphi)))$$

$\varphi \trianglelefteq \psi$ is then defined as

$$\varphi \trianglelefteq \psi \triangleq \omega_1 \vee \omega_2$$

Precedence ($\prec$) and inclusion ($\sqsubset$) are then defined using the following abbreviation.

$$\textbf{always}\varphi \triangleq \Box\varphi \wedge \boxminus\varphi$$

Precedence ($\prec$) is defined as

$$\varphi \prec \psi \triangleq \textbf{always}(\varphi \Rightarrow \varphi \triangleleft \psi) \wedge \textbf{always}(\psi \Rightarrow \varphi \triangleleft \psi)$$

and *inclusion* ($\sqsubset$) as

$$\varphi \sqsubset \psi \triangleq \textbf{always}(\varphi \Rightarrow \varphi \triangle \psi) \wedge \textbf{always}(\psi \Rightarrow \varphi \triangle \psi)$$

Precedence is transitive while inclusion is transitive and reflexive.

Theorems related to the above operators are given in the Appendix A.1 .

5 Example

The example we are concerned with is described below. The informal specification is originally from [Pic86].

5.1 The Specification Statement

The token bus is a technique used for controlling access to a communication bus. Nodes on the bus are assigned logical positions in an ordered sequence so that they form a logical ring. Each node is assigned an address. All active nodes know the address of the active node preceding it and the one following it. The last member of the logical ring is followed by the

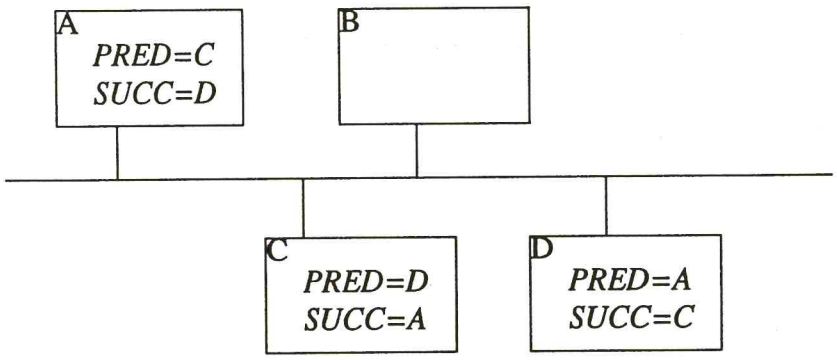

Figure 1: Token Bus

first. The physical position of a station on the bus is irrelevent and independent of the logical ordering. Figure 1 illustrates the bus network.

Communication between nodes is achieved by passing frames. There are six types of frames : token, solicit-successor, set-successor, resolve-contention, who-follows and claim-token. The token frame is used for regulating the right of access to the bus. It contains a destination address and the node receiving the token is granted access to the bus for a specified time after which it passes the token frame to the next node in the logical order. During the period a node holds the token, it can transmit frames, it may also drop itself from the ring, poll other nodes and send messages. An inactive node can only respond to polls or request for acknowledgement. As part of maintenance, the token bus provides following functions.

1. *Deletion of a node.* A node may voluntarily remove itself from the ring.
2. *Addition of a node.* Periodically, inactive nodes are granted an opportunity to insert themselves in the ring.
3. *Fault management.* A number of errors can occur. These include duplicate addresses (two nodes think it is their turn) and a broken ring (no node thinks that it is its turn).
4. *Ring initialisation.* When the ring is started up, or after the logical ring has broken down, it is reinitialised. A decentralised algorithm is used to sort out the logical order.

5.2 Formalisation of Requirements

There are six types of frames.

$$frame : \{token, sol\_succ, set\_succ, res\_cont, who\_follows, claim\}$$

Each node in the ring knows the address of its logical successor and predecessor. That is, given an address of a node, we define functions, $succ$ and $pred$ which return addresses of a node's successor and predecessor respectively.

$$succ : address \rightarrow address$$
$$pred : address \rightarrow address$$

We also denote the period during which a node A holds the token by a predicate $H(A)$. $H(A)$ is true iff a node with an address A holds the token and false otherwise. A node may send frames to other nodes. $S(F, A, X)$ is true iff the node with address A sends a frame F to a node with address X.

Predicates

$H(A)$: node with address A holds the token
$S(F, A, X)$: node with address A sends a frame F to a node with address X

We include the following general requirement about the duration each node may hold the token where c is some constant.

$$\forall x, i.\mathcal{D}(H(x), i) \leq c$$

5.2.1 Deletion of a node

If a node wishes to drop out, it waits until it receives the token, then sends a set-successor frame to its predecessor, instructing it to change its successor to be the token holder's successor.

Figure 2 illustrates the operation of allowing a node to drop out of the ring, using a timing diagram . It is formalised in stages below.

If a node wishes to drop out, it waits until it receives the token, then sends a set-successor frame to its predecessor.

$$S(set\_succ, A, pred(A)) \triangle H(A)$$

The predecessor splices to the token holder's successor after receiving the set-successor frame.

$$S(set\_succ, A, pred(A)) \lhd (succ(pred(A)) = succ(A))$$

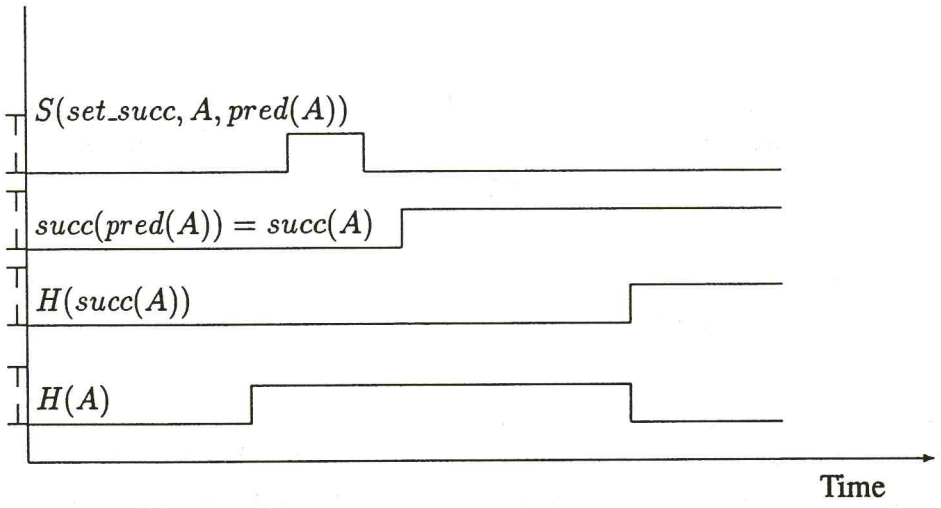

Figure 2: Timing diagram for the delete operation

The token holder passes the token to its logical successor.

$$H(A) \trianglelefteq H(succ(A))$$

The delete operation can now be described:

$$\begin{aligned} delete \triangleq \Box(&S(set\_succ, A, pred(A)) \Rightarrow \\ &(S(set\_succ, A, pred(A)) \triangle H(A) \\ &\wedge S(set\_succ, A, pred(A)) \triangleleft (succ(pred(A)) = succ(A)) \\ &\wedge H(A) \trianglelefteq H(succ(A)))) \end{aligned}$$

5.2.2 Addition of a node

Each node in the ring has the responsibility of periodically granting inactive nodes to enter the ring. While holding the token, the node issues a solicit-successor frame, inviting inactive nodes with an address between itself and the next node in the logical sequence to demand entrance. Let the set of nodes between the node A and the next node in the logical sequence be

$$I(A) \triangleq \{y | A < y < succ(A)\}$$

The token holder sends a solicit-successor frame to all the members of the above set.

$$\begin{aligned} send\_sol &\triangleq \forall x \in I(A).S(sol\_succ, A, x) \\ solicit &\triangleq send\_sol \Rightarrow (send\_sol \triangle H(A)) \end{aligned}$$

The transmitting node then waits for a response window. We denote a node A waiting for a response window by a predicate $W(A)$.

$$wait \triangleq send\_sol \Rightarrow (send\_sol \trianglelefteq W(A) \wedge W(A) \triangle H(A))$$

The duration of the wait period is 1 time slot.

$$\forall x, i.\mathcal{D}(W(x), i) = 1$$

Three events can occur.

1. *No response.* Nobody wants to enter the ring, in which case the token is transferred to the logical successor.

$$no\_resp \triangleq (\neg\exists x \in I(A).(S(set\_succ, x, A) \triangle W(A)) \wedge send\_sol \trianglelefteq W(A) \wedge W(A)) \Rightarrow H(A) \trianglelefteq H(succ(A))$$

2. *One response.* There is exactly one response by issuing set-successor frame. The token holder sets its successor node to be the requesting node and transmits the token to it. The requestor sets its linkages accordingly and proceeds.

$$one\_resp \triangleq \exists!x \in I(A).(S(set\_succ, x, A) \triangle W(A) \wedge send\_sol \trianglelefteq W(A) \wedge W(A)) \Rightarrow (H(A) \trianglelefteq H(x) \wedge (S(set\_succ, x, A) \wedge succ(A) = y) \triangleleft (succ(A) = x \wedge pred(x) = A \wedge succ(x) = y))$$

3. *Multiple Responses.* The token holder will detect a garbled response if more than one node demands entrance. The conflict is resolved as follows. The token holder transmits a resolve-contention frame and waits four demand windows.

$$send\_res \triangleq \forall x \in I(A).S(res\_cont, A, x)$$

We denote a node A waiting in a demand window by following state predicates.

$W_1(A)$: Node with address A waiting in demand window 1
$W_2(A)$: Node with address A waiting in demand window 2
$W_3(A)$: Node with address A waiting in demand window 3
$W_4(A)$: Node with address A waiting in demand window 4

The duration of each demand window is 1 time slot.

$$\forall x, i.\mathcal{D}(W_1(x), i) = \mathcal{D}(W_2(x), i) = \mathcal{D}(W_3(x), i) = \mathcal{D}(W_4(x), i) = 1$$

Each demanding node can respond in one of these windows. If a demanding node hears anything before its window comes up, it refrains from demanding. If the token holder receives a valid set-successor frame, it is in business. Otherwise it tries again, and only those nodes that responded the first time are allowed to respond this time. This process of resolving is defined as

$$\begin{aligned} resolve \triangleq\ & (W_1(A) \trianglelefteq W_2(A) \vee W_2(A) \trianglelefteq W_3(A) \vee W_3(A) \trianglelefteq W_4(A)) \\ & \wedge W_1(A) \triangle H(A) \wedge W_2(A) \triangle H(A) \wedge W_3(A) \triangle H(A) \\ & \wedge W_4(A) \triangle H(A) \\ & \wedge (set\_w1 \Rightarrow \neg(set\_w2 \wedge set\_w3 \wedge set\_w4)) \\ & \wedge (set\_w2 \Rightarrow \neg(set\_w3 \wedge set\_w4)) \\ & \wedge (set\_w3 \Rightarrow \neg set\_w4) \end{aligned}$$

set_w1, set_w2, set_w3, and set_w4 are defined as follows.

$$\begin{aligned} set\_w1 &\triangleq \exists x \in I(A).(S(set\_succ, x, A) \triangle W_1(A)) \\ set\_w2 &\triangleq \exists x \in I(A).(S(set\_succ, x, A) \triangle W_2(A)) \\ set\_w3 &\triangleq \exists x \in I(A).(S(set\_succ, x, A) \triangle W_3(A)) \\ set\_w4 &\triangleq \exists x \in I(A).(S(set\_succ, x, A) \triangle W_4(A)) \end{aligned}$$

The process is continued until one of the three events occur.

(a) A valid set-successor frame is received, and the token is passed.

$$\begin{aligned} valid \triangleq\ & \exists! x \in I(A).(S(set\_succ, x, A) \wedge S(set\_succ, x, A) \triangle W_1(A) \\ & \wedge H(A) \trianglelefteq H(x) \wedge (S(set\_succ, x, A) \wedge succ(A) = y) \triangleleft \\ & (succ(A) = x \wedge pred(x) = A \wedge succ(x) = y))) \end{aligned}$$

(b) No response is received and the token is passed to the logical successor.

$$zero\_resp \triangleq W_1(A) \wedge \neg set\_w1 \wedge H(A) \trianglelefteq H(succ(A))$$

(c) The maximum retry count is reached and the token is passed to the logical successor. Let x be the number of retries when the resolve-contention frame is sent and max be the number of retries allowed.

$$\begin{aligned} max\_resp \triangleq\ & H(A) \trianglelefteq H(succ(A)) \wedge H(succ(A)) \\ & \wedge \mathcal{N}(W_1(A)) = x - max \end{aligned}$$

Let two or more nodes responding in the response window be defined as

$$\begin{aligned} many \triangleq\ & \exists x, y.(S(set\_succ, x, A) \triangle W(A) \wedge S(set\_succ, y, A) \triangle W(A) \\ & \wedge x \neq y) \end{aligned}$$

Condition	Action
Multiple Tokens	Drop the token
Unaccepted Token	Retry
Failed Node	"Who follows" process
Failed Receiver	Drop out of the ring
No Token	Initialize after timeout

Table 1: Fault Management

mult_resp is then defined as

$$mult\_resp \triangleq (W(A) \wedge send\_sol \trianglelefteq W(A) \wedge many \wedge \mathcal{N}(W_1(A)) = x) \Rightarrow (send\_res \triangle H(A) \wedge W(A) \trianglelefteq (send\_res \wedge (resolve\mathcal{U}(valid \vee zero\_resp \vee max\_resp)))$$

The complete operation is as described below.

$$add \triangleq \Box(solicit \wedge wait \wedge (no\_resp \vee one\_resp \vee mult\_resp))$$

5.2.3 Fault Management

Fault Management by the token holder covers a number of contingencies, see Table 1.

Multiple Tokens

While holding the token, a node may hear a frame indicating that another node has the token. If so, it immediately drops the token by reverting to listener mode. In this way, the number of token holders drops immediately to one or zero, thus overcoming the multiple-token problem.

Let *mult_token* be defined as

$$mult\_token \triangleq H(A) \wedge H(B) \wedge A \neq B$$

then

$$\Box(mult\_token \Rightarrow mult\_token\mathcal{U}(\neg H(A) \vee \neg H(B)))$$

Unaccepted Token

Upon completion of its turn, the token holder will issue a token frame to its successor. The successor should immediately issue a data or token frame. Therefore, after sending a token, the token issuer will listen for one time slot to make sure that its successor is active. This precipitates a sequence of events:

1. If the successor node is active, the token issuer will hear a valid frame and revert to listener mode. If the issuer does not hear a valid frame, it reissues the token to the same successor one more time.

$$\begin{aligned} issue\_token \triangleq & (W(A) \wedge H(A) \trianglelefteq W(A) \wedge S(token, A, x) \trianglelefteq W(A) \\ & \wedge \neg(\exists f, y.(S(f, x, y)) \triangle W(A))) \Rightarrow W(A) \trianglelefteq S(token, A, x) \end{aligned}$$

2. After issueing the token the second time, the token issuer may hear a valid frame in which case it is in business or it assumes that its successor has failed and issues a who-follows frame, asking for the identity of the node that follows the nonresponding node.

$$\begin{aligned} token\_again \triangleq & (W(A) \wedge \neg(H(A) \trianglelefteq W(A)) \wedge S(token, A, x) \trianglelefteq W(A) \\ & \wedge \neg(\exists f, y.(S(f, x, y)) \triangle W(A))) \\ & \Rightarrow W(A) \trianglelefteq \forall y.S(who\_follows, A, y) \end{aligned}$$

3. The issuer should get a set-successor frame from the second node down the line. If so, the issuer adjusts its linkage and issues a token (back to step 1). If the issueing node gets no response to its who-follows frame, it tries again.

   We define

$$\begin{aligned} & send\_token \triangleq S(token, A, succ(succ(A))) \\ & send\_who \triangleq \forall y.S(who\_follows, A, y) \\ & send\_set \triangleq S(set\_succ, succ(succ(A)), A) \\ & first\_who \triangleq (W(A) \wedge send\_token \trianglelefteq W(A)) \trianglelefteq send\_who \wedge send\_who \end{aligned}$$

   then

$$\begin{aligned} issue\_who \triangleq & W(A) \wedge first\_who \trianglelefteq W(A) \\ & \wedge((\neg(send\_set \trianglelefteq W(A)) \Rightarrow W(A) \trianglelefteq send\_who) \\ & \vee(send\_set \trianglelefteq W(A) \\ & \Rightarrow W(A) \triangleleft (succ(A) = succ(succ(A)))) \end{aligned}$$

4. After issueing the who-follows frame the second time, it may hear a valid set-successor frame in which case it is in business, or it fails and the node issues a solicit-successor frame to all the nodes.

   We define sol_all and sec_who as

$$sol\_all \triangleq \forall y.S(sol\_succ, A, y)$$
$$sec\_who \triangleq (W(A) \wedge send\_who \trianglelefteq W(A)) \trianglelefteq send\_who \wedge send\_who$$

then

$$\begin{aligned} who\_again \triangleq\ & W(A) \wedge send\_who \trianglelefteq W(A) \\ & \wedge((\neg(send\_set \trianglelefteq W(A)) \Rightarrow W(A) \trianglelefteq sol\_all) \\ & \quad \vee(send\_set \trianglelefteq W(A) \\ & \quad\quad \Rightarrow send\_set \triangleleft (succ(A) = succ(succ(A)))) \end{aligned}$$

5. The issuer should get a set-successor frame from one of the nodes. If so, it adjusts its linkages accordingly and issues a token. If the issueing node does not get a response then it tries again.

   We define $first\_sol$ as

$$first\_sol \triangleq (W(A) \wedge send\_who \trianglelefteq W(A)) \trianglelefteq sol\_all \wedge sol\_all$$

   then $issue\_solicit$ is defined as

$$\begin{aligned} issue\_solicit \triangleq\ & W(A) \wedge first\_sol \trianglelefteq W(A) \\ & \wedge((\neg(\exists!x.S(set\_succ, x, A) \triangle W(A)) \Rightarrow W(A) \trianglelefteq sol\_all) \\ & \quad \vee(\exists!x.S(set\_succ, x, A) \triangle W(A) \\ & \quad\quad \Rightarrow S(set\_succ, x, A) \triangleleft (succ(A) = x))) \end{aligned}$$

6. If it fails, it tries again. If it succeeds then it adjusts its linkages. If after two attempts , the node assumes that a catastrophe has occurred; perhaps the node's receiver has failed. In any case, the node ceases activity and listens to the bus.

   We define $second\_sol$ as

$$second\_sol \triangleq (W(A) \wedge sol\_all \trianglelefteq W(A))sol\_all \wedge sol\_all$$

$$\begin{aligned} solicit\_again \triangleq\ & W(A) \wedge second\_sol \trianglelefteq W(A) \\ & \wedge(\exists!x.S(set\_succ, x, A) \triangle W(A) \\ & \quad \Rightarrow S(set\_succ, x, A) \triangleleft (succ(A) = x)) \end{aligned}$$

The complete operation can be described as follows

$$\begin{aligned} unacc\_token \triangleq\ & \Box(issue\_token \wedge token\_again \wedge issue\_who \wedge who\_again \\ & \wedge issue\_solicit \wedge solicit\_again) \end{aligned}$$

5.2.4 Ring Initialisation

Logical ring initialisation occurs when one or more nodes detect a lack of bus activity of duration longer than a time-out value: the token has been lost. This can result from a number of causes, such as the network has just been powered up, or a token-holding node fails. Once its time expires, a node will issue a claim-token frame. After transmission, a claimant listens to the medium and if it hears anything, drops its claim. Otherwise it tries again. The process repeats until a node succeeds. It considers itself the token-holder. The ring can now be rebuilt by the response window process as described in the operation to add nodes.

We define a predicate $T(A)$ which holds whenever A is listening to the bus for some activity. Its duration is t time slots.

$$\mathcal{D}(T(A), i) = t$$

Then *no_act* defines the period there is no activity on the bus.

$$no\_act \triangleq \neg(\exists f, x, y.S(f, x, y) \triangle T(A)) \Rightarrow T(A) \lhd send\_claim$$

We define *send_claim* as

$$send\_claim \triangleq \forall y.S(claim, A, y)$$

and the fact no other node claims the token as

$$no\_other \triangleq send\_claim \trianglelefteq W(A) \wedge W(A) \wedge \neg(\exists f, x, y.S(f, x, y) \triangle W(A))$$

If no other node claims the token, then it considers itself the token holder.

$$claim \triangleq no\_other \Rightarrow W(A) \lhd H(A)$$

The complete operation can be described as follows :

$$ring\_init \triangleq \Box(no\_act \wedge claim)$$

6 Conclusion

The logic described in this paper is a temporal logic without stuttering. It departs from the usual treatment by abandoning the central concept in most specification approaches that events are instantaneous. There are many examples where one needs to consider events which extend in time. For example, the delay action can not be described as instantaneous but has a duration. Properties described using first order logic also have duration. Occurrences of predicates were defined and used to describe behaviours of systems using operators such as *during*, *immediately before* etc. A calculus of occurrences of predicates was introduced in Section 4 to reason about the duration and number of occurrences.In Section 5, the IEEE 802 Token Bus was specified in the logic.

The use of intervals in requirements specifications in not new : for example, Zhou ChaoChen, Hoare and Ravn [CHR91] use an interval logic extended with a calculus of durations to specify timing properties, and RTL [JM86] to describe and analyse systems.

In duration calculus, timing constraints are described using integrated durations of states in an interval. A duration of a state predicate φ in a closed interval $[b, e]$ is defined by an integral $\int_b^e \varphi(t)dt$ and is denoted by a variable $\int\varphi$. It is easy to see that $\int true$ in any interval $[b, e]$ is $e - b$ and $\int false$ is zero. Given this definition, axioms are given which relate durations of different predicates and theorems proved from them.

A further notation is introduced which converts a state predicate to an interval one ($[\varphi]$). $[\varphi]$ is defined as

$$[\varphi] \triangleq \int\varphi = \int true \wedge \int true > 0$$

That is, $[\varphi]$ holds in an interval iff it holds throughout a nonempty interval. Point interval is defined as

$$[\ ] \triangleq \int true = 0$$

The definition of $[\varphi]$ differs from an occurrence of φ in one respect. While $[\varphi]$ holds for any interval where φ holds throughout, an occurrence of φ denotes a maximal interval (by inclusion) where φ holds everywhere.

A further modal operator *chop* ($\varphi \,\hat{}\, \psi$) is introduced and defined in an interval which can be subdivided into two sub-intervals of which in the first φ holds and in the second ψ holds. Interval temporal operators *eventually* and *henceforth* are defined in terms of *chop*. The resulting logic is very expressive for stating timing requirements using durations. For example, the formula

$$\int true = 60 \Rightarrow \int busy(m) \leq 50$$

asserts that in any given hour, m is not busy for more than 50 minutes.

In RTL, the event-action model captures timing constraints which are transformed into an RTL formula. An RTL formula is formed using constants and an occurrence function. There are three kinds of constants: actions, events and integers. The occurrence function @ is introduced to capture the notion of time. Given an event e and a non negative integer i, the @ function returns the time of the i^{th} occurrence of e. Using the occurrence function, one can specify requirements such as periodicity and duration of events. For example, to state that an action a occurs periodically every c time units, a formula of the form

$$\forall i \geq 1.@(\uparrow a, i+1) - @(\uparrow a, i) = c$$

is used. $\uparrow a$ and $\downarrow a$ are used to denote starting and finishing events marking start and finish of the action a respectively.

Duration of actions is specified similarly. The formula

$$\forall i \geq 1.@(\downarrow a, i) - @(\uparrow a, i) = c$$

asserts that the time of finishing of an action a is exactly c time units after it started.

Further notations are used to denote that a state predicate is true in an interval. For example, $\varphi[t_1, t_2]$ asserts that φ becomes true at t_1 and false at t_2, and $\varphi[t_1, t_2 >$ states that φ becomes true at t_1 and false after t_2. $\varphi[t_1, t_2]$ is the maximal interval φ is true everywhere and therefore (in our terminology) defines an occurrence of φ. However, there are two major differences between RTL and our approach. Firstly, RTL does not have any temporal operators and secondly, we donot refer explicitly to time.

This paper describes only preliminary ideas on using temporal logic to specify requirements. One of the aspects of future work is to investigate whether the primitives defined here can specify large class of real-time systems. Also an area of investigation is the relationship between the logic and programming languages such as a real-time extension of CSP.

Acknowledgements : I would like to thank my supervisor, Mathai Joseph for detail comments on earlier drafts and to Asis Goswami and Liu Zhiming for numerous technical discussions and ideas . Thanks are also due to the referees for detail technical comments and improvements.

A Theorems

A.1 Theorems about $\triangle$, $\trianglelefteq$, and $\triangleleft$

Following are some useful theorems stated without proofs.

I1. $\vdash \varphi \lhd \psi \Rightarrow (\varphi \vee \psi)$
I2. $\vdash \varphi \triangle \psi \Rightarrow \psi$
I3. $\vdash \varphi \trianglelefteq \psi \Rightarrow \varphi \lhd \psi$
I4. $\vdash \neg((\neg\varphi) \triangle \varphi)$
I5. $\vdash \varphi \triangle \psi \wedge \psi \triangle \omega \Rightarrow \varphi \triangle \omega$
I6. $\vdash \varphi \triangle (\psi \wedge \omega) \Rightarrow \varphi \triangle \psi \wedge \varphi \triangle \omega$
I7. $\vdash \varphi \triangle (\neg\psi) \Rightarrow \neg(\varphi \triangle \psi)$
I8. $\vdash (\varphi \vee \psi) \triangle \omega \Rightarrow \varphi \triangle \omega \vee \psi \triangle \omega$
I9. $\vdash \varphi \triangle \psi \vee \varphi \triangle \omega \Rightarrow \varphi \triangle (\psi \vee \omega)$
I10. $\vdash \neg(\varphi \trianglelefteq \varphi)$
I11. $\vdash \varphi \trianglelefteq (\psi \trianglelefteq \omega) \Leftrightarrow (\varphi \trianglelefteq \psi) \trianglelefteq \omega$
I12. $\vdash \varphi \trianglelefteq \psi \wedge \varphi \trianglelefteq \omega \Rightarrow \varphi \trianglelefteq (\psi \wedge \omega)$
I13. $\vdash (\varphi \vee \psi) \trianglelefteq \omega \Rightarrow \varphi \trianglelefteq \omega \vee \psi \trianglelefteq \omega$
I14. $\vdash \varphi \trianglelefteq (\psi \vee \omega) \Rightarrow \varphi \trianglelefteq \psi \vee \varphi \trianglelefteq \omega$
I15. $\vdash (\neg\varphi) \trianglelefteq \psi \Rightarrow \neg(\varphi \trianglelefteq \psi)$
I16. $\vdash \varphi \trianglelefteq (\neg\psi) \Rightarrow \neg(\varphi \trianglelefteq \psi)$
I17. $\vdash \neg(\varphi \lhd \varphi)$
I18. $\vdash (\varphi \vee \psi) \lhd \omega \Rightarrow \varphi \lhd \omega \vee \psi \lhd \omega$
I19. $\vdash \varphi \lhd (\psi \vee \omega) \Rightarrow \varphi \lhd \psi \vee \varphi \lhd \omega$

A.2 Theorems about Occurrence of Predicates

Following are some useful theorems stated without proofs.

M1. $\mathcal{T}(\varphi) + \mathcal{T}(\neg\varphi) = \mathcal{T}(true)$
M2. $\mathcal{T}(\varphi) \leq \mathcal{T}(true)$
M3. $\Box(\varphi \Rightarrow \psi) \Rightarrow \Box(\mathcal{T}(\varphi) \leq \mathcal{T}(\psi))$

References

[ACD90] R. Alur, C. Courcoubetis, and D. L. Dill. Model checking for real-time systems. In *Proceedings Symposium on Logic in Computer Science*, 1990.

[BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proceedings of the 16th ACM Symposium on the Theory of Computing*, pages 51–63, Washington D.C., 1984.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pages 173–183, Florida, 1986.

[CHR91] Z. ChaoChen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. Unpublished, 1991.

[EC82] E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[HLP90] D. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proceedings Symposium on Logic in Computer Science*, pages 402–413, 1990.

[HW89] J. Hooman and J. Widom. A temporal-logic based compositional proof system for real-time message passing. In *Lecture Notes in Computer Science 366*, pages 424–441. Springer-Verlag, Heidelberg, 1989.

[JM86] F. Jahanian and A. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12:890–904, 1986.

[Koy89] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, 1989.

[MM83] B. Moszkowski and Z. Manna. Reasoning in interval temporal logic. In *Lecture Notes in Computer Science 164*, pages 371–383. Springer-Verlag, Heidelberg, 1983.

[MP81] Z. Manna and A. Pnueli. Verification of concurrent programs: The temporal framework. In R.S. Boyer and J.S. Moore, editors, *The Correctness Problem in Computer Science*, pages 215–273. Academic Press, London, 1981.

[Ost89] J.S. Ostroff. *Temporal Logic for Real-time Systems*. Research Studies Press, 1989.

[PH88] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real time systems (extended abstract). In *Lecture Notes in Computer Science 331*, pages 84–98. Springer-Verlag, Heidelberg, 1988.

[Pic86] R. L. Pickholtz, editor. *Local Area Networks and Multiple Access Networks*, chapter 1, pages 1–30. Computer Science Press, 1986.